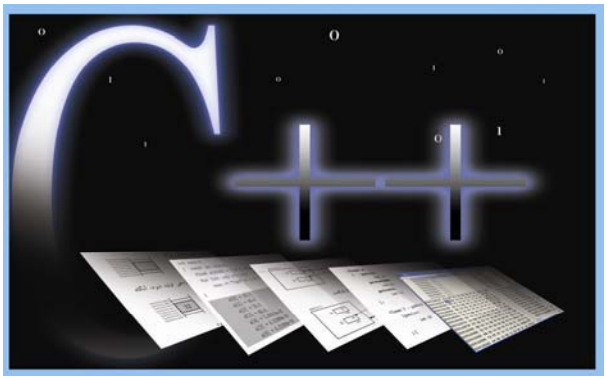
برنامه سازی پیشرفته



دانشگاه پیام نور

«برنامه‌سازی پیشرفته»

منبع درسی رشته کامپیوتر

تالیف و تدوین: دکتر احمد فراهی

مقدمه

دانشگاه آن زمان توانست شانه‌های خود را از سنگینی استعلای غربی سبک سازد و راه به سوی دامنه‌های موفقیت بگشاید که مدد نیروهای جوان و عمل‌گرای خود را پذیرفت و با تکیه بر اندیش ۀ دانشمندان و فرهیختگان دلسوز، رو به جلو حرکت کرد، چه حرکت نتیج‌ۀ بازاندیشی و خواست تغییر است. این خواست مهم است نه فقط به خاطر آن که جنبشی از درون و برای بیرون‌رفت از شرایط حکومت غرب بر دانشگاه بود بلکه به آن جهت که دانشگاه نه در دوران جنگ تحمیلی و نه پس از آن در لابه‌لای برنامه‌های سازندگی و تب و تاب نوپایی سیاسی از زمان خود عقب نماند. زمانی که عزم بر جبران و تکاپو باشد، سستی و کاستی به بالندگی و غرور می‌انجامد و شرایط نامتعادل اجتماعی به نظام‌های علمی و فرهنگی. به سادگی نمی‌توان افق دانش‌هایی چون انرژی اتمی و نانوتکنولوژی را درنوردید؛ تلاش مضاعف می‌طلبد و همگانی. همان گونه که راهبر فرزانه گفت:

«اگر همه تلاش کنند قله‌های علوم و فنون به دست ایرانیان فتح خواهد شد»

اکنون نهضت دیگری به جوشش درآمده است و مسیر جریانش را می‌جوید. نهضتی بر آمده از درد ... «جنبش نرم‌افزاری و نهضت تولید علم». منابع علمی غنی و در خور، از مهم‌ترین ملزوماتی است که پیشروی علمی را هموار می‌سازد، خاصه این که دانشجویان دانشگاه پیام نور به جبر نظام آموزش از دور از پرتو «استاد» کم‌تر بهره می‌برند. لازم است برای غنی‌سازی منابع درسی، فن‌آوری آموزشی به بهترین شکل به خدمت گرفته شود تا دانشجویان در سای‌ۀ تلاش مضاعف، با اطمینان و آسایش بیشتری در وادی علم ره سپارند. کتاب حاضر در همین راستا، منبع درسی واحد «برنامه‌سازی پیشرفته» برای رشت‌ۀ کامپیوتر وضع شده. کوشیده‌ایم محک «مصوب‌ۀ شورای عالی انقلاب فرهنگی» را به دست گیریم و با نیم‌نگاهی به منابع درسی دانشگاه‌های معتبر جهان و با اتکا به محاسن و نواقص منبع قبلی، طرحی نو در اندازیم. شیوۀ نگارش به گونه‌ای است که دانشجو خود را در کلاس و در نزد استاد حاضر می‌بیند. انتخاب مثال‌ها به شکلی است که هم کوتاه و موجز باشند و هم در هر کدام نکات ریز و درشت آموزشی لحاظ شده باشد. همچنین جلوه‌های بصری متن، خواننده را یاری می‌دهد تا تمرکز خویش را روی موضوع مورد مطالعه از دست ندهد.

این کتاب نه فقط زبان C++ را، بلکه برنامه‌نویسی را آموزش می‌دهد، به هر زبان که باشد. به نظرات و پیشنهادات اساتید و دانشجویان صادقانه ارج می‌نهیم و دست‌بوس انتقادها و بیان کاستی‌ها هستیم.

هزاران سپاس الطاف الهی را که در تدارک این مجموعه توفیق‌مان داد.

راهنمای مطالعهٔ کتاب

– از حجم زیاد کتاب نهراسید. بیشتر حجم کتاب مربوط به شرح برنامه‌ها و مثال‌های گوناگون است که شما را در یادگیری مطالب یاری می‌کنند.

– فصول کتاب کاملا با یکدیگر مرتبط‌اند و از مطالب بخش‌های قبلی به وفور در بخش‌های بعدی استفاده شده است. پس سعی کنید درس به درس و همراه با کتاب پیش بروید.

– اگر از قبل با زبان برنامه‌نویسی دیگری(مانند پاسکال) آشنایی و مهارت مختصر داشته باشید، فصل‌های اول تا هشتم را به سرعت فرا خواهید گرفت. کافی است یک بار به طور عمیق این فصول را مطالعه نمایید و سپس به راحتی می‌توانید تمرین‌های آخر این فصول را حل کنید.

– فصل‌های هشتم تا آخر (مخصوصا فصل نهم تا یازدهم که مباحث مربوط به شی‌گرایی است) را بیشتر مطالعه کنید و زمان بیشتری برای این فصول منظور کنید. ممکن است مجبور شوید این فصول را بیش از یک بار مطالعه و مرور کنید اما جای نگرانی نیست. این اشکال مربوط به ماهیت این فصل‌ها است که تا کنون راجع به آن‌ها کمتر خوانده یا شنیده‌اید. مایوس نشوید و با جدیت مطالعه کنید و یقین داشته باشید که مطلب پیچیده‌ای در این فصل‌ها وجود ندارد.

– هنگام مطالعه به نحوهٔ حروفچینی برنامه‌ها دقت کنید. مطالبی که در حال فراگیری آن هستید با حروف تیره‌تر در هر برنامه نوشته شده تا به راحتی منظور برنامه را درک نمایید و نکات آن را فرا بگیرید. بعد از هر برنامه شرح نحوهٔ کار کردن آن نیز آمده است تا ابهامات احتمالی را از بین ببرد.

– هر چند خروجی هر برنامه در چهارگوش تیره رنگ نشان داده شده اما سعی کنید خودتان برنامه‌ها را روی رایانه نوشته و اجرا کنید تا از مطالعهٔ این درس لذت بیشتری ببرید. در این رهگذار ممکن است به مطالب آموزشی جالبی برخورد کنید که در کتاب نیامده است.

– اندازهٔ برنامه‌ها متناسب با فصول افزایش می‌یابد. برنامه‌های فصل‌های آغازین کوتاه‌تر و برنامه‌های فصل‌های پایانی طولانی‌ترند. اگر در فهم برنامه‌های طولانی با مشکل مواجه شدید احتمالا فصل‌های آغازین کتاب را به خوبی فرا نگرفته‌اید.

– کوشش کنید تا همهٔ تمرین‌ها و پرسش‌ها را حل کنید. در تمرین‌ها و مسایل کتاب، گنج‌هایی نهفته است که فقط دانشجویان سخت‌کوش به آن‌ها دست می‌یابند.

– ضمیمه‌های کتاب به هیچ عنوان جنبهٔ امتحانی ندارند و فقط برای مطالعهٔ بیشتر ذکر شده‌اند. دانشجویان علاقمند می‌توانند با مطالعهٔ ضمیمه‌ها دانش برنامه‌نویسی بالاتری کسب کنند.

– بهتر است بین یک تا دو ساعت به طور پیوسته مطالعه کنید تا بهتر یاد بگیرید و کمتر مجبور شوید به مطالبی که دفعهٔ قبل مطالعه کرده‌اید، بازگردید. سعی کنید در آرامش به مطالعه بپردازید و عواملی که تمرکزتان را می‌کاهند به حداقل برسانید. به اندازهٔ کافی استراحت کنید تا خستگی اثر سویی روی اندوخته‌هایتان نگذارد.

– یاد خدا و ذکر الطاف او و امید به رحمتش آرامش‌بخش روح و روان است. به او توکل کنید و امور خویش را به او بسپارید و از او مدد بخواهید که صاحب همه علوم، فقط اوست.

فصل اول

« مقدمات برنامه‌نویسی با C++ »

1 - 1چرا C++ ؟

از زمانی که اولین زبان از خانوادهٔ C به شکل رسمی انتشار یافت، متخصصین بسیاری دربارهٔ توانایی‌ها و قابلیت‌های آن قلم زده و در این وادی قدم زده‌اند. از نظر ایشان آن‌چه بیشتر جلب نظر می‌کرد نکات زیر بود:

- زبان C یک زبان همه منظوره است. دستورالعمل‌های این زبان بسیار شبیه عبارات جبری است و نحو آن شبیه جملات انگلیسی. این امر سبب می‌شود که C یک زبان سطح بالا باشد که برنامه‌نویسی در آن آسان است.

- در این زبان عملگرهایی تعبیه شده که برنامه‌نویسی سطح پایین و به زبان ماشین را نیز امکان‌پذیر می‌سازد. این خاصیت سبب می‌شود تا بتوانیم با استفاده از C برنامه‌های سیستمی و بسیار سریع ایجاد کنیم. به این ترتیب خلاء بین زبان‌های سطح بالا و زبان ماشین پر می‌شود. به همین دلیل به C زبان «سطح متوسط» نیز گفته می‌شود.

- چون C عملگرهای فراوانی دارد، کد منبع برنامه‌ها در این زبان بسیار کوتاه است.

– زبان C برای اجرای بسیاری از دستوراتش از توابع کتابخانه‌ای استفاده می‌کند و بیشتر خصوصیات وابسته به سخت‌افزار را به این توابع واگذار می‌نماید. نتیجه این است که نرم‌افزار تولید شده با این زبان به سخت‌افزار خاص بستگی ندارد و با اندک تغییراتی می‌توانیم نرم‌افزار مورد نظر را روی ماشینی متفاوت اجرا کنیم. یعنی برنامه‌هایی که با C نوشته می‌شوند «قابلیت انتقال» دارند و مستقل از ماشین هستند. علاوه بر این، C اجازه می‌دهد تا کاربر توابع کتابخانه‌ای خاص خودش را ایجاد کند و از آن‌ها در برنامه‌هایش استفاده کند. به این ترتیب کاربر می‌تواند امکانات C را گسترش دهد.

– برنامهٔ مقصدی که توسط کامپایلرهای C ساخته می‌شود بسیار فشرده‌تر و کم‌حجم‌تر از برنامه‌های مشابه در سایر زبان‌ها است.

++C که از نسل C است، تمام ویژگی‌های جذاب بالا را به ارث برده است. این فرزند اما برتری فنی دیگری هم دارد: ++C اکنون «شی‌گرا» است. می‌توان با استفاده از این خاصیت، برنامه‌های شی‌گرا تولید نمود. برنامه‌های شی‌گرا منظم و ساخت‌یافته‌اند، قابل روزآمد کردن‌اند، به سهولت تغییر و بهبود می‌یابند و قابلیت اطمینان و پایداری بیشتری دارند.

و سرانجام آخرین دلیل استفاده از ++C ورود به دنیای **#C** (بخوانید سی شارپ) است. #C که اخیرا عرضه شده، زبانی است کاملا شی‌گرا. این زبان در شی‌گرایی پیشرفت‌های زیادی دارد و همین موضوع سبب پیچیدگی بیشتر آن شده است. ولی برای عبور از این پیچیدگی یک میان‌بر وجود دارد و آن ++C است. نحو #C بسیار شبیه ++C است. اگر ++C را بلد باشیم #C آن‌قدرها هم پیچیده نیست. کلید ورود به دنیای #C میان کدهای ++C نهفته است.

2 – 1 تاریخچهٔ ++C

در دهه 1970 در آزمایشگاه‌های بل زبانی به نام C ایجاد شد. انحصار این زبان در اختیار شرکت بل بود تا این که در سال 1978 توسط Kernighan و Richie شرح کاملی از این زبان منتشر شد و به سرعت نظر برنامه‌نویسان حرفه‌ای را جلب نمود. هنگامی که بحث شی‌گرایی و مزایای آن در جهان نرم‌افزار رونق یافت، زبان C

که قابلیت شی‌گرایی نداشت ناقص به نظر می‌رسید تا این که در اوایل دهۀ 1980 دوباره شرکت بل دست به کار شد و Bjarne Stroustrup زبان C++ را طراحی نمود. C++ ترکیبی از دو زبان C و Simula بود و قابلیت‌های شی‌گرایی نیز داشت. از آن زمان به بعد شرکت‌های زیادی کامپایلرهایی برای C++ طراحی کردند. این امر سبب شد تفاوت‌هایی بین نسخه‌های مختلف این زبان به وجود بیاید و از قابلیت سازگاری و انتقال آن کاسته شود. به همین دلیل در سال 1998 زبان C++ توسط موسسۀ استانداردهای ملی آمریکا (ANSI) به شکل استاندارد و یک‌پارچه درآمد. کامپایلرهای کنونی به این استاندارد پایبندند. کتاب حاضر نیز بر مبنای همین استاندارد نگارش یافته است.

3 – 1 آماده‌سازی مقدمات

یک «برنامه[1]» دستورالعمل‌های متوالی است که می‌تواند توسط یک رایانه اجرا شود. برای نوشتن و اجرای هر برنامه به یک «ویرایش‌گر متن[2]» و یک «کامپایلر[3]» احتیاج داریم. با استفاده از ویرایش‌گر متن می‌توانیم کد برنامه را نوشته و ویرایش کنیم. سپس کامپایلر این کد را به زبان ماشین ترجمه می‌کند. گرچه ویرایش‌گر و کامپایلر را می‌توان به دلخواه انتخاب نمود اما امروزه بیشتر تولیدکنندگان کامپایلر، «محیط مجتمع تولید[4] (IDE)» را توصیه می‌کنند. محیط مجتمع تولید یک بسته نرم‌افزاری است که تمام ابزارهای لازم برای برنامه‌نویسی را یک‌جا دارد: یک ویرایش‌گر متن ویژه که امکانات خاصی دارد، یک کامپایلر، ابزار خطایابی و کنترل اجرا، نمایش‌گر کد ماشین، ابزار تولید خودکار برای ایجاد امکانات استاندارد در برنامه، پیونددهنده‌های[5] خودکار، راهنمای سریع و هوشمند و ... . بیشتر این ابزارها برای سهولت برنامه‌نویسی، توانایی‌های ویژه‌ای دارند. ویرایش‌گرهای متن که در محیط‌های IDE استفاده می‌شوند قابلیت‌های بصری به کد برنامه می‌دهند تا کد خواناتر شود و نوشتن و دنبال کردن برنامه آسان‌تر باشد. به عنوان مثال، دستورات را با رنگ خاصی متمایز می‌سازند، متغیرها را با رنگ دیگری مشخص می‌کنند، توضیحات اضافی را به شکل مایل نشان

---

1 – Program
2 - Text editor
3 – Compiler
4 - Integrated Development Environment
5 - Linker

می‌دهند و حتی با نوشته‌های خاصی شما را راهنمایی می‌کنند که کجای برنامه چه چیزهایی بنویسید. تمام این امکانات سبب شده تا برنامه‌نویسی جذاب‌تر از گذشته باشد و برنامه‌نویس به عوض این که نگران سرگردانی در کد برنامه یا رفع خطا باشد، تمرکز خویش را بر منطق برنامه و قابلیت‌های آن استوار کند.

بستهٔ Visual C++ محصول شرکت میکروسافت و بستهٔ C++ Builder محصول شرکت بورلند نمونه‌های جالبی از محیط مجتمع تولید برای زبان C++ به شمار می‌روند. البته هر دوی این‌ها مخصوص سیستم‌عامل ویندوز هستند. اگر می‌خواهید روی سیستم عامل دیگری مثل Unix یا Linux برنامه بنویسید باید کامپایلری که مخصوص این سیستم‌عامل‌ها است پیدا کنید.

قبل از این که برنامه‌نویسی با C++ را شروع کنیم یک محیط مجتمع تولید روی رایانه‌تان نصب کنید تا بتوانید مثال‌های کتاب را خودتان نوشته و امتحان کنید. این کار هم کمک می‌کند تا C++ را بهتر یاد بگیرید و هم مهارت‌های حرفه‌ای‌تان را در کار با محیط‌های مجتمع تولید افزایش می‌دهد.

4 – 1 شروع کار با C++

حالا شما رایانه‌ای دارید که به یک کامپایلر C++ مجهز است. در ادامهٔ فصل، مثال‌های ساده‌ای از برنامه‌های C++ را ذکر می‌کنیم و نکاتی را در قالب این مثال‌ها بیان خواهیم کرد. اگر از قبل با C یا C++ آشنایی نداشته باشید ممکن است این مثال‌ها مبهم به نظر برسند و برخی از نکات را خوب درک نکنید، اما اصلا جای نگرانی نیست زیرا این فصل یک مرور کلی راجع به C++ است و تمام این نکات در فصل‌های بعدی به شکل کامل شرح داده می‌شوند. این فصل به شما کمک می‌کند تا نکات اولیه و ضروری C++ را یاد بگیرید و همچنین مطلع شوید که در فصول بعدی باید منتظر چه مطالبی باشید.

× مثال 1 – 1 اولین برنامه

برنامهٔ زیر، اولین برنامه‌ای است که می‌نویسیم. اما قبل از این کار نکتهٔ بسیار مهم زیر را همیشه به خاطر داشته باشید:

C++ نسبت به حروف «حساس به حالت[1]» است. یعنی A و a را یکی نمی‌داند. پس در عبارت‌های MY و My و mY و my هیچ یک با دیگری برابر نیست. برای این که در برنامه‌ها دچار این اشتباه نشوید، از قانون زیر پیروی کنید: «همه چیز را با حروف کوچک بنویسید، مگر این‌که برای بزرگ نوشتن برخی از حروف دلیل قانع‌کننده‌ای داشته باشید».

اولین برنامه‌ای که می‌نویسیم به محض تولد، به شما سلام می‌کند و عبارت "Hello, my programmer!" را نمایش می‌دهد:

```
#include <iostream>
int main()
{   std::cout << "Hello, my programmer!\n" ;
    return 0;
}
```

اولین خط از کد بالا یک «راهنمای پیش‌پردازنده[2]» است. راهنمای پیش‌پردازنده شامل اجزای زیر است:

1 – کاراکتر # که نشان می‌دهد این خط، یک راهنمای پیش‌پردازنده است. این کاراکتر باید در ابتدای همهٔ خطوط راهنمای پیش‌پردازنده باشد.

2 – عبارت include

3 – نام یک «فایل کتابخانه‌ای» که میان دو علامت <> محصور شده است. به فایل کتابخانه‌ای «سرفایل[3]» نیز می‌گویند. فایل کتابخانه‌ای که در این‌جا استفاده شده iostream نام دارد.

با توجه به اجزای فوق، راهنمای پیش‌پردازنده خطی است که به کامپایلر اطلاع می‌دهد در برنامه موجودیتی است که تعریف آن را باید در فایل کتابخانه‌ای مذکور جستجو کند. در این برنامه موجودیت std::cout استفاده شده که کامپایلر راجع به آن چیزی نمی‌داند، پس به فایل iostream مراجعه می‌کند، تعریف آن را

---

1 – Case Sensitive
2 – Preprocessor Directive
3 - Header

می‌یابد و سپس آن را اجرا می‌کند.

هر برنامه‌ای که از ورودی و خروجی استفاده می‌کند باید شامل این خط راهنما باشد.

خط دوم برنامه نیز باید در همه برنامه‌های C++ وجود داشته باشد. این خط به کامپایلر می‌گوید که «بدنهٔ اصلی برنامه» از کجا شروع می‌شود. این خط دارای اجزای زیر است:

1 – عبارت int که یک نوع عددی در C++ است. راجع به انواع عددی در C++ در فصل دوم مطالب کاملی خواهید دید.

2 – عبارت main که به آن «تابع اصلی» در C++ می‌گویند.

3 – دو پرانتز () که نشان می‌دهد عبارت main یک «تابع[1]» است. توابع را در فصل پنجم بررسی خواهیم کرد.

هر برنامه فقط باید یک تابع main() داشته باشد. وقتی برنامه اجرا شد، یک عدد صحیح به سیستم‌عامل بازگردانده می‌شود تا سیستم‌عامل بفهمد که برنامه با موفقیت به پایان رسیده یا خیر. عبارت int که قبل از main استفاده شده نشان می‌دهد که این برنامه یک عدد صحیح را به سیستم عامل برمی‌گرداند.

سه خط آخر برنامه، «بدنهٔ اصلی برنامه» را تشکیل می‌دهند. «بدنهٔ اصلی برنامه» مجموعه‌ای از دستورات متوالی است که میان دو علامت براکت **{}** بسته شده است. این براکت‌ها شروع برنامه و پایان برنامه را نشان می‌دهند.

دستورات برنامه از خط سوم شروع شده است. این برنامه فقط دو دستور دارد. اولین دستور یعنی:

```
Std::cout << "Hello, my programmer!\n";
```

رشتهٔ "Hello, my programmer!\n" را به فرایند خروجی std::cout می‌فرستد. این خروجی معمولا صفحه‌نمایش می‌باشد. علامت **<<** «عملگر خروجی[2]» در C++ نامیده می‌شود. این عملگر اجزای سمت راستش را به خروجی سمت چپش

---

1 – Function

2 – output operator

می‌فرستد. حاصل کار این است که رشتهٔ

```
Hello, my programmer!
```

روی صفحه‌نمایش چاپ می‌شود. کاراکتر **\n** نیز در رشته فوق وجود دارد ولی به جای آن چیزی چاپ نمی‌شود، بلکه چاپ این کاراکتر باعث می‌شود مکان‌نما به خط بعدی صفحه‌نمایش پرش کند. به این کاراکتر، کاراکتر «خط جدید» نیز می‌گویند. نمونه‌های دیگری از این نوع کاراکترها را باز هم خواهیم دید.

دستور خط سوم با علامت سمیکولن ; پایان یافته است. این دومین قانون مهم و ساده C++ است: «حتما باید در پایان هر دستور، علامت سمیکولن ; قرار دهید». این علامت به معنای پایان آن دستور است. اگر سمیکولن پایان یک دستور را فراموش کنید، کامپایلر از برنامهٔ شما خطا می‌گیرد و اصلا برنامه را اجرا نمی‌کند.

خط چهارم (دومین دستور برنامه) یعنی

```
return 0;
```

مقدار 0 را به سیستم‌عامل باز می‌گرداند و برنامه را پایان می‌دهد. در انتهای این دستور نیز علامت سمیکولن استفاده شده است. این خط در C++ استاندارد، اختیاری است اما اگر از کامپایلری استفاده می‌کنید که حتما این خط را انتظار دارد (مثل Visual C++ یا C++ Builder) باید دستور return 0; را در انتهای بدنهٔ برنامه قرار دهید. کامپایلرهای مذکور، این دستور را به معنای پایان تابع main() تلقی می‌کنند.

به فاصله‌گذاری‌ها در مثال 1-1 دقت کنید. کامپایلر این فاصله‌های اضافی را نادیده می‌گیرد مگر جایی که لازم باشد شناسه‌ها از هم جدا شوند. یعنی کامپایلر، برنامهٔ فوق را این‌چنین می‌بیند:

```
#include<iostream>
Int main(){std::cout<<"Hello, my programmer!\n";return 0;}
```

ما نیز می‌توانستیم برنامه را به همین شکل بنویسیم اما درک برنامه‌ای که بدون فاصله نوشته می‌شود بسیار مشکل است. استفاده از فاصله‌های مناسب سبب می‌شود خواندن برنامه‌هایتان راحت‌تر باشد.

× مثال 2 – 1 یک برنامهٔ دیگر

برنامهٔ زیر همان خروجی مثال 1-1 را دارد:

```
#include <iostream>
using namespace std;
int main()
{  //prints "Hello, my programmer!" :
   cout << "Hello, my programmer!\n" ;
   return 0;
}
```

دومین خط از برنامهٔ بالا یعنی

```
using namespace std;
```

به کامپایلر می‌گوید که عبارت std:: را در سراسر برنامه در نظر داشته باشد تا مجبور نباشیم برای دستوراتی مثل cout این پیشوند را به کار ببریم. به این طریق می‌توانیم برای دستور خروجی به جای std::cout از عبارت cout تنها استفاده نماییم (که در خط پنجم همین کار را کرده‌ایم). در نهایت یک خط به برنامه اضافه می‌شود اما در عوض مجبور نیستیم قبل از هر cout عبارت std:: را اضافه کنیم. به این ترتیب خواندن و نوشتن برنامه‌های طولانی آسان‌تر می‌شود.

Std یک «فضای نام[1]» است. فضای نام محدوده‌ای است که چند موجودیت در آن تعریف شده است. مثلا موجودیت cout در فضای نام std در سرفایل iostream تعریف شده. با استفاده از فضای نام می‌توانیم چند موجودیت را با یک نام در برنامه داشته باشیم، مشروط بر این که فضای نام هر کدام را ذکر کنیم. برای آشنایی بیشتر با فضای نام به مراجع C++ مراجعه کنید.

همهٔ برنامه‌های این کتاب با دو خط

```
#include <iostream>
using namespace std;
```

شروع می‌شوند. هر چند برای اختصار از این به بعد این دو خط را در برنامه‌ها ذکر

1 – Namespace

نخواهیم کرد اما فراموش نکنید که برای اجرای برنامه‌ها حتما دو خط بالا را به ابتدای برنامه‌تان اضافه کنید.

به خط چهارم برنامه توجه کنید:

```
//  prints "Hello, my programmer!" :
```

این خط ، یک «توضیح[1]» است. توضیح، متنی است که به منظور راهنمایی و درک بهتر به برنامه اضافه می‌شود و تاثیری در اجرای برنامه ندارد. کامپایلر توضیحات برنامه را قبل از اجرا حذف می‌کند. استفاده از توضیح سبب می‌شود که سایر افراد کد برنامهٔ شما را راحت‌تر درک کنند. ما هم در برنامه‌های این کتاب برای راهنمایی شما توضیحاتی اضافه کرده‌ایم. به دو صورت می‌توانیم به برنامه‌های C++ توضیحات اضافه کنیم:

1 – با استفاده از دو علامت اسلش **//** : هر متنی که بعد از دو علامت اسلش بیاید تا پایان همان سطر یک توضیح تلقی می‌شود.

2 – با استفاده از حالت C : هر متنی که با علامت **/*** شروع شود و با علامت ***/** پایان یابد یک توضیح تلقی می‌شود. توضیح حالت C در زبان C به کار می‌رفته که برای حفظ سازگاری در C++ هم می‌توان از آن استفاده کرد.

پس توضیح برنامهٔ بالا را به این شکل هم می‌توانیم بنویسیم:

```
/* prints "Hello, my programmer!" */
```

به فرق بین این دو توضیح توجه کنید: در حالت اول، متنی که بعد از // تا آخر سطر آمده توضیح تلقی می‌شود و با شروع خط بعدی، توضیح نیز خود به خود به پایان می‌رسد ولی در حالت C توضیح با علامت /* شروع می‌شود و همچنان ادامه می‌یابد تا به علامت */ برخورد شود. یعنی توضیح حالت C می‌تواند چند خط ادامه داشته باشد ولی توضیح با // فقط یک خط است و برای ادامهٔ توضیح در خط بعدی باید دوباره در ابتدای خط علامت // را قرار داد.

---

1 – Comment

5 – 1 عملگر خروجی

علامت >> عملگر خروجی در C++ نام دارد (به آن عملگر درج نیز می‌گویند). یک «عملگر[1]» چیزی است که عملیاتی را روی یک یا چند شی انجام می‌دهد. عملگر خروجی، مقادیر موجود در سمت راستش را به خروجی سمت چپش می‌فرستد.

به این ترتیب دستور

```
cout << 66 ;
```

مقدار 66 را به خروجی cout می‌فرستد که cout معمولا به صفحه‌نمایش اشاره دارد. در نتیجه مقدار 66 روی صفحه نمایش درج می‌شود.

cout این قابلیت را دارد که چند چیز را به شکل متوالی و پشت سر هم روی صفحه‌نمایش درج کند. با استفاده از این خاصیت می‌توان چند رشته مجزا را از طریق cout با یکدیگر پیوند داد و خروجی واحد تولید نمود. مثال بعد این موضوع را بیشتر روشن می‌کند. قبل از این که مثال زیر را ببینید باز هم یادآوری می‌کنیم که فراموش نکنید دو خط اصلی راهنمای پیش‌پردازنده و فضای نام را به ابتدای برنامه اضافه کنید.

× مثال 3 – 1 برنامه دیگری از Hello

```
int main()
{  //prints "Hello, my programmer!" :
   cout << "Hello, m" << "y progra" << "mmer!" << endl;
   return 0;
}
```

در این برنامه از عملگر خروجی >> چهار بار استفاده شده و چهار عنصر را به cout فرستاده تا روی صفحه‌نمایش چاپ شوند. سه تای اولی یعنی "Hello, m" و "y progra" و "mmer!" سه رشته‌اند که به یکدیگر پیوند می‌خورند تا عبارت "Hello, my programmer!" در خروجی تشکیل شود. عبارت چهارم یعنی endl همان کار کاراکتر '\n' را انجام می‌دهد. یعنی مکان‌نما را به خط بعدی روی صفحه‌نمایش منتقل می‌کند.

---

1 – Operator

6 – 1 لیترال‌ها و کاراکترها

یک «لیترال[1]» رشته‌ای از حروف، ارقام یا علایم چاپی است که میان دو علامت نقل قول " " محصور شده باشد. در مثال 3 -1 سه عنصر "Hello, m" و "y progra" و "mmer!" لیترال هستند. لیترال می‌تواند تهی باشد : "" و یا می‌تواند فقط یک فاصلهٔ خالی باشد " " و یا فقط یک حرف باشد: "W".

یک «کاراکتر[2]» یک حرف، رقم یا علامت قابل چاپ است که میان دو نشانهٔ ' ' محصور شده باشد. پس 'w' و '!' و '1' هر کدام یک کاراکتر است. هر کاراکتر یک بایت از حافظه را اشغال می‌کند. رایانه‌ها 128 کاراکتر استاندارد را می‌شناسند؛ حروف الفبای انگلیسی کوچک و بزرگ و اعداد صفر تا 9 و کاراکترهای کنترلی و ویرایشی. رایانه‌ها به هر کاراکتر یک عدد یک بایتی تخصیص می‌دهند تا به وسیلهٔ آن عدد، کاراکتر مورد نظر را شناسایی یا دستیابی کنند. این 128 کاراکتر و اعداد تخصیصی هر یک در جدولی به نام جدول ASCII (بخوانید اَسکی) قرار می‌گیرند. این جدول در ضمیمهٔ کتاب آمده است. با دقت در این جدول می‌بینیم که بعضی از کاراکترها دو عضوی هستند، مثل کاراکتر '\n' که قبلا دیدیم. گرچه این کاراکتر از دو عضو n و \ تشکیل شده اما رایانه آن دو با هم را یک کاراکتر فرض می‌کند. بیشتر کاراکترهایی که دو عضوی هستند برای کنترل به کار می‌روند مثل کاراکتر '\n' که مکان‌نما را به خط جدید می‌برد. راجع به کاراکترها در فصل بعدی شرح بیشتری خواهید دید.

به تفاوت سه موجودیت «عدد» و «کاراکتر» و «لیترال رشته‌ای» دقت کنید: 6 یک عدد است، '6' یک کاراکتر است و "6" یک لیترال رشته‌ای است.

کاراکترها را مانند لیترال‌های رشته‌ای می‌توان در خروجی نمایش داد. به مثال بعدی توجه نمایید.

× مثال 4 – 1 نسخهٔ چهارم از برنامهٔ سلام

این برنامه همان خروجی مثال 1-1 را دارد:

---

1 – Literal

2 – Character

```
int main()
{  // prints "Hello, my programmer!":
   cout << "Hello, " << 'm' << "y programmer" << '!' << '\n';
   return 0;
}
```

مثال بالا نشان می‌دهد که کاراکترها را نیز می‌توان به لیترال رشته‌ای پیوند داد و خروجی ترکیبی درست کرد.

× مثال 5 – 1 درج عدد در خروجی

```
int main()
{  // prints "Today is Feb 5 2005":
   cout << "Today is Feb " << 5 << ' ' << 2005 << endl;
   return 0;
}
```

وقتی مثال بالا را اجرا کنیم، خروجی به شکل `Today is Feb 5 2005` روی صفحه‌نمایش چاپ می‌گردد. دقت کنید که یک «کاراکتر جای خالی» ' ' بین 5 و 2005 گنجانده‌ایم تا این دو عدد با فاصله از یکدیگر چاپ شوند و به هم پیوند نخورند.

7 – 1 متغیرها و تعریف آن‌ها:

همهٔ برنامه‌هایی که نوشته می‌شود برای پردازش داده‌ها به کار می‌رود. یعنی اطلاعاتی را از یک ورودی می‌گیرد و آن‌ها را پردازش می‌کند و نتایج مورد نظر را به یک خروجی می‌فرستد. برای پردازش، لازم است که داده‌ها و نتایج ابتدا در حافظهٔ اصلی ذخیره شوند. برای این کار از «متغیرها» استفاده می‌کنیم.

«متغیر[1]» مکانی در حافظه است که چهار مشخصه دارد: نام، نوع، مقدار، آدرس.

وقتی متغیری را تعریف می‌کنیم، ابتدا با توجه به نوع متغیر، آدرسی از حافظه در نظر گرفته می‌شود، سپس به آن آدرس یک نام تعلق می‌گیرد. نوع متغیر بیان می‌کند که

---

1 – Variable

در آن آدرس چه نوع داده‌ای می‌تواند ذخیره شود و چه اعمالی روی آن می‌توان انجام داد. مقدار نیز مشخص می‌کند که در آن محل از حافظه چه مقداری ذخیره شده است.

در C++ قبل از این که بتوانیم از متغیری استفاده کنیم، باید آن را اعلان[1] نماییم. نحو اعلان یک متغیر به شکل زیر است:

```
type name initializer
```

عبارت **type** نوع متغیر را مشخص می‌کند. نوع متغیر به کامپایلر اطلاع می‌دهد که این متغیر چه مقادیری می‌تواند داشته باشد و چه اعمالی می‌توان روی آن انجام داد. مثلا نوع int برای تعریف متغیری از نوع عدد صحیح استفاده می‌شود و نوع char برای تعریف متغیری از نوع کاراکتر به کار می‌رود. انواع اصلی در C++ را در فصل بعدی بررسی می‌کنیم.

عبارت name نام متغیر را نشان می‌دهد. این نام حداکثر می‌تواند 31 کاراکتر باشد، نباید با عدد شروع شود، علایم ریاضی نداشته باشد و همچنین «کلمهٔ کلیدی» نیز نباشد. «کلمهٔ کلیدی» کلمه‌ای است که در C++ برای کارهای خاصی منظور شده است. C++ 63 کلمه کلیدی دارد که در ضمیمه آمده است. بهتر است نام یک متغیر با یک حرف شروع شود.

عبارت *initializer* عبارت «مقداردهی اولیه» نام دارد. با استفاده از این عبارت می‌توان مقدار اولیه‌ای در متغیر مورد نظر قرار داد.

دستور زیر تعریف یک متغیر صحیح را نشان می‌دهد:

```
int n = 50;
```

این دستور متغیری به نام n تعریف می‌کند و مقدار اولیه 50 را درون آن قرار می‌دهد. این متغیر از نوع int است، یعنی فقط می‌تواند اعداد صحیح را نگهداری کند.

برای این که مقداری را در یک متغیر قرار دهیم، از عملگر انتساب « **=** » استفاده می‌کنیم. مثلا دستور n=50; مقدار 50 را در متغیر n قرار می‌دهد.

---

1 – Declaration

به طور غیر مستقیم نیز می‌توانیم مقداری را به یک متغیر تخصیص دهیم. برای مثال اگر متغیر m مقدار 45 داشته باشد، آنگاه دستور n=m; سبب می‌شود که مقدار n برابر با مقدار m شود؛ یعنی مقدار n برابر با 45 شود.

همچنین می‌توانیم یک مقدار را به طور هم‌زمان در چند متغیر از یک نوع قرار دهیم. دستور n=m=k=45; مقدار 45 را ابتدا در k و سپس در m و سرانجام در n قرار می‌دهد. به این ترتیب هر سه متغیر فوق مقدار 45 خواهند داشت.

× مثال 6 – 1 استفاده از متغیرهای نوع صحیح

```
int main()
{  // prints "m = 45 and n = 55":
   int m = 45;
   int n = 55;
   cout << "m = " << m << " and n = " << n << endl;
   return 0;
}
```

خروجی این برنامه به شکل زیر است:

```
m = 44 and n = 77
```

در برنامهٔ بالا متغیر m از نوع صحیح int و مقدار اولیهٔ 45 تعریف شده. سپس متغیر n از نوع صحیح int و مقدار اولیهٔ 55 تعریف گشته است. سرانجام مقادیر این دو متغیر با دستور cout روی خروجی چاپ شده است.

می‌توانیم متغیرها را هنگام تعریف، بدون مقدار رها کنیم و مقداردهی را به درون برنامه موکول نماییم. به مثال زیر نگاه کنید:

× مثال 7 – 1 تعریف متغیر بدون مقداردهی

این برنامه همان خروجی مثال 6 – 1 را دارد:

```
int main()
{  // prints "m = 45 and n = 55":
   int m;
   int n;
   m = 45;          // assigns the value 45 to m
```

```
   n = m + 10;      // assigns the value 55 to n
   cout << "m = " << m << " and n = " << n << endl;
   return 0;
}
```

خروجی این برنامه به شکل زیر است:

```
m = 45 and n = 55
```

در خط سوم و چهارم، متغیرهای m و n تعریف شده‌اند اما مقداردهی نشده‌اند. در خط پنجم مقدار 45 در متغیر m قرار داده می‌شود. در خط ششم نیز مقدار m+10 یعنی 45+10 که برابر با 50 است در n قرار داده می‌شود. پس از این که دو متغیر مقداردهی شدند، می‌توانیم با دستور cout آن‌ها را چاپ کنیم.

در مثال بالا می‌توانستیم متغیرهای m و n را روی یک خط تعریف کنیم به شکل زیر:

```
int m,n;
```

به این ترتیب هر دو متغیر از نوع int تعریف می‌شوند و هیچ کدام مقداردهی نمی‌شوند. توجه کنید که m و n با یک علامت کاما , از یکدیگر جدا شده‌اند. لذا می‌توانیم چند متغیر را روی یک سطر تعریف کنیم به شرطی که همه از یک نوع باشند.

8 – 1 مقداردهی اولیه[1] به متغیرها

در بسیاری از موارد بهتر است متغیرها را در همان محلی که اعلان می‌شوند مقداردهی کنیم. استفاده از متغیرهای مقداردهی نشده ممکن است باعث ایجاد دردسرهایی شود. مثال زیر این موضوع را نشان می‌دهد.

× مثال 8 – 1 متغیر مقداردهی نشده

```
int main()
{  // prints "x = ?? and y = 45":
   int x;            // BAD: x is not initialized
   int y=45;
   cout << "x = " << x << " and y = " << y << endl;
```

1 – Initializing

```
    return 0;
}
```

```
x = ?? and y = 45
```

در مثال بالا متغیر x تعریف شده اما در سراسر برنامه هیچ مقداری در آن گذاشته نشده است. اگر سعی کنیم چنین متغیری را چاپ کنیم با نتایج غیرمنتظره‌ای مواجه خواهیم شد. کامپایلری که برنامهٔ بالا را اجرا کرده، مقدار x را ?? چاپ نموده است به این معنی که مقدار درون x شناخته شده نیست. یک کامپایلر دیگر ممکن است خروجی زیر را بدهد:

```
x = 7091260 and y = 45
```

با وجود این که به x هیچ مقداری تخصیص نداده‌ایم، در خروجی مقدار 7091260 چاپ شده است. به این مقدار «زباله[1]» می‌گویند (یعنی مقداری که قبلا در آن قسمت از حافظه بوده و سپس بدون استفاده رها شده است).

دردسر متغیرهای مقداردهی نشده وقتی بزرگ‌تر می‌شود که سعی کنیم متغیر مقداردهی نشده را در یک محاسبه به کار ببریم. مثلا اگر x را که مقداردهی نشده در عبارت `y = x + 5;` به کار ببریم، حاصل y غیر قابل پیش‌بینی خواهد بود. برای اجتناب از چنین مشکلاتی عاقلانه است که متغیرها را همیشه هنگام تعریف، مقداردهی کنیم.

9 – 1 ثابت‌ها

در بعضی از برنامه‌ها از متغیری استفاده می‌کنیم که فقط یک بار لازم است آن را مقداردهی کنیم و سپس مقدار آن متغیر در سراسر برنامه بدون تغییر باقی می‌ماند. مثلا در یک برنامهٔ محاسبات ریاضی، متغیری به نام PI تعریف می‌کنیم و آن را با 3.14 مقداردهی می‌کنیم و می‌خواهیم که مقدار این متغیر در سراسر برنامه ثابت بماند. در چنین حالاتی از «ثابت‌ها[2]» استفاده می‌کنیم. یک ثابت، یک نوع متغیر است که فقط یک بار مقداردهی می‌شود و سپس تغییر دادن مقدار آن در ادامهٔ برنامه ممکن نیست.

---

1 – Garbage

2 - Constant

تعریف ثابت‌ها مانند تعریف متغیرهاست با این تفاوت که کلمه کلیدی **const** به ابتدای تعریف اضافه می‌شود. پس دستور `int k=3;` متغیری به نام k و با مقدار اولیهٔ 3 تعریف می‌کند که در ادامهٔ برنامه می‌توان مقدار آن را تغییر داد ولی دستور

```
const int k=3;
```

ثابتی به نام k و با مقدار اولیهٔ 3 تعریف می‌کند که این مقدار را نمی‌توان در ادامهٔ برنامه تغییر داد.

ثابت‌ها را باید هنگام تعریف، مقداردهی اولیه نمود. یک ثابت می‌تواند از نوع کاراکتری، صحیح ، اعشاری و ... باشد. مثال زیر چند نوع ثابت را نشان می‌دهد.

× مثال 9 – 1 تعریف ثابت‌ها

برنامه زیر خروجی ندارد:

```
int main()
{  // defines constants; has no output:
   const char BEEP ='\b';
   const int MAXINT=2147483647;
   const float DEGREE=23.53;
   const double PI=3.14159265358979323846
   return 0;
}
```

در برنامهٔ بالا نام ثابت‌ها را با حروف انگلیسی بزرگ نوشته‌ایم. معمولا در برنامه‌ها برای نشان دادن ثابت‌ها از حروف بزرگ استفاده می‌کنند. هر چند این کار اجباری نیست، اما با رعایت این قرارداد به راحتی می‌توانیم ثابت‌ها را از متغیرها تمیز دهیم.

10 – 1 عملگر ورودی

در بیشتر برنامه‌ها از کاربر خواسته می‌شود تا مقادیری را وارد کند. برای این که بتوانیم هنگام اجرای برنامه مقادیری را وارد کنیم از عملگر ورودی **>>** استفاده می‌کنیم.

عملگر ورودی مانند عملگر خروجی است و به همان سادگی کار می‌کند. استفاده از دستور ورودی به شکل زیر است:

cin >> *variable*;

variable نام یک متغیر است. مثلا دستور cin >> m; مقداری را از ورودی (صفحه کلید) گرفته و درون متغیر m قرار می‌دهد. این که ورودی عدد باشد یا کاراکتر یا ترکیبی از این دو، به نوع m بستگی دارد.

× مثال 10 – 1 استفاده از عملگر ورودی

برنامهٔ زیر یک عدد از کاربر گرفته و همان عدد را دوباره در خروجی نمایش می‌دهد:

```
int main()
{  // reads an integer from input:
   int m;
   cout << "Enter a number: ";
   cin >> m;
   cout << "your number is: " << m << endl;
   return 0;
}
```

هنگامی که برنامهٔ بالا اجرا شود، عبارت Enter a number: روی صفحه چاپ می‌شود و منتظر می‌ماند تا یک عدد را وارد کنید. برای وارد کردن عدد باید آن را تایپ کرده و سپس کلید Enter را فشار دهید. خط cin >> m; عدد را از ورودی گرفته و در متغیر m قرار می‌دهد. توسط خط بعدی نیز جملهٔ your number is: و سپس مقدار m چاپ می‌شود. شکل زیر یک نمونه از اجرای برنامهٔ بالا را نشان می‌دهد:

```
Enter a number: 52
your number is: 52
```

در این شکل، عددی که کاربر وارد کرده با حروف سیاه‌تر نشان داده شده است. باز هم به کد برنامه نگاه کنید. در خط چهارم کد یعنی:

```
cout << "Enter a number: ";
```

از کاراکتر '\n' یا endl استفاده نکرده‌ایم تا مکان‌نما در همان خط باقی بماند و ورودی در جلوی همان خط وارد شود.

عملگر ورودی نیز مانند عملگر خروجی به شکل جریانی رفتار می‌کند. یعنی همان طور که در عملگر خروجی می‌توانستیم چند عبارت را با استفاده از چند عملگر >> به صورت پشت سر هم چاپ کنیم، در عملگر ورودی نیز می‌توانیم با استفاده از چند عملگر << چند مقدار را به صورت پشت سر هم دریافت کنیم. مثلا با استفاده از دستور:

```
cin >> x >> y >> z;
```

سه مقدار x و y و z به ترتیب از ورودی دریافت می‌شوند. برای این کار باید بین هر ورودی یک فضای خالی (space) بگذارید و پس از تایپ کردن همهٔ ورودی‌ها، کلید enter را بفشارید. آخرین مثال فصل، این موضوع را بهتر نشان می‌دهد.

× مثال 11 – 1 چند ورودی روی یک خط

برنامهٔ زیر مانند مثال 10 – 1 است با این تفاوت که سه عدد را از ورودی گرفته و همان اعداد را دوباره در خروجی نمایش می‌دهد:

```
int main()
{  // reads 3 integers from input:
   int q, r, s;
   cout << "Enter three numbers: ";
   cin >> q >> r >> s;
   cout << "your numbers are: << q << ", " << r
        << ", " << s << endl;
   return 0;
}
```

نمونه‌ای از اجرای برنامهٔ بالا در زیر آمده است:

```
Enter three numbers: 35 70 9
your numbers are: 35, 70, 9
```

اعدادی که کاربر در این اجرا وارد نموده به صورت سیاه‌تر نشان داده شده است.

پرسش‌های گزینه‌ای

1 – کدام یک از موارد زیر در مورد C++ صحیح نیست؟

الف) از نسل زبان C است　　ب) شی‌گراست

ج) همه منظوره است　　د) سطح پایین است

2 – کدام یک از زبان‌های زیر در تولید C++ نقش داشته است؟

الف) Basic　　ب) Java

ج) Simula　　د) Pascal

3 – محیط مجتمع تولید (IDE) شامل کدام یک از امکانات زیر است؟

الف) ویرایش‌گر متن　　ب) کامپایلر

ج) ابزارخطایابی　　د) همهٔ موارد

4 – کدام یک از عملگرهای زیر عملگر خروجی در C++ است؟

الف) >>　　ب) #　　ج) <<　　د) //

5 – از موارد زیر کدام در مورد عبارت "10" صحیح است؟

الف) "10" یک کاراکتر است　　ب) "10" یک عدد صحیح است

ج) "10" یک لیترال است　　د) "10" یک عدد دودویی است

6 – در مورد عبارت `int k = 8;` کدام عبارت صحیح نیست؟

الف) متغیر k با مقدار 8 مقداردهی اولیه شده است

ب) متغیر k از نوع int است

ج) متغیر k در آدرس 8 از حافظه قرار گرفته است

د) در آدرس k مقدار 8 قرار گرفته است

7 – عملگر انتساب در C++ چیست؟

الف) =　　ب) ==　　ج) #　　د) =!

8 – در کد مقابل چه روی می‌دهد؟

```
const int x=7;    x+=9;
```

الف) مقدار 9 در x ذخیره می‌شود

ب) مقدار 9+7 یعنی 16 در x ذخیره می‌شود

ج) مقدار 9 و 7 هر کدام جدا در x ذخیره می‌شود

د) دستور دوم اجرا نمی‌شود و کامپایلر خطا می‌گیرد.

9 – کد `cin >> age;` چه کاری انجام می‌دهد؟

الف) مقدار متغیر `age` را چاپ می‌کند

ب) مقداری از ورودی گرفته و در `age` می‌گذارد

ج) بررسی می‌کند که آیا `cin` بزرگ‌تر از `age` است

د) سینوس مقدار `age` را محاسبه می‌کند

10 – در مورد دستور `#include<iostream>` کدام گزینه صحیح است؟

الف) یک دستور خروجی است که عبارت `"iostream"` را در خروجی چاپ می‌کند

ب) یک راهنمای پیش‌پردازنده است که سرفایل `iostream` را معرفی می‌کند

ج) یک دستور ورودی است که مقدار دریافتی را در متغیر `iostream` قرار می‌دهد

د) این دستور معتبر نیست زیرا علامت سمیکولن ندارد

پرسش‌های تشریحی

1- توضیح دهید چرا به C++ یک زبان سطح متوسط می‌گویند؟

2- یک توضیح حالت C با یک توضیح حالت C++ چه تفاوت‌هایی دارد؟

3- چه اشتباهی در این برنامه هست؟

```
#include <iostream>
int main()
{  //prints "Hello, World!" :
   cout << "Hello, World!\n"
   return 0;
}
```

4- در توضیح حالت C زیر چه اشتباهی وجود دارد؟

```
cout << "Hello, /* change? */ world.\n";
```

5- چه اشتباهی در این برنامه است؟

```
#include <iostream>
int main
{  //prints "n = 22":
   n = 22;
   cout << "n = << n << endl;
   return 0;
}
```

6- در دستور مقداردهی زیر چه اشکالی وجود دارد؟

```
int k=6, age=20, grade=1, A+ =20;
```

7- در کد زیر چه اشتباهی هست؟

```
int Grade;
grade = 18;
```

8- قطعه برنامهٔ زیر را اصلاح کنید:

```
int main()
   cout >> "Enter a number:";
   cin >> n;
   cout >> "Your number is" >> n >> endl
```

9- برنامهٔ زیر سن کاربر را از ورودی دریافت کرده و سپس آن مقدار را در خروجی چاپ می‌کند اما خطوط این برنامه به هم ریخته است. آن را به ترتیب درست مرتب کنید:

```
int main()
{  // testing:
   cout << "Your age is: " << age << " years." << endl;
   cin >> age;
   cout << "Enter your age: ";
   int age;
   return 0;
}
```

تمرین‌های برنامه‌نویسی

1- برنامه‌ای بنویسید که همهٔ حروف الفبای انگلیسی را به ترتیب چاپ کند به طوری که کنار هر حرف بزرگ، مشابه کوچک آن هم وجود داشته باشد.

2- برنامه‌ای بنویسید که به وسیلهٔ ستاره‌ها، حرف B را در یک بلوک 7×6 مانند زیر چاپ کند.

```
xxxxx
x  x
x  x
xxxxx
x  x
x  x
xxxxx
```

3- برنامه‌ای نوشته و اجرا کنید که اولین حرف نام فامیل شما را به وسیلهٔ ستاره‌ها در یک بلوک 7×7 چاپ کند.

4- برنامه‌ای نوشته و اجرا کنید که نشان دهد چه اتفاقی می‌افتد اگر هر یک از ده سویچ خروجی زیر چاپ شود:

```
\a , \b , \n , \r , \t , \v , \' , \" , \\ , \?
```

5- برنامه‌ای را نوشته و اجرا کنید که مجموع، تفاضل، حاصل‌ضرب، خارج‌قسمت و باقیماندهٔ دو عدد 60 و 7 را چاپ کند.

6- برنامه‌ای را نوشته و اجرا کنید که دو عدد صحیح از ورودی گرفته و مجموع، تفاضل، حاصل‌ضرب، خارج‌قسمت و باقیماندهٔ آن دو عدد را چاپ کند.

فصل دوم

«انواع اصلی»

1 – 2 انواع دادهٔ عددی

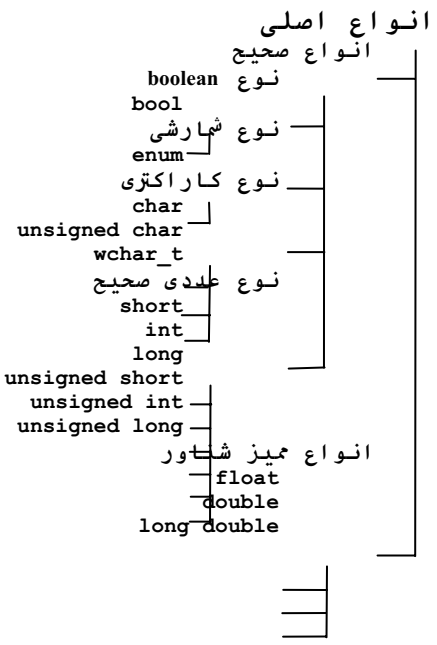
ما در زندگی روزمره از داده‌های مختلفی استفاده می‌کنیم: اعداد ، تصاویر، نوشته‌ها یا حروف الفبا، صداها، بوها و ... . با پردازش این داده‌ها می‌توانیم تصمیماتی اتخاذ کنیم، عکس‌العمل‌هایی نشان دهیم و مساله‌ای را حل کنیم. رایانه‌ها نیز قرار است همین کار را انجام دهند. یعنی داده‌هایی را بگیرند، آن‌ها را به شکلی که ما تعیین می‌کنیم پردازش کنند و در نتیجه اطلاعات مورد نیازمان را استخراج کنند. اما رایانه‌ها یک محدودیت مهم دارند: فقط اعداد را می‌توانند پردازش کنند. پس هر داده‌ای برای این که قابل پردازش باشد باید تبدیل به عدد شود. ممکن است عجیب به نظر برسد که مثلا صدا یا تصویر را چطور می‌توان به اعداد تبدیل کرد اما این کار واقعا در رایانه‌ها انجام می‌گیرد و هر نوع داده‌ای به ترکیبی از صفرها و یک‌ها که «اعداد دودویی[1]» خوانده می‌شوند، تبدیل می‌گردد. سر و کار داشتن با اعدادی که فقط از صفرها و یک‌های طولانی تشکیل شده‌اند بسیار گیج‌کننده و وقت‌گیر است. علاوه بر این مایلیم که با داده‌های واقعی در برنامه‌ها کار کنیم. بنابراین در زبان‌های برنامه‌نویسی، این تبدیل

---

1 – Binary

داده‌ها به کامپایلر واگذار شده است و برنامه‌نویس با خیال راحت می‌تواند انواع واقعی را که آن زبان در اختیار می‌گذارد به کار برد. وقتی برنامه کامپایل شد، این داده‌ها خود به خود به اعداد دودویی تبدیل می‌شوند.

در C++ دو نوع اصلی داده وجود دارد: «نوع صحیح[1]» و «نوع ممیز شناور[2]». همهٔ انواع دیگر از روی این دو ساخته می‌شوند (به شکل زیر دقت کنید).



نوع صحیح برای نگهداری اعداد صحیح (اعداد 0 و 1 و 2 و ...) استفاده می‌شود. این اعداد بیشتر برای شمارش به کار می‌روند و دامنه محدودی دارند.

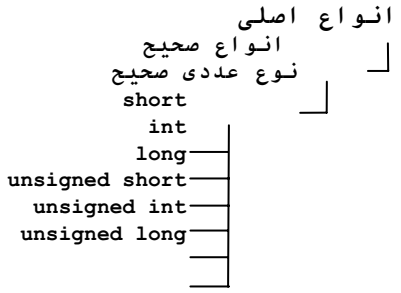
نوع ممیز شناور برای نگهداری اعداد اعشاری استفاده می‌شود. اعداد اعشاری بیشتر برای اندازه‌گیری دقیق به کار می‌روند و دامنهٔ بزرگ‌تری دارند. یک عدد اعشاری مثل 187/352 را می‌توان به شکل $18/7352\times10$ یا $1/87352\times10^{2}$ یا $1873/52\times10^{-1}$ یا $18735/2\times10^{-2}$ و یا ... نوشت. به این ترتیب با کم و زیاد کردن توان عدد 10 ممیز عدد نیز جابه‌جا می‌شود. به همین دلیل است که به اعداد اعشاری «اعداد ممیز شناور» می‌گویند.

2 – 2 متغیر عدد صحیح

C++ شش نوع متغیر عدد صحیح دارد که در شکل آمده است. تفاوت این شش نوع مربوط به میزان حافظهٔ مورد استفاده و محدودهٔ مقادیری است که هر کدام

1 – Integer
2 – Floating point

می‌توانند داشته باشند. این میزان حافظهٔ مورد استفاده و محدودهٔ مقادیر، بستگی زیادی به سخت‌افزار و همچنین سیستم عامل دارد. یعنی ممکن است روی یک رایانه، نوع int دو بایت از حافظه را اشغال کند در حالی که روی رایانه‌ای از نوع دیگر نوع int به چهار بایت حافظه نیاز داشته باشد. با استفاده از سرفایل <limits> می‌توان مشخص نمود که هر نوع عدد صحیح روی رایانه‌تان چه محدوده‌ای دارد.



× مثال 1 – 2 محدوده‌های نوع عدد صحیح

این برنامه محدوده‌های شش نوع عدد صحیح در C++ را چاپ می‌کند:

```
#include <iostream>
#include <limits>     //defines the constants SHRT_MIN, etc.
using namespace std;
int main()
{  //prints some of the constants stored in the <limits>
header:
   cout << "minimum short = " << SHRT_MIN << endl;
   cout << "maximum short = " << SHRT_MAX << endl;
   cout << "minimum unsigned short = 0" << endl;
   cout << "maximum unsigned short = " << USHRT_MAX << endl;
   cout << "minimum int = " << INT_MIN << endl;
   cout << "maximum int = " << INT_MAX << endl;
   cout << "minimum unsigned int = 0" << endl;
   cout << "maximum unsigned int = " << UINT_MAX << endl;
   cout << "minimum long = " << LONG_MIN << endl;
   cout << "maximum long = " << LONG_MAX << endl;
   cout << "minimum unsigned long = 0" << endl;
   cout << "maximum unsigned long = " << ULONG_MAX << endl;
   return 0;
}
```

```
minimum short = -32768
maximum short = 32767
```

```
minimum unsigned short = 0
maximum unsigned short = 65535
minimum int = -2147483648
maximum int = 2147483647
minimum unsigned int = 0
maximum unsigned int = 4294967295
minimum long = -2147483648
maximum long = 2147483647
minimum unsigned long = 0
maximum unsigned long = 4294967295
```

سرفایل <limits> حاوی تعریف شناسه‌های SHRT_MIN ، SHRT_MAX ، USHRT_MAX و سایر شناسه‌هایی است که در برنامهٔ بالا استفاده شده است. این شناسه‌ها گستره‌ای که نوع عدد صحیح مربوطه می‌تواند داشته باشد را نشان می‌دهند. مثلا شناسه SHRT_MIN نشان می‌دهد که متغیری از نوع short حداقل چه مقداری می‌تواند داشته باشد و شناسه SHRT_MAX بیان می‌کند که متغیری از نوع short حداکثر چه مقداری می‌تواند داشته باشد. مثال بالا روی یک رایانه با پردازنده Pentium II با سیستم عامل ویندوز 98 اجرا شده است. خروجی این مثال نشان می‌دهد که شش نوع عدد صحیح در این رایانه محدوده‌های زیر را دارند:

short:	-32,786 تا 32,767	($2^8 \Rightarrow$ 1 byte)
int:	-2,147,483,648 تا 2,147,483,647	($2^{32} \Rightarrow$ 4 bytes)
long:	-2,147,483,648 تا 2,147,483,647	($2^{32} \Rightarrow$ 4 bytes)
unsigned short:	0 تا 65,535	($2^8 \Rightarrow$ 1 byte)

با دقت در این جدول مشخص می‌شود که در رایانهٔ مذکور، نوع long مانند نوع int است و نوع unsigned long نیز مانند unsigned int است. گرچه ممکن است این انواع روی رایانه‌ای از نوع دیگر متفاوت باشد.

وقتی برنامه‌ای می‌نویسید، توجه داشته باشید که از نوع صحیح مناسب استفاده کنید تا هم برنامه دچار خطا نشود و هم حافظهٔ سیستم را هدر ندهید.

3 – 2 محاسبات اعداد صحیح

اکنون که با انواع متغیرهای عدد صحیح آشنا شدیم، می‌خواهیم از این متغیرها در محاسبات ریاضی استفاده کنیم. C++ مانند اغلب زبان‌های برنامه‌نویسی برای محاسبات

از عملگرهای جمع (+) ، تفریق (-) ، ضرب (*) ، تقسیم (/) و باقیمانده (%) استفاده می‌کند.

× مثال 2 – 2 محاسبات اعداد صحیح

برنامهٔ زیر نحوهٔ استفاده و عملکرد عملگرهای حسابی را نشان می‌دهد:

```
int main()
{  //tests operators +, -, *, /, and %:
   int m=54;
   int n=20;
   cout << "m = " << m << " and n = " << n << endl;
   cout << "m+n = " << m+n << endl;   // 54+20 = 74
   cout << "m-n = " << m-n << endl;   // 54-20 = 34
   cout << "m*n = " << m*n << endl;   // 54*20 = 1080
   cout << "m/n = " << m/n << endl;   // 54/20 = 2
   cout << "m%n = " << m%n << endl;   // 54%20 = 14
   return 0;
}
```

```
m = 54 and n = 20
m+n = 74
m-n = 34
m*n = 1080
m/n = 2
m%n = 14
```

نتیجهٔ تقسیم m/n جالب توجه است. حاصل این تقسیم برابر با 2 است نه 2.7 توجه به این مطلب بسیار مهم است. این امر نشان می‌دهد که حاصل تقسیم یک عدد صحیح بر عدد صحیح دیگر، همواره یک عدد صحیح است نه عدد اعشاری. همچنین به حاصل m%n نیز دقت کنید. عملگر % باقیماندهٔ تقسیم را به دست می‌دهد. یعنی حاصل عبارت 54%20 برابر با 14 است که این مقدار، باقیماندهٔ تقسیم 54 بر 20 است.

4 – 2 عملگرهای افزایشی و کاهشی

C++ برای دستکاری مقدار متغیرهای صحیح، دو عملگر جالب دیگر دارد: عملگر ++ مقدار یک متغیر را یک واحد افزایش می‌دهد و عملگر -- مقدار یک متغیر

را یک واحد کاهش می‌دهد. اما هر کدام از این عملگرها دو شکل متفاوت دارند: شکل «پیشوندی» و شکل «پسوندی».

در شکل پیشوندی، عملگر قبل از نام متغیر می‌آید مثل ++m یا --n . در شکل پسوندی، عملگر بعد از نام متغیر می‌آید مثل m++ یا n-- . تفاوت شکل پیشوندی با شکل پسوندی در این است که در شکل پیشوندی ابتدا متغیر، متناسب با عملگر، افزایش یا کاهش می‌یابد و پس از آن مقدار متغیر برای محاسبات دیگر استفاده می‌شود ولی در شکل پسوندی ابتدا مقدار متغیر در محاسبات به کار می‌رود و پس از آن مقدار متغیر یک واحد افزایش یا کاهش می‌یابد. برای درک بهتر این موضوع به مثال بعدی توجه کنید.

× مثال 3 – 2 استفاده از عملگرهای پیش‌افزایشی و پس‌افزایشی

```
int main()
{  //shows the difference between m++ and ++m:
   int m, n;
   m = 75;
   n = ++m;     // the pre-increment operator is applied to m
   cout << "m = " << m << ", n = " << n << endl;
   m = 75;
   n = m++;     // the post-increment operator is apllied to m
   cout << "m = " << m << ", n = " << n << endl;
   return 0;
}
```

```
m = 45, n = 45
m = 45, n = 44
```

در خط پنجم برنامه یعنی در عبارت

```
n = ++m;
```

از عملگر پیش‌افزایشی استفاده شده است. پس ابتدا مقدار m به 76 افزایش می‌یابد و سپس این مقدار به n داده می‌شود. بنابراین وقتی در خط ششم مقدار این دو متغیر چاپ می‌شود، m = 76 و n = 76 خواهد بود.

در خط هشتم برنامه یعنی در عبارت `n = m++;` از عملگر پس‌افزایشی استفاده شده است. بنا بر این ابتدا مقدار m که 75 است به n تخصیص می‌یابد و پس از آن مقدار m به 76 افزایش داده می‌شود. پس وقتی در خط نهم برنامه مقدار این دو متغیر چاپ می‌شود، `m = 76` است ولی `n = 75` خواهد بود.

عملگرهای افزایشی و کاهشی در برنامه‌های C++ فراوان به کار می‌روند. گاهی به شکل پیشوندی و گاهی به شکل پسوندی؛ این بستگی به منطق برنامه دارد که کجا از کدام نوع استفاده شود.

5 – 2 عملگرهای مقدارگذاری مرکب

قبلا از عملگر = برای مقدارگذاری در متغیرها استفاده کردیم. مثلا دستور `m=75;` مقدار 75 را درون m قرار می‌دهد و همچنین دستور `m = m+8;` مقدار m را هشت واحد افزایش می‌دهد. C++ عملگرهای دیگری دارد که مقدارگذاری در متغیرها را تسهیل می‌نمایند. مثلا با استفاده از عملگر += می‌توانیم هشت واحد به m اضافه کنیم اما با دستور کوتاه‌تر: **`m += 8;`**

دستور بالا معادل دستور `m = m + 8;` است با این تفاوت که کوتاه‌تر است. به عملگر **+=** «عملگر مرکب» می‌گویند زیرا ترکیبی از عملگرهای + و = می‌باشد. پنج عملگر مرکب در C++ عبارتند از: **+=** و **-=** و ***=** و **/=** و **%=**

نحوهٔ عمل این عملگرها به شکل زیر است:

```
m += 8;      ⟶    m = m + 8;
m -= 8;      ⟶    m = m - 8;
m *= 8;      ⟶    m = m * 8;
m /= 8;      ⟶    m = m / 8;
m %= 8;      ⟶    m = m % 8;
```

مثال زیر، کار این عملگرها را نشان می‌دهد.

× مثال 4 – 2 کاربرد عملگرهای مرکب

```
int main()
{  //tests arithmetic assignment operators:
```

```
    int n=22;
    cout << " n = " << n << endl;
    n += 9;  // adds 9 to n
    cout << "After n += 9, n = " << n << endl;
    n -= 5;  //substracts 5 from n
    cout << "After n -= 5, n = " << n << endl;
    n *= 2;  //multiplies n by 2
    cout << "After n *= 2, n = " << n << endl;
    n /= 3;  //divides n by 3
    cout << "After n /= 3, n = " << n << endl;
    n %= 7;  //reduces n to the remainder from dividing by 4
    cout << "After n %= 7, n = " << n << endl;
    return 0;
}
```

```
n = 22
After n += 9, n = 31
After n -= 5, n = 26
After n *= 2, n = 52
After n /= 3, n = 17
After n %= 7, n = 3
```

6 – 2 انواع ممیز شناور

عدد ممیز شناور به بیان ساده همان عدد اعشاری است. عددی مثل 123.45 یک عدد اعشاری است. برای این که مقدار این عدد در رایانه ذخیره شود، ابتدا باید به شکل دودویی تبدیل شود:

$$123.45 = 1111011.0111001_2$$

اکنون برای مشخص نمودن محل اعشار در عدد، تمام رقم‌ها را به سمت راست ممیز منتقل می‌کنیم. البته با هر جابجایی ممیز، عدد حاصل باید در توانی از 2 ضرب شود:

$$123.45 = 0.11110110111001 \times 2^7$$

به مقدار 11110110111001 «مانتیس عدد» و به 7 که توان روی دو است، «نمای عدد» گفته می‌شود. از آن‌جا که ممیز می‌تواند به شکل شناور جابجا شود، به اعداد اعشاری اعداد ممیز شناور می‌گویند. حال برای ذخیره‌سازی عدد مفروض کافی است

که مانتیس و نما را ذخیره کنیم. هنگامی که بخواهیم این مقدار ذخیره شده را بازیابی کنیم، سیستم‌عامل نما و مانتیس را در مسیری عکس مسیر بالا به کار می‌گیرد تا عدد 123.45 را از روی آن دوباره بسازد. در مورد عددی مثل عدد مذکور ممکن است این روش ذخیره‌سازی، طولانی و بی‌مورد به نظر برسد. اما اعداد ممیز شناور شامل اعداد خیلی کوچک مثل 0.000000001 یا اعداد خیلی بزرگ مثل 100000000.00000002 هستند که ذخیره‌سازی و انجام محاسبات ریاضی روی آن‌ها با استفاده از مانتیس و نما بسیار آسان‌تر است.

در C++ سه نوع ممیز شناور وجود دارد: نوع float و نوع double و نوع long double.

معمولا نوع float از چهار بایت برای نگهداری عدد استفاده می‌کند، نوع double از هشت بایت و نوع long double از هشت یا ده یا دوازده یا شانزده بایت. در یک float 32 بیتی (چهار بایتی) از 23 بیت برای ذخیره‌سازی مانتیس استفاده می‌شود و 8 بیت نیز برای ذخیره‌سازی نما به کار می‌رود و یک بیت نیز علامت عدد را نگهداری می‌کند.

در یک double 64 بیتی (هشت بایتی) از 52 بیت برای ذخیره‌سازی مانتیس استفاده می‌شود و 11 بیت برای نگهداری نما به کار می‌رود و یک بیت نیز علامت عدد را نشان می‌دهد.

7 – 2 تعریف متغیر ممیز شناور

تعریف متغیر ممیز شناور مانند تعریف متغیر صحیح است. با این تفاوت که از کلمهٔ کلیدی float یا double برای مشخص نمودن نوع متغیر استفاده می‌کنیم. مثلا دستور `float x;` متغیر x را از نوع ممیز شناور تعریف می‌کند. دستور `float x=12.3;` متغیر x را از نوع ممیز شناور تعریف کرده و مقدار اولیهٔ 12.3 را درون آن قرار می‌دهد. دستور `double x,y=0;` دو متغیر x و y را از نوع double تعریف می‌کند که مقدار x هنوز مشخص نیست ولی y مقدار 0.0 دارد.

× مثال 5 – 2 حساب ممیز شناور

اعداد ممیز شناور را نیز مثل اعداد صحیح می‌توانیم در محاسبات به کار ببریم. مثال زیر این موضوع را نشان می‌دهد. این مثال مانند مثال 2 – 2 است با این تفاوت که متغیرها از نوع ممیز شناور float هستند:

```
int main()
{  //tests operators +, -, *, /, and %:
   float x=54.0;
   float y=20.0;
   cout << "x = " << x << " and y = " << y << endl;
   cout << "x+y = " << x+y << endl;   // 54.0+20.0 = 74.0
   cout << "x-y = " << x-y << endl;   // 54.0-20.0 = 34.0
   cout << "x*y = " << x*y << endl;   // 54.0*20.0 = 1080.0
   cout << "x/y = " << x/y << endl;   // 54.0/20.0 = 2.7
   return 0;
}
```

```
x = 54 and y = 20
x+y = 74
x-y = 34
x*y = 1080
x/y = 2.7
```

به پاسخ‌های بالا دقت کنید: بر خلاف تقسیم اعداد صحیح، تقسیم اعداد ممیز شناور به صورت بریده‌شده نیست: 54.0 / 20.0 = 2.7

تفاوت نوع float با نوع double در این است که نوع double دو برابر float از حافظه استفاده می‌کند. پس نوع double دقتی بسیار بیشتر از float دارد. به همین دلیل محاسبات double وقت‌گیرتر از محاسبات float است. بنابراین اگر در برنامه‌هایتان به محاسبات و پاسخ‌های بسیار دقیق نیاز دارید، از نوع double استفاده کنید. ولی اگر سرعت اجرا برایتان اهمیت بیشتری دارد، نوع float را به کار بگیرید.

8 – 2 شکل علمی مقادیر ممیز شناور

اعداد ممیز شناور به دو صورت در ورودی و خروجی نشان داده می‌شوند: به شکل «ساده» و به شکل «علمی».

مقدار 12345.67 شکل ساده عدد است و مقدار 1.234567×10^4 شکل علمی همان عدد است. مشخص است که شکل علمی برای نشان دادن اعداد خیلی کوچک و همچنین اعداد خیلی بزرگ، کارآیی بیشتری دارد:

$-0.000000000123 = -1.23\times10^{-10}$
$123000000000 = 1.23\times10^{11}$

در C++ برای نشان دادن حالت علمی اعداد ممیز شناور از حرف انگلیسی e یا E استفاده می‌کنیم:

-1.23×10^{-10} = -1.23e-10
1.23×10^{11} = 1.23e11

هنگام وارد کردن مقادیر ممیز شناور، می‌توانیم از شکل ساده یا شکل علمی استفاده کنیم. هنگام چاپ مقادیر ممیز شناور، معمولا مقادیر بین 0.1 تا 999.999 به شکل ساده چاپ می‌شوند و سایر مقادیر به شکل علمی نشان داده می‌شوند.

× مثال 6 – 2 شکل علمی اعداد ممیز شناور

برنامهٔ زیر یک عدد ممیز شناور (x) را از ورودی گرفته و معکوس آن (1/x) را چاپ می‌کند:

```
int main()
{  // prints reciprocal value of x:
   double x;
   cout << "Enter float: "; cin >> x;
   cout << "Its reciprocal is: " << 1/x << endl;
   return 0;
}
```

```
Enter float: 234.567e89
Its reciprocal is: 4.26317e-92
```

تا این‌جا انواع عددی را در C++ دیدیم. این انواع برای محاسبات استفاده می‌شوند و تقریبا در هر برنامه‌ای که می‌نویسید به کار می‌روند. اما C++ انواع دیگری نیز دارد که کاربردهای دیگری دارند. نوع بولین که برای عملیات منطقی استفاده می‌شود و نوع کاراکتری که برای به کار گرفتن کاراکترها تدارک دیده شده است و نوع شمارشی که بیشتر برای مجموعه‌هایی که برنامه‌نویس تعریف می‌کند به کار می‌رود. این انواع جدید گرچه کاربردشان با اعداد تفاوت دارد اما در حقیقت به شکل اعداد صحیح در رایانه ذخیره و شناسایی می‌شوند. به همین دلیل این نوع‌ها را نیز زیرمجموعه‌ای از انواع صحیح در C++ می‌شمارند. در ادامهٔ این فصل به بررسی این انواع می‌پردازیم.

9 – 2 نوع بولین[1] bool

نوع bool یک نوع صحیح است که متغیرهای این نوع فقط می‌توانند مقدار **true** یا **false** داشته باشند. true به معنی درست و false به معنی نادرست است. گرچه درون برنامه مجبوریم از عبارات true یا false برای مقداردهی به این نوع متغیر استفاده کنیم، اما این مقادیر در اصل به صورت 1 و 0 درون رایانه ذخیره می‌شوند: 1 برای true و 0 برای false. مثال زیر این مطلب را نشان می‌دهد.

× مثال 7 – 2 استفاده از متغیرهای نوع bool

```
int main()
{  //prints the vlaue of a boolean variable:
   bool flag=false;
   cout << "flag = " << flag << endl;
   flag = true;
   cout << "flag = " << flag << endl;
   return 0;
}
```

```
flag = 0
flag = 1
```

1 – Boolean

در خط سوم از برنامهٔ بالا متغیری به نام falg از نوع bool تعریف شده و با مقدار false مقداردهی اولیه شده است. در خط بعدی مقدار این متغیر در خروجی چاپ شده و در خط پنجم مقدار آن به true تغییر یافته است و دوباره مقدار متغیر چاپ شده است. گرچه به متغیر flag مقدار false و true دادهایم اما در خروجی به جای آن‌ها مقادیر 0 و 1 چاپ شده است.

10 -2 نوع کاراکتری char

یک ***کاراکتر*** یک حرف، رقم یا نشانه است که یک شمارهٔ منحصر به فرد دارد. به عبارت عامیانه، هر کلیدی که روی صفحه‌کلید خود می‌بینید یک کاراکتر را نشان می‌دهد (البته به غیر از کلیدهای مالتی‌مدیا یا کلیدهای اینترنتی که اخیرا در صفحه‌کلیدها مرسوم شده‌اند). مثلا هر یک از حروف 'A' تا 'Z' و 'a' تا 'z' و هر یک از اعداد '0' تا '9' و یا نشانه‌های '~' تا '+' روی صفحه‌کلید را یک کاراکتر می‌نامند. رایانه‌ها برای شناسایی کاراکترهای استاندارد از جدول اسکی استفاده می‌کنند. با دقت در این جدول خواهید دید که هر کاراکتر یک شمارهٔ منحصر به فرد دارد. مثلا کاراکتر 'A' کد 65 دارد. کاراکترها در رایانه به شکل عددی‌شان ذخیره می‌شوند اما به شکل کاراکتری‌شان نشان داده می‌شوند. مثلا کاراکتر 'A' به شکل عدد 65 ذخیره می‌شود اما اگر سعی کنیم متغیری که کاراکتر 'A' در آن ذخیره شده را چاپ کنیم، شکل A را در خروجی می بینیم نه عدد 65 را.

برای تعریف متغیری از نوع کاراکتر از کلمه کلیدی **char** استفاده می‌کنیم. یک کاراکتر باید درون دو علامت آپستروف (') محصور شده باشد. پس 'A' یک کاراکتر است؛ همچنین '8' یک کاراکتر است اما 8 یک کاراکتر نیست بلکه یک عدد صحیح است.

مثال بعدی نحوهٔ به کارگیری متغیرهای کاراکتری را نشان می‌دهد.

× مثال 8 – 2 استفاده از متغیرهای نوع char

```
int main()
{  //prints the character and its internally stored integer value:
   char c ='A';
```

```
    cout << "c = " << c << ", int(c) = " << int(c) << endl;
    c ='t';
    cout << "c = " << c << ", int(c) = " << int(c) << endl;
    c ='\t';   // the tab character
    cout << "c = " << c << ", int(c) = " << int(c) << endl;
    c ='!';
    cout << "c = " << c << ", int(c) = " << int(c) << endl;
    return 0;
}
```

```
c = A, int(c) = 65
c = t, int(c) = 116
c =      , int(c) = 9
c = !, int(c) = 33
```

در خط سوم از برنامهٔ بالا متغیری به نام c از نوع char تعریف شده و با مقدار 'A' مقدارگذاری اولیه شده است. سپس در خط بعدی ابتدا مقدار c چاپ شده که در خروجی همان A دیده می‌شود نه مقدار عددی آن. در ادامهٔ خط چهارم، با استفاده از دستور int(c) مقدار عددی c یعنی 65 در خروجی چاپ خواهد شد. در خطوط بعدی کاراکترهای دیگری به c اختصاص یافته و به همین ترتیب مقدار c و معادل عددی آن چاپ شده است.

به خط هفتم برنامه نگاه کنید: `c = '\t';`

کاراکتر '\t' یک کاراکتر خاص است. اگر سعی کنیم کاراکتر '\t' را روی صفحه‌نمایش نشان دهیم، هفت جای خالی روی صفحه دیده می‌شود(به خروجی دقت کنید). غیر از این کاراکتر، کاراکترهای خاص دیگری نیز هستند که کارهایی مشابه این انجام می‌دهند. مثل کاراکتر '\n' که در فصل قبل دیدیم و مکان‌نما را به سطر بعدی منتقل می‌کند. این کاراکترها برای شکل‌دهی صفحه‌نمایش و کنترل آن استفاده می‌شوند. کاراکترهای خاص در جدول اسکی بین شماره‌های 0 تا 32 قرار گرفته‌اند. سعی کنید مثل برنامهٔ بالا یک برنامهٔ آزمایشی بنویسید و با استفاده از آن مقادیر کاراکترهای خاص را چاپ کنید تا ببینید چه اتفاقی می‌افتد. لابه‌لای برنامه‌های بعدی از این کاراکترهای خاص استفاده خواهیم کرد تا با آن‌ها بهتر آشنا شوید.

11 – 2 نوع شمارشی enum

علاوه بر انواعی که تا کنون بررسی کردیم، می توان در C++ انواع جدیدی که کاربر نیاز دارد نیز ایجاد نمود. برای این کار راه‌های مختلفی وجود دارد که بهترین و قوی‌ترین راه، استفاده از کلاس‌ها است(فصل ده)، اما راه ساده‌تری نیز وجود دارد و آن استفاده از نوع شمارشی enum است.

یک نوع ***شمارشی*** یک نوع صحیح است که توسط کاربر مشخص می‌شود. نحو تعریف یک نوع شمارشی به شکل زیر است:

```
enum typename{enumerator-list}
```

که **enum** کلمه‌ای کلیدی است، typename نام نوع جدید است که کاربر مشخص می‌کند و *enumerator-list* مجموعه مقادیری است که این نوع جدید می‌تواند داشته باشد. به عنوان مثال به تعریف زیر دقت کنید:

```
enum Day{SAT,SUN,MON,TUE,WED,THU,FRI}
```

حالا Day یک نوع جدید است و متغیرهایی که از این نوع تعریف می‌شوند می‌توانند یکی از مقادیر SAT و SUN و MON و TUE و WED و THU و FRI را داشته باشند:

```
Day day1,day2;
day1 = MON;
day2 = THU;
```

وقتی نوع جدید Day و محدودهٔ مقادیرش را تعیین کردیم، می‌توانیم متغیرهایی از این نوع جدید بسازیم. در کد بالا متغیرهای day1 و day2 از نوع Day تعریف شده‌اند. آنگاه day1 با مقدار MON و day2 با مقدار THU مقداردهی شده است.

مقادیر SAT و SUN و ... هر چند که به همین شکل به کار می‌روند اما در رایانه به شکل اعداد صحیح 0 و 1 و 2 و ... ذخیره می‌شوند. به همین دلیل است که به هر یک از مقادیر SAT و SUN و ... یک ***شمارشگر***[1] می‌گویند. وقتی فهرست شمارشگرهای یک نوع تعریف شد، به طور خودکار مقادیر 0 و 1 و 2 و ... به ترتیب

1 – Enumerator

به آن‌ها اختصاص می‌یابد. هرچند که می‌توان این ترتیب را شکست و مقادیر صحیح دلخواهی را به شمارشگرها نسبت داد:

```
enum Day{SAT=1,SUN=2,MON=4,TUE=8,WED=16,THU=32,FRI=64}
```

اگر فقط بعضی از شمارشگرها مقداردهی شوند، آنگاه سایر شمارشگرها که مقداردهی نشده‌اند مقادیر متوالی بعدی را خواهند گرفت:

```
enum Day{SAT=1,SUN,MON,TUE,WED,THU,FRI}
```

دستور بالا مقادیر 1 تا 7 را به ترتیب به روزهای هفته تخصیص خواهد داد.

همچنین دو یا چند شمارشگر در یک فهرست می‌توانند مقادیر یکسانی داشته باشند:

```
enum Answer{NO=0,FALSE=0,YES=1,TRUE=1,OK=1}
```

در کد بالا دو شمارشگر NO و FALSE دارای مقدار یکسان 0 و شمارشگرهای YES و TRUE و OK نیز دارای مقدار یکسان 1 هستند. پس کد زیر معتبر است و به درستی کار می‌کند:

```
Answer answer;
cin >> answer;
if (answer==TRUE) cout << "you said OK.";
```

به اولین خط کد فوق نگاه کنید. این خط ممکن است کمی عجیب به نظر برسد:

```
Answer answer;
```

این خط متغیری به نام answer از نوع Answer تعریف می‌کند. اولین قانون در برنامه‌های C++ را به خاطر بیاورید: «C++ بین حروف کوچک و بزرگ تفاوت قایل است». پس Answer با answer متفاوت است. Answer را در خط‌های قبلی یک نوع شمارشی تعریف کردیم و answer را متغیری که از نوع Answer است. یعنی answer متغیری است که می‌تواند یکی از مقادیر YES یا TRUE یا OK یا FALSE یا NO را داشته باشد. نحوهٔ انتخاب نام‌ها آزاد است اما بیشتر برنامه‌نویسان از توافق زیر در برنامه‌هایشان استفاده می‌کنند:

1 – برای نام ثابت‌ها از حروف بزرگ استفاده کنید

2 – اولین حرف از نام نوع شمارشی را با حرف بزرگ بنویسید.

3 – در هر جای دیگر از حروف کوچک استفاده کنید.

رعایت این توافق به خوانایی برنامه‌تان کمک می‌کند. همچنین سبب می‌شود که انواع شمارشی که کاربر تعریف می‌کند از انواع استاندارد مثل int و float و char راحت‌تر تمیز داده شود.

شمارشگرها قواعد خاصی دارند. نام شمارشگر باید معتبر باشد. یعنی کلمهٔ کلیدی نباشد، با عدد شروع نشود و نشانه‌های ریاضی نیز نداشته باشد. پس تعریف زیر غیرمعتبر است:

```
enum Score{A+,A,A-,B+,B,B-,C+,C,C-}
```

زیرا A+ و A- و B+ و B- و C+ و C- نام‌های غیرمعتبری هستند چون در نام آن‌ها از نشانه‌های ریاضی استفاده شده.

علاوه بر این شمارشگرهای هم‌نام نباید در محدوده‌های مشترک استفاده شوند. برای مثال تعریف‌های زیر را در نظر بگیرید:

```
enum Score{A,B,C,D}
enum Group{AB,B,BC}
```

دو تعریف بالا غیرمجاز است زیرا شمارشگر B در هر دو تعریف Score و Group آمده است.

آخر این که نام شمارشگرها نباید به عنوان نام متغیرهای دیگر در جاهای دیگر برنامه استفاده شود. مثلا:

```
enum Score{A,B,C,D}
float B;
char c;
```

در تعریف‌های بالا B و C را نباید به عنوان نام متغیرهای دیگر به کار برد زیرا این نام‌ها در نوع شمارشی Score به کار رفته است. پس اگر این سه تعریف در یک محدوده

باشند، دو تعریف آخری غیرمجاز خواهد بود. انواع شمارشی برای تولید کد «خود مستند» به کار می‌روند، یعنی کدی که به راحتی درک شود و نیاز به توضیحات اضافی نداشته باشد. مثلا تعاریف زیر خودمستند هستند زیرا به راحتی نام و نوع کاربرد و محدودهٔ مقادیرشان درک می‌شود:

```
enum Color{RED,GREEN,BLUE,BLACK,ORANGE}
enum Time{SECOND,MINUTE,HOUR}
enum Date{DAY,MONTH,YEAR}
enum Language{C,DELPHI,JAVA,PERL}
enum Gender{MALE,FEMALE}
```

12 – 2 تبدیل نوع، گسترش نوع

در قسمت‌های قبلی با انواع عددی آشنا شدیم و نحوهٔ اعمال ریاضی آن‌ها را مشاهده نمودیم اما در محاسبات ریاضی که انجام دادیم همهٔ متغیرها از یک نوع بودند. اگر بخواهیم در یک محاسبه دو یا چند متغیر از انواع مختلف به کار ببریم چه اتفاقی می‌افتد؟

قانون کلی این است که در محاسباتی که چند نوع متغیر وجود دارد، جواب همیشه به شکل متغیری است که دقت بالاتری دارد. یعنی اگر یک عدد صحیح را با یک عدد ممیز شناور جمع ببندیم، پاسخ به شکل ممیز شناور است. به این منظور ابتدا متغیرها و مقادیری که از نوع با دقت کمتر هستند به نوع با دقت بیشتر تبدیل می‌شوند و سپس محاسبه روی آن‌ها انجام می‌شود. پس اگر یک عدد صحیح را با یک عدد ممیز شناور جمع ببندیم، ابتدا عدد صحیح تبدیل به یک عدد ممیز شناور می‌شود، سپس این عدد با عدد ممیز شناور دیگر جمع بسته می‌شود و واضح است که پاسخ نیز به شکل ممیز شناور خواهد بود. این کار به شکل خودکار انجام می‌گیرد و C++ در چنین محاسباتی به شکل خودکار متغیرهای با دقت کمتر را به متغیرهایی با دقت بیشتر تبدیل می‌کند تا همه متغیرها از یک نوع شوند و آنگاه محاسبه را انجام می‌دهد و پاسخ را نیز به شکل نوع با دقت بیشتر به دست می‌دهد. به این عمل ***گسترش نوع*** می‌گویند.

اما اگر عکس این عمل مورد نظر باشد، یعنی اگر بخواهیم یک متغیر صحیح را با یک متغیر ممیز شناور جمع ببندیم و بخواهیم که حاصل از نوع صحیح باشد نه ممیز شناور، چه باید بکنیم؟ در چنین حالتی از عملگر تبدیل نوع استفاده می‌کنیم. این تبدیل خودکار نیست بلکه کاملا باید دستی انجام شود و برنامه‌نویس، خود باید مراقب این عمل باشد. برای این که مقدار یک متغیر از نوع ممیز شناور را به نوع صحیح تبدیل کنیم از عبارت int () استفاده می‌کنیم.

مثال‌های زیر تبدیل نوع و گسترش نوع را نشان می‌دهند.

× مثال 9 – 2 تبدیل نوع

این برنامه، یک نوع double را به نوع int تبدیل می‌کند:

```
int main()
{  // casts a double value as an int:
   double v=1234.987;
   int n;
   n = int(v);
   cout << "v = " << v << ", n = " << n << endl;
   return 0;
}
```

```
v = 1234.987, n = 1234
```

در این برنامه متغیر v از نوع double و با مقدار 1234.987 تعریف شده است. همچنین متغیر n از نوع int تعریف گشته است. در خط پنجم از کد بالا از تبدیل نوع استفاده شده:

```
n = int(v);
```

با استفاده از این دستور، مقدار v ابتدا به نوع int تبدیل می‌شود و سپس این مقدار درون n قرار می‌گیرد. خروجی برنامه نشان می‌دهد که وقتی از عملگر int () استفاده کنیم، عدد ممیز شناور «بریده» می‌شود، گرد نمی‌شود. یعنی قسمت اعشاری عدد به طور کامل حذف می‌شود و فقط قسمت صحیح آن باقی می‌ماند. بنابراین وقتی

عدد 1234.987 به نوع int تبدیل شود، حاصل برابر با 1234 خواهد بود و قسمت اعشاری آن (هر قدر هم بزرگ باشد) نادیده گرفته می‌شود.

در تبدیل نوع همواره نوع و مقدار متغیرهای تبدیل شده بدون تغییر می‌ماند. در برنامهٔ بالا مقدار v تا پایان برنامه به همان مقدار 1234.987 باقی مانده و نوع v نیز تغییر نکرده و همچنان از نوع double مانده است. تنها اتفاقی که افتاده این است که مقدار v در یک محل موقتی تبدیل به int شده تا این مقدار درون n قرار گیرد.

× مثال 10 – 2 گسترش نوع

برنامهٔ زیر یک عدد صحیح را با یک عدد ممیز شناور جمع می‌کند:

```
int main()
{  // adds an int value with a double value:
   int n = 22;
   double p = 3.1415;
   p += n;
   cout << "p = " << p << ", n = " << n << endl;
   return 0;
}
```

```
p = 24.1415, n = 22
```

در برنامهٔ بالا ابتدا مقدار n از مقدار صحیح 22 به مقدار اعشاری 22.0 گسترش می‌یابد و سپس این مقدار با مقدار قبلی p جمع می‌شود. حاصل یک عدد ممیز شناور است.

× مثال 11 – 2 گسترش نوع

این برنامه، یک char را به int ، float و double گسترش می‌دهد:

```
int main()
{  //prints promoted values of 65 from char to double:
   char c='A';     cout << "   char c = " << c << endl;
   short k=c;      cout << "  short k = " << k << endl;
   int m=k;        cout << "    int m = " << m << endl;
   long n=m;       cout << "   long n = " << n << endl;
   float x=n;      cout << "  float x = " << x << endl;
```

```
   double y=x;     cout << " double y = " << y << endl;
   return 0;
}
```

```
char c = A
short k = 65
 int m = 65
 long n = 65
float x = 65
double y = 65
```

در مثال بالا ابتدا متغیر c از نوع char تعریف شده و کاراکتر 'A' در آن قرار گرفته است. سپس مقدار c درون متغیر k که از نوع short است قرار گرفته. چون نوع k بالاتر از نوع c است، پس مقدار c به نوع short گسترش می‌یابد و مقدار 65 که معادل عددی کاراکتر 'A' است درون k قرار می‌گیرد.

در خط بعدی، مقدار k درون متغیر m قرار می‌گیرد. m از نوع int است که نوع بالاتری از short می‌باشد. پس مقدار k به int گسترش می‌یابد و این مقدار گسترش‌یافته درون m نهاده می‌شود.

به همین ترتیب در خطوط بعدی مقدار m به نوع long گسترش یافته و درون n قرار می‌گیرد. مقدار n نیز به نوع float گسترش یافته و درون x قرار می‌گیرد. مقدار x نیز به نوع double گسترش می‌یابد و درون y قرار می‌گیرد. دقت کنید که مقدار x و y در خروجی به جای آن که 65.0 باشد به شکل 65 نشان داده شده. این مقدار، یک عدد صحیح نیست اما چون قسمت اعشاری آن صفر است، اعشار حذف شده و 65 تنها نشان داده شده است.

13 – 2 برخی از خطاهای برنامه‌نویسی

اکنون که انواع متغیر در C++ را شناختیم، می‌توانیم از این انواع در برنامه‌های مفیدتر و جدی‌تر استفاده کنیم. اما باید دقت نمود که اگر از متغیرها به شکل نادرست یا کنترل‌نشده استفاده کنیم، برنامه دچار خطا می‌شود. البته عوامل دیگری نیز هست که باعث می‌شود اجرای برنامه مختل گردد، مثل استفاده از متغیری که تعریف نشده یا جا انداختن سمیکولن انتهای دستورها. این قبیل خطاها که اغلب خطاهای نحوی هستند و

توسط کامپایلر کشف می‌شوند «خطای زمان کامپایل» نامیده می‌شوند و به راحتی می‌توان آن‌ها را رفع نمود. اما خطاهای دیگری نیز وجود دارند که کشف آن‌ها به راحتی ممکن نیست و کامپایلر نیز چیزی راجع به آن نمی‌داند. به این خطاها «خطای زمان اجرا» می‌گویند. برخی از خطاهای زمان اجرا سبب می‌شوند که برنامه به طور کامل متوقف شود و از کار بیفتد. در چنین حالتی متوجه می‌شویم که خطایی رخ داده است و در صدد کشف و رفع آن برمی‌آییم. برخی دیگر از خطاهای زمان اجرا، برنامه را از کار نمی‌اندازند بلکه برنامه همچنان کار می‌کند اما پاسخ‌های عجیب و نادرست می‌دهد. این بدترین نوع خطاست زیرا در حالات خاصی رخ می‌دهد و گاهی سبب گیج شدن برنامه‌نویس می‌گردد. در بخش‌های بعدی برخی از خطاهای رایج زمان اجرا را نشان می‌دهیم تا در برنامه‌هایتان از آن‌ها پرهیز کنید؛ دست کم اگر با پاسخ‌های غیرمنتظره و غلط مواجه شدید، محل رخ دادن خطا را راحت‌تر پیدا کنید.

14 – 2 سرریزی[1] عددی

نوع صحیح long یا نوع ممیز شناور double محدودهٔ وسیعی از اعداد را می‌توانند نگهداری کنند. به بیان ساده‌تر، متغیری که از نوع long یا double باشد، گنجایش زیادی دارد. اما حافظهٔ رایانه‌ها متناهی است. یعنی هر قدر هم که یک متغیر گنجایش داشته باشد، بالاخره مقداری هست که از گنجایش آن متغیر بیشتر باشد. اگر سعی کنیم در یک متغیر مقداری قرار دهیم که از گنجایش آن متغیر فراتر باشد، متغیر «سرریز» می‌شود. مثل یک لیوان آب که اگر بیش از گنجایش آن در لیوان آب بریزیم، سرریز می‌شود. در چنین حالتی می‌گوییم که خطای سرریزی رخ داده است.

× مثال 12 – 2 سرریزی عدد صحیح

این برنامه به طور مکرر n را در 1000 ضرب می‌کند تا سرانجام سرریز شود:

```
int main()
{  //prints n until it overflows:
   int n =1000;
   cout << "n = " << n << endl;
```

1 – Overflow

```
   n *= 1000; // multiplies n by 1000
   cout << "n = " << n << endl;
   n *= 1000; // multiplies n by 1000
   cout << " n = " << n << endl;
   n *= 1000; // multiplies n by 1000
   cout << " n = " << n << endl;
   return 0;
}
```

```
n = 1000
n = 1000000
n = 1000000000
n = -727379968
```

این مثال نشان می‌دهد رایانه‌ای که این برنامه را اجرا کرده است، نمی‌تواند بیشتر از1,000,000,000 را با 1000 به طور صحیح ضرب کند.

× مثال 13 – 2 سرریزی عدد ممیز شناور

این برنامه شبیه چیزی است که در مثال قبل ذکر شد؛ به طور مکرر x را به توان می‌رساند تا این که سرریز شود.

```
int main()
{  //prints x until it overflows:
   float x=1000.0;
   cout << "x = " << x << endl;
   x *= x;   //multiplies n by itself; i.e., it squares x
   cout << "x = " << x << endl;
   x *= x;   //multiplies n by itself; i.e., it squares x
   cout << "x = " << x << endl;
   x *= x;   //multiplies n by itself; i.e., it squares x
   cout << "x = " << x << endl;
   x *= x;   //multiplies n by itself; i.e., it squares x
   cout << "x = " << x << endl;
   return 0;
}
```

```
x = 1000
x = 1e+06
x = 1e+12
```

```
x = 1e+24
x = inf
```

مثال بالا نشان می‌دهد که این رایانه نمی‌تواند x را با شروع از 1000 بیش از سه بار مجذور کند. آخرین خروجی یعنی inf نمادی است که به معنای بی‌نهایت می‌باشد (این نماد مخفف infinity به معنای بی‌انتها است).

به تفاوت سرریزی عدد صحیح و سرریزی ممیز شناور توجه کنید. وقتی یک عدد صحیح سرریز شود، عدد سرریز شده به یک مقدار منفی «گردانیده» می‌شود اما وقتی یک عدد ممیز شناور سرریز شود، نماد inf به معنای بی‌نهایت را به دست می‌دهد، نشانه‌ای مختصر و مفید.

15 – 2 خطای گرد کردن[1]

خطای گرد کردن نوع دیگری از خطاست که اغلب وقتی رایانه‌ها روی اعداد حقیقی محاسبه می‌کنند، رخ می‌دهد. برای مثال عدد 1/3 ممکن است به صورت 0.333333 ذخیره شود که دقیقا معادل 1/3 نیست. به این اختلاف، ***خطای گرد کردن*** می‌گویند. این خطا از آن‌جا ناشی می‌شود که اعدادی مثل 1/3 مقدار دقیق ندارند و رایانه نمی‌تواند این مقدار را پیدا کند، پس نزدیک‌ترین عدد قابل محاسبه را به جای چنین اعدادی منظور می‌کند. در بعضی حالات، این خطاها می‌تواند مشکلات حادی را ایجاد کند.

× مثال 14 – 2 خطای گرد کردن

این برنامه محاسبات ساده‌ای را انجام می‌دهد تا خطای گرد کردن را نشان دهد:

```
int main()
{  //illustrates round-off error:
   double x = 1000/3.0;
   cout << "x = " << x << endl;          // x = 1000/3
   double y = x-333.0;
   cout << "y = " << y << endl;          // y = 1/3
   double z = 3*y-1.0;
```

1 – Round-off

```
    cout << "z = " << z << endl;           // z = 3(1/3) - 1
    if (z == 0) cout << "z == 0.\n";
    else cout << "z does not equal 0.\n";  //z != 0
    return 0;
}
```

```
x = 333.333
y = 0.333333
z = -5.68434e-14
z does not equal 0.
```

منطق برنامه به این شکل است که ابتدا مقدار x برابر با 1000/3 یعنی $333\frac{1}{3}$ است. سپس قسمت صحیح x یعنی 333 از آن کسر می‌شود و حاصل که برابر با 1/3 است در y قرار می‌گیرد. حالا y در 3 ضرب می‌شود تا حاصل برابر با 1 شود. این مقدار از 1 کم می‌شود و حاصل در z قرار می‌گیرد. انتظار این است که z صفر باشد اما پاسخ برنامه به ما می‌گوید که z صفر نیست!

اشکال برنامهٔ بالا در کجاست؟ منطق برنامه که درست است، پس جایی در محاسبات باید غلط باشد. مشکل در مقدار y است. رایانه مقدار 1/3 را برابر با 0.333333 محاسبه نموده است، حال آن که می‌دانیم این مقدار دقیقا برابر با 1/3 نیست. این خطا از آن جا ناشی می‌شود که رایانه نمی‌تواند مقدار دقیق 1/3 را پیدا کند چون این مقدار به تعداد نامتناهی اعشار 3 دارد، پس رایانه این مقدار را گرد می‌کند و مقدار «نسبتا درست» 0.333333 را می‌دهد. این مقدار در محاسبات بعدی استفاده می‌شود اما چون دقیق نیست، پاسخ‌های بعدی نیز به تناسب بر میزان خطا می‌افزاید. نتیجه این است که مقدار z صفر نمی‌شود، هرچند که بسیار نزدیک به صفر باشد.

مثال بالا نکتهٔ مهمی را در استفاده از متغیرهای ممیز شناور نشان می‌دهد: «هیچ‌گاه از متغیر ممیز شناور برای مقایسه برابری استفاده نکنید» زیرا در متغیرهای ممیز شناور خطای گرد کردن سبب می‌شود که پاسخ با آن چه مورد نظر شماست متفاوت باشد. در حالت بالا گر چه مقدار z بسیار نزدیک صفر است، اما رایانه همین مقدار کوچک را صفر نمی‌داند. پس مقایسهٔ برابری شکست می‌خورد.

× مثال 15 – 2 خطای گرد کردن پنهان

برنامهٔ زیر با استفاده از رابطهٔ معادلات درجه دوم، ریشه‌های این معادله‌ها را پیدا می‌کند:

```
#include <cmath>   //defines the sqrt() function
#include <iostream>
using namespace std;
int main()
{  //implements the quadratic formula
   float a, b, c;
   cout << "Enter the coefficients of a quadratic equation:"
        << endl;
   cout << "\ta: ";
   cin >> a;
   cout << "\tb: ";
   cin >> b;
   cout << "\tc: ";
   cin >> c;
   cout << "The equation is: " << a << "*x*x + " << b
        << "*x + " << c << " = 0" << endl;
   float d = b*b - 4*a*c;   // discriminant
   float sqrtd = sqrt(d);
   float x1 = (-b + sqrtd /(2*a);
   float x2 = (-b - sqrtd /(2*a);
   cout << "The solutions are:" << endl;
   cout << "\tx1 = " << x1 << endl;
   cout << "\tx2 = " << x2 << endl;
   cout << "check:" << endl;
   cout << "\ta+x1*x1 + b*x1 + c = " << a*x1*x1 + b*x1 + c
        << endl;
   cout << "\ta+x2*x2 + b*x2 + c = " << a*x2*x2 + b*x2 + c
        << endl;
   return 0;
}
```

این برنامه ضرایب a و b و c را برای معادلهٔ $ax^2+bx+c=0$ می‌گیرد و سپس سعی می‌کند ریشه‌های x1 و x2 را پیدا کند. برای این کار سه متغیر a و b و c از نوع float تعریف شده‌اند تا بتوانند مقادیر اعشاری را هم از ورودی بگیرند. خط هشتم تا سیزدهم از برنامهٔ بالا مقادیر a و b و c را دریافت می‌کنند (دقت کنید که از کاراکتر خاص '\t' در پیغام‌های خروجی استفاده شده تا قبل از هر ورودی، هفت جای خالی قرار بگیرد. ولی خود حرف t چاپ نمی‌شود). پس از دریافت ضرایب، یک بار دیگر شکل کلی معادله‌ای که مورد نظر کاربر بوده است چاپ می‌شود.

در خط یازدهم، رابطهٔ دلتا یعنی b^2-4ac تشکیل شده است. این مقدار درون متغیر دیگری که d نام دارد و از نوع float است، قرار گرفته. در خط بعدی مقدار جذر دلتا با استفاده از تابع sqrt() محاسبه شده است. تابع sqrt() جذر عددی که درون پرانتزهایش قرار می‌گیرد را به دست می‌دهد. این تابع در سرفایل <cmath> تعریف شده. پس راهنمای پیش‌پردازندهٔ #include<cmath> به ابتدای برنامه افزوده شده است. مقدار جذر دلتا درون متغیر دیگری به نام sqrtd نگهداری شده تا با استفاده از آن در خطوط بعدی مقادیر x1 و x2 به دست آید.

در چهار خط آخر برنامه، مقادیر x1 و x2 که بدست آمده است دوباره در معادله جای‌گذاری می‌شود تا بررسی شود که آیا جواب معادله صفر می‌شود یا خیر. به این وسیله صحت پاسخ‌های x1 و x2 تحقیق می‌شود.

خروجی زیر نشان می‌دهد که برنامه، معادلهٔ $2x^2+1x-3=0$ را حل کرده است:

```
Enter the coeficients of a quadratic equation:
   a: 2
   b: 1
   c: -3
The equation is: 2*x*x + 1*x + -3 = 0
The solutions are:
   x1 = 1
   x2 = -1.5
check:
   a*x1*x1 + b*x1 + c = 0
   a*x2*x2 + b*x2 + c = 0
```

می‌بینید که برنامه پاسخ‌های x1=1 و x2=-1.5 را پیدا کرده است و آزمون پاسخ نیز جواب صفر داده است. خروجی دیگری از برنامه نشان می‌دهد که برنامه تلاش کرده معادله $2x^2 + 8.001x + 8.002 = 0$ را حل کند ولی شکست می‌خورد:

```
Enter the coeficients of a quadratic equation:
   a: 2
   b: 8.001
   c: 8.002
The equation is: 2*x*x + 8.001*x + 8.002 = 0
The solutions are:
   x1 = -1.9995
   x2 = -2.00098
check:
   a*x1*x1 + b*x1 + c = 5.35749e-11
   a*x2*x2 + b*x2 + c = -2.96609e-1
```

مقدار x1 که در اجرای بالا به دست آمده، در آزمون شرکت کرده و پاسخی بسیار نزدیک به صفر داده است. اما مقدار x2 در آزمون شکست خورده زیرا جواب معادله به ازای آن صفر نیست. چه چیزی باعث شده تا معادله پاسخ غلط بدهد؟ جواب باز هم در خطای گرد کردن است. x2 یک پاسخ گردشده است نه یک پاسخ دقیق. این پاسخ گردشده دوباره در یک محاسبه دیگر شرکت می‌کند. پاسخ این محاسبه هم گردشده است. پس انحراف از جواب افزایش می‌یابد و نتیجه‌ای دور از انتظار به بار می‌آورد.

× مثال 16 – 2 انواع دیگری از خطاهای زمان اجرا

دوباره به برنامهٔ محاسبهٔ ریشه‌ها برگردیم. به اجرای زیر نگاه کنید:

```
Enter the coeficients of a quadratic equation:
   a: 1
   b: 2
   c: 3
The equation is: 1*x*x + 2*x + 3 = 0
The solutions are:
   x1 = nan
   x2 = nan
check:
   a*x1*x1 + b*x1 + c = nan
   a*x2*x2 + b*x2 + c = nan
```

در این اجرا سعی شده تا معادله $1x^2+2x+3=0$ حل شود. این معادله جواب حقیقی ندارد زیرا دلتا منفی است. وقتی برنامه اجرا شود، تابع sqrt() تلاش می‌کند جذر یک عدد منفی را بگیرد ولی موفق نمی‌شود. در این حالت پاسخ nan داده می‌شود(nan مخفف عبارت not a number است یعنی پاسخ عددی نیست). سپس هر محاسبهٔ دیگری که از این مقدار استفاده کند، همین پاسخ nan را خواهد داشت. به همین دلیل در همهٔ خروجی‌ها پاسخ nan آمده است.

سرانجام به اجرای زیر دقت نمایید:

```
Enter the coeficients of a quadratic equation:
   a: 0
   b: 2
   c: 5
The equation is: 0*x*x + 2*x + 5 = 0
The solutions are:
   x1 = nan
   x2 = -inf
check:
   a*x1*x1 + b*x1 + c = nan
```

در این اجرا کوشش شده تا معادلهٔ $0x^2+2x+5=0$ حل شود. این معادله دارای جواب x=2.5 است اما برنامه نمی‌تواند این جواب را بیابد و با پاسخ‌های عجیبی روبرو می‌شویم. علت این است که a صفر است و در حین اجرای برنامه، سعی می‌شود عددی بر صفر تقسیم شود. یعنی برنامه معادلهٔ زیر را حل می‌کند:

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}=\frac{-(2)+\sqrt{(2)^2-4(0)(5)}}{2(0)}=\frac{-2+2}{0}=\frac{0}{0}$$

در چنین حالتی دوباره پاسخ nan بدست می‌آید. همچنین برای x2 داریم:

$$x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}=\frac{-(2)-\sqrt{(2)^2-4(0)(5)}}{2(0)}=\frac{-2-2}{0}=\frac{-4}{0}$$

پاسخ این تقسیم، عبارت inf- یعنی بی‌نهایت منفی است.

سه نشانهٔ nan و inf و -inf ثابت‌های عددی هستند. یعنی می‌توانید این مقادیر را در محاسبات به کار ببرید اما نتیجه معمولا بی‌فایده است. مثلا می‌توانید عددی را با inf جمع کنید یا از آن تفریق نمایید اما نتیجه باز هم inf خواهد بود.

16 – 2 حوزهٔ متغیرها

متغیرها بخش مهمی از هر برنامه هستند. استفاده از متغیرهایی با نوع نامناسب سبب هدر رفتن حافظه و کاهش سرعت و افزایش خطاهای زمان اجرا می‌شود. انتخاب نام‌های نامفهوم یا ناقص سبب کاهش خوانایی برنامه و افزایش خطاهای برنامه‌نویسی می‌شود. استفاده از متغیرها در حوزهٔ نامناسب هم سبب بروز خطاهایی می‌شود. «حوزه[1] متغیر» محدوده‌ای است که یک متغیر خاص اجازه دارد در آن محدوده به کار رود یا فراخوانی شود.

اصطلاح «بلوک[2]» در C++ واژه مناسبی است که می‌توان به وسیلهٔ آن حوزهٔ متغیر را مشخص نمود. یک بلوک برنامه، قسمتی از برنامه است که درون یک جفت علامت کروشه { } محدود شده است. در برنامه‌هایی که تاکنون دیدیم از بلوک استفاده کرده‌ایم. همیشه بعد از عبارت int main() یک کروشه باز { گذاشته‌ایم و در پایان برنامه یک کروشه بسته } قرار دادیم. پس تمام برنامه‌هایی که تا کنون ذکر شد، یک بلوک داشته. به طور کلی می‌توان گفت که حوزهٔ یک متغیر از محل اعلان آن شروع می‌شود و تا پایان همان بلوک ادامه می‌یابد. خارج از آن بلوک نمی‌توان به متغیر دسترسی داشت. همچنین قبل از این که متغیر اعلان شود نمی‌توان آن را استفاده نمود. مثال زیر را بررسی کنید.

× مثال 17 – 2 حوزهٔ متغیرها

برنامهٔ زیر خطادار است:

```
int main()
{  //illustrates the scope of variables:
   x = 11;        // ERROR: this is not in the scope of x
   int x;
```

```
{
    x = 22;      // OK: this is in the scope of x
    y = 33;      // ERROR: this is not in the scope of y
    int y;
```

---

1 – Scope 2 - Block

```
    x = 44;      // OK: this is in the scope of x
    y = 55;      // OK: this is in the scope of y
}
x = 66;          // OK: this is in the scope of x
y = 77;          // ERROR: this is not in the scope of y
return 0;
}
```

برنامهٔ بالا دو بلوک تودرتو دارد. اولین بلوک بعد از عبارت `int main()` شروع می‌شود و در خط آخر برنامه بسته می‌شود. بلوک داخلی نیز از خط پنجم آغاز می‌شود و در خط دهم پایان می‌یابد. نحوهٔ تورفتگی خطوط برنامه به درک و تشخیص شروع و پایان بلوک‌ها کمک می‌کند. خط پنجم تا دهم تورفتگی بیشتری دارد، یعنی این خطوط تشکیل یک بلوک می‌دهند. همچنین خط دهم به بعد تورفتگی به اندازهٔ خط سوم و چهارم دارد، یعنی مجموعه این خطوط هم در یک حوزهٔ مشترک قرار دارند.

اولین خطا در خط سوم رخ داده. متغیر `x` در خط چهارم اعلان شده است. پس حوزهٔ `x` از خط چهارم به بعد شروع می‌شود، در حالی که در خط سوم متغیر `x` فراخوانی شده و این خارج از محدودهٔ `x` است.

دومین خطا در خط ششم اتفاق افتاده است. متغیر `y` در خط هفتم اعلان شده. پس حوزهٔ `y` از خط هفتم به بعد است، در حالی که در خط ششم `y` فراخوانی شده و این خارج از محدودهٔ `y` است.

سومین خطا که در خط دوازدهم روی داده نیز مربوط به `y` است. گرچه `y` در خطوط قبلی تعریف شده اما این تعریف در یک بلوک داخلی بوده است. این بلوک داخلی در خط دهم به پایان رسیده است. پس تمام تعاریفی که در این بلوک وجود

داشته نیز فقط تا خط دهم اعتبار دارد. یعنی حوزهٔ y فقط از خط هفتم تا خط دهم است. لذا نمی‌توان در خط دوازدهم که خارج از محدوده y است، آن را به کار برد.

مثال بالا مطلب ظریفی را بیان می‌کند: می‌توانیم در یک برنامه، چند متغیر متفاوت با یک نام داشته باشیم به شرطی که در حوزه‌های مشترک نباشند. آخرین برنامهٔ فصل اول این موضوع را به خوبی نشان می‌دهد.

× مثال 18 – 2 متغیرهای تودرتو

```
int x = 11;                          // this x is global

int main()
{  //illustrates the nested and parallel scopes:
   int x = 22;
   {  //begin scope of internal block
      int x = 33;
      cout << "In block inside main() : x = " << x << endl;
   }                                 //end scope of internal block
   cout << "In main() : x = " << x << endl;
   cout << "In main() : ::x = " << ::x << endl;
   return 0;
}                                    //end scope of main()
```

```
In block inside main() : x = 33
In main() : x = 22
In main() : ::x = 11
```

در برنامهٔ بالا سه شیء متفاوت با نام x وجود دارد. اولین x که مقدار 11 دارد یک متغیر سراسری است زیرا داخل هیچ بلوکی قرار ندارد. پس حوزهٔ آن سراسر برنامه (حتی خارج از بلوک main()) است. دومین x درون بلوک main() با مقدار 22 تعریف شده است. پس حوزهٔ آن تا پایان بلوک main() است. این x حوزه x قبلی را کور می‌کند. یعنی درون بلوک main() فقط x دوم دیده می‌شود و x اول مخفی می‌شود. پس اگر درون این بلوک به x ارجاع کنیم فقط x دوم را خواهیم دید.

سومین x در یک بلوک داخلی تعریف شده است. حوزه این x فقط تا پایان همان بلوک است. این x حوزهٔ هر دو x قبلی را کور می‌کند. پس اگر درون این بلوک

x را فراخوانی کنیم فقط x سوم را خواهیم دید. وقتی از این بلوک خارج شویم ، x قبلی آزاد می‌شود و دوباره می‌توان به مقدار آن دسترسی داشت. اگر از بلوک main() نیز خارج شویم ، x اول آزاد خواهد شد. برنامه را از اول دنبال کنید و خروجی را بررسی نمایید تا متوجه شوید که کدام x معتبر بوده است. در خط دهم از عملگر :: استفاده شده. به آن عملگر «جداسازی حوزه» می‌گویند. این عملگر را در فصل‌های بعدی بررسی می‌کنیم. در این‌جا فقط می‌گوییم با عملگر جداسازی حوزه می‌توان به یک شی که خارج از حوزهٔ فعلی است دسترسی پیدا کنیم. پس x:: یعنی متغیر x که در حوزهٔ بیرونی است.

می‌بینید که تعریف چند متغیر در یک برنامه با نام یکسان به شیوهٔ بالا ممکن و مجاز است. اما سعی کنید از این کار اجتناب کنید زیرا در غیر این صورت همیشه مجبورید به خاطر بسپارید که الان داخل کدام حوزه هستید و کدام متغیر مورد نظر شماست. این طوری راحت‌ترید؟ اختیار با شماست!

پرسش‌های گزینه‌ای

1 – از میان انواع زیر، کدام یک نوع صحیح حساب نمی‌شود؟

الف) double ب) int ج) char د) bool

2 – اگر m و n هر دو از نوع short باشند و m=6 و n=4 باشد، آنگاه حاصل m/n برابر است با:

الف) 1 ب) 1.5 ج) 2 د) nan

3 – اگر m از نوع double و n از نوع int باشد و m=6.0 و n=4 باشد، حاصل m/n چقدر است؟

الف) 1 ب) 1.5 ج) 2 د) nan

4 – متغیر m از نوع float و متغیر n از نوع short است. اگر بخواهیم حاصل m*n را در متغیری به نام k نگهداریم، آنگاه k باید از نوع ..... باشد.

الف) short ب) long

ج) float د) int

5 – در عبارت z += ++y اگر مقدار اولیهٔ y برابر با 5 و مقدار اولیهٔ z برابر با 7 باشد،حاصل z پس از اجرای آن دستور عبارت است از:

الف) 12 ب) 6 ج) 5 د) 13

6 – عدد 1.23e-1 معادل کدام یک از اعداد زیر است؟

الف) 12.3 ب) 12.3- ج) 0.123 د) 1.23-

7 – اگر a متغیری از نوع float با مقدار 5.63 باشد، آنگاه حاصل int(a) برابر است با:

الف) 5 ب) 6 ج) 0.63 د) 5.6

8 – برای این که حاصل m/n از نوع صحیح باشد باید:

الف) m از نوع صحیح باشد ب) n از نوع صحیح باشد

ج) یا m و یا n از نوع صحیح باشد د) هم m و هم n از نوع صحیح باشد

9 – برای تعریف انواع شمارشی از چه کلمهٔ کلیدی استفاده می‌شود؟

الف) include ب) enum ج) sqrt د) const

10 – اگر بخواهیم کاراکتر M را درون متغیر ch که از نوع کاراکتری است بگذاریم از چه دستوری استفاده می‌کنیم؟

الف) `ch = M;` ب) `ch = "M";`

ج) `ch = 'M';` د) `ch M`

11 – خطای گرد کردن مربوط به کدام نوع در C++ است؟

الف) نوع ممیز شناور ب) نوع صحیح

ج) نوع کاراکتری د) نوع شمارشی

12 – اگر یک متغیر از نوع int سرریز شود، چه مقداری در آن قرار می‌گیرد؟

الف) `-inf` ب) `inf` ج) `nan` د) عدد صحیح منفی

پرسش‌های تشریحی

1- قبل از اجرای دستورات زیر، مقدار m برابر 5 و مقدار n برابر 2 است. بعد از اجرای هر یک از دستورات زیر مقدار جدید m و n چیست؟

a. `m *= n++;`
b. `m += --n;`

2- مقدار هر یک از عبارات زیر را پس از مقداردهی برآورد کنید. ابتدا فرض کنید که m برابر 25 و n برابر 7 است.

a. `m - 8 - n`
b. `m = n = 3`
c. `m%n`
d. `m%n++`
e. `m%++n`
f. `++m - n—`

3- دو دستور زیر چه تفاوتی با هم دارند؟

```
char ch = 'A';
char ch = 65;
```

4- برای پیدا کردن کاراکتری که کد اسکی آن 100 است، چه کدی را می‌توانید اجرا کنید؟

5- معنای «ممیز شناور» چیست و چرا به این نام نامیده می‌شود؟

6- سرریزی عددی چیست؟

7- فرق سرریزی عدد صحیح با سرریزی عدد ممیز شناور چیست؟

8- خطای زمان اجرا چیست؟ مثال‌هایی برای دو نوع متفاوت از خطاهای زمان اجرا بنویسید.

9- خطای زمان کامپایل چیست؟ مثال‌هایی برای دو نوع متفاوت از خطاهای زمان کامپایل بنویسید.

10- کد زیر چه اشتباهی دارد؟

```
enum Semester {FALL, SPRING, SUMMER};
enum Season {SPRING, SUMMER, FALL, WINTER};
```

11- کد زیر چه اشتباهی دارد؟

```
enum Friends {"Jerry", "Henry", "W.D"};
```

تمرین‌های برنامه‌نویسی

1- چهار دستور متفاوت C++ بنویسید که 1 را از متغیر عدد صحیح n کم کند.

2- یک بلوک کد C++ بنویسید که مشابه جملهٔ زیر عمل کند بدون این که از عملگر ++ استفاده کنید.

```
n = 100 + m++;
```

3- یک بلوک کد C++ بنویسید که مشابه جملهٔ زیر عمل کند بدون این که از عملگر ++ استفاده کنید.

```
n = 100 + ++m;
```

4- یک دستور C++ تکی بنویسید که مجموع x و y را از z کم کند و سپس y را افزایش دهد.

5- یک دستور C++ تکی بنویسید که متغیر n را کاهش بدهد و سپس آن را به total اضافه کند.

7- برنامه‌ای را نوشته و اجرا کنید که موجب خطای پاریزی متغیری از نوع short شود.

8- برنامه‌ای مانند مثال 8 - 2 نوشته و اجرا کنید که تنها کد اسکی ده حرف صدادار بزرگ و کوچک را چاپ می‌کند.(برای بررسی خروجی، از ضمیمهٔ «الف» استفاده کنید)

9- برنامهٔ مثال 15-2 را طوری تغییر دهید که از نوع double به جای float استفاده کند. سپس مشاهده کنید که این برنامه چطور با ورودی‌هایی که خطای زمان اجرا را نشان می‌دهند، بهتر اجرا می‌شود.

10- برنامه‌ای بنویسید که اینچ را به سانتیمتر تبدیل کند. برای مثال اگر کاربر 16.9 را برای یک طول بر حسب اینچ وارد کند، خروجی 42.946 cm چاپ شود.(یک اینچ برابر 2.54 سانتیمتر است)

فصل سوم

«انتخاب»

همهٔ برنامه‌هایی که در دو فصل اول بیان شد، به شکل ترتیبی اجرا می‌شوند، یعنی دستورات برنامه به ترتیب از بالا به پایین و هر کدام دقیقا یک بار اجرا می‌شوند. در این فصل نشان داده می‌شود چگونه از دستورالعمل‌های انتخاب[1] جهت انعطاف‌پذیری بیشتر برنامه استفاده کنیم. همچنین در این فصل انواع صحیح که در C++ وجود دارد بیشتر بررسی می‌گردد.

3-1 دستور if

دستور **if** موجب می‌شود برنامه به شکل شرطی اجرا شود. نحو آن به گونهٔ زیر است:

```
If (condition) statement;
```

Condition که **شرط** نامیده می‌شود یک عبارت صحیح است (عبارتی که با یک مقدار صحیح برآورد می‌شود) و statement می‌تواند هر فرمان قابل اجرا باشد. Statement وقتی اجرا خواهد شد که *condition* مقدار غیر صفر داشته باشد.

---

1 – Selection

دقت کنید که شرط باید درون پرانتز قرار داده شود.

× مثال 1-3 آزمون بخش‌پذیری

این برنامه بررسی می‌کند که یک عدد صحیح مثبت بر عدد دیگر قابل تقسیم نباشد:

```
int main()
{  int n, d;
   cout << "Enter two positive integers: ";
   cin >> n >> d;
   if (n%d) cout << n << " is not divisible by "
                 << d << endl;
}
```

در اولین اجرا، اعداد 66 و 7 را وارد می‌کنیم:

```
Enter two positive integers: 66 7
66 is not divisible by 7
```

مقدار 66%7 برابر با 3 برآورد می‌گردد. چون این مقدار، یک عدد صحیح غیرصفر است، پس شرط به عنوان درست تفسیر می‌شود و در نتیجه دستور cout اجرا شده و پیغام عدم قابلیت تقسیم چاپ می‌شود.

برنامهٔ بالا را دوباره اجرا می‌نماییم و این دفعه اعداد 56 و 7 را وارد می‌کنیم:

```
Enter two positive integers: 56 7
```

مقدار 56%7 برابر با 0 برآورد می‌شود که این به معنی نادرست تفسیر می‌گردد، پس دستور cout نادیده گرفته شده و هیچ پیغامی روی صفحه چاپ نمی‌شود.

در C++ هر وقت یک عبارت صحیح به عنوان یک شرط استفاده شود، مقدار 0 به معنی «نادرست» و همهٔ مقادیر دیگر به معنی «درست» است.

برنامهٔ مثال 1-3 ناقص به نظر می‌آید زیرا اگر n بر d قابل تقسیم باشد، برنامه هیچ عکس‌العملی نشان نمی‌دهد. این نقص به کمک دستور if..else رفع می‌شود.

3-2 دستور if..else

دستور if..else موجب می‌شود بسته به این که شرط درست باشد یا خیر، یکی از دو دستورالعمل فرعی اجرا گردد. نحو این دستور به شکل زیر است:

```
if (condition) statement1;
else statement2;
```

condition همان شرط مساله است که یک عبارت صحیح می‌باشد و statement1 و statement2 فرمان‌های قابل اجرا هستند. اگر مقدار شرط، غیر صفر باشد، statement1 اجرا خواهد شد وگرنه statement2 اجرا می‌شود.

× مثال 2-3 یک آزمون دیگر قابلیت تقسیم

این برنامه مانند برنامهٔ مثال 1-3 است بجز این که دستور if با دستور if..else جایگزین شده است:

```
int main()
{  int n, d;
   cout << " Enter two positive integers: ";
   cin >> n >> d;
   if (n%d) cout << n << " is not divisible by "
                 << d << endl;
   else cout << n << " is divisible by " << d << endl;
}
```

حالا وقتی در این برنامه اعداد 56 و 7 را وارد کنیم، برنامه پاسخ می‌دهد که 56 بر 7 قابل تقسیم است:

```
Enter two positive integers: 56 7
56 is divisible by 7
```

چون حاصل 56%7 برابر با صفر است، پس این عبارت به عنوان نادرست تفسیر می‌گردد. در نتیجه دستور بعد از if نادیده گرفته شده و دستور بعد از else اجرا می‌شود. توجه کنید که if..else به تنهایی یک دستور است، گر چه به دو سمیکولن نیاز دارد.

3-4 عملگرهای مقایسه‌ای

در C++ شش عملگر مقایسه‌ای وجود دارد: < و > و <= و >= و == و =! . هر یک از این شش عملگر به شکل زیر به کار می‌روند:

```
x < y   // x کوچک‌تر از y است
x > y   // x بزرگ‌تر از y است
x <= y  // x کوچک‌تر یا مساوی y است
x >= y  // x بزرگ‌تر یا مساوی y است
x == y  // x مساوی با y است
x != y  // x مساوی با y نیست
```

این‌ها می‌توانند برای مقایسهٔ مقدار عبارات با هر نوع ترتیبی استفاده شوند. عبارت حاصل به عنوان یک شرط تفسیر می‌شود. مقدار این شرط صفر است اگر شرط نادرست باشد و غیر صفر است اگر شرط درست باشد. برای نمونه، عبارت 7×8<6×9 برابر با صفر ارزیابی می‌شود، به این معنی که این شرط نادرست است.

× مثال 3-3 کمینهٔ دو عدد صحیح

این برنامه مشخص می‌کند که از دو عدد صحیح ورودی، کدام یک کوچک‌تر است:

```
int main()
{  int m, n;
   cout << "Enter two integers: ";
   cin >> m >> n;
   if ( m < n ) cout << m << " is the minimum." << endl;
   else cout << n << " is the minimum." << endl;
}
```

```
Enter two integers: 77 55
55 is the minimum.
```

دقت کنید که در C++ عملگر جایگزینی با عملگر برابری فرق دارد. عملگر جایگزینی یک مساوی تکی " = " است ولی عملگر برابری، دو مساوی " = = " است. مثلا

دستور `x = 33;` مقدار 33 را در x قرار می‌دهد ولی دستور `x == 33;` بررسی می‌کند که آیا مقدار x با 33 برابر است یا خیر. درک این تفاوت اهمیت زیادی دارد.

```
x = 33;        // مقدار33 را به X تخصیص می‌دهد
x == 33;       // صفر ارزیابی می‌شود (به معنی نادرست) مگر این که مقدار x برابر با 33 باشد
```

× مثال 4-3 یک خطای برنامه‌نویسی متداول

این برنامه خطادار است:

```
int main()
{  int n;
   cout << "Enter an integer: ";
   cin >> n;
   if (n = 22) cout << "n = 22" << endl;  // LOGICAL ERROR!
   else cout << "n != 22" << endl;
}
```

```
Enter an integer: 77
n = 22
```

ظاهرا منطق برنامهٔ فوق به این گونه است که عددی از ورودی دریافت می‌شود و اگر این عدد با 22 برابر بود، پیغام برابری چاپ می‌شود و در غیر این صورت پیغام عدم برابری چاپ می‌گردد. ولی اجرای بالا نشان می‌دهد که برنامه درست کار نمی‌کند. عدد 77 وارد شده ولی پیغام `n = 22` در خروجی چاپ شده است! ایراد در خط پنجم برنامه است. عبارت `n = 22` مقدار 22 را در n قرار داده و مقدار قبلی آن که 77 است را تغییر می‌دهد. اما عبارت `n = 22` به عنوان شرط دستور `if` استفاده شده پس به عنوان یک عبارت صحیح با مقدار 22 برآورد می‌شود. لذا شرط `(n = 22)` به عنوان «درست» تفسیر می‌شود زیرا فقط مقدار 0 به معنای «نادرست» است. به همین دلیل دستور قبل از `else` اجرا می‌شود. خط پنجم باید این‌طور نوشته می‌شد:

```
if (n = = 22) cout << "n = 22" << endl;        // CORRECT
```

خطای نشان داده شده در این مثال، ***خطای منطقی***[1] نام دارد. این نوع خطا، بدترین نوع خطاهاست. خطاهای زمان کامپایل (مانند از قلم افتادن یک سمیکولن) به

1 – Logical Error

وسیلهٔ کامپایلر گرفته می‌شود. خطاهای زمان اجرا (مانند تقسیم بر صفر) نیز به وسیلهٔ سیستم عامل گرفته می‌شود اما خطای منطقی را نمی‌توان با این ابزارها کشف کرد.

× مثال 5-3 کمینهٔ سه عدد صحیح

این مثال شبیه مثال 3-3 است با این تفاوت که از سه عدد صحیح استفاده می‌کند:

```
int main()
{  int n1, n2, n3;
   cout << "Enter three integers: ";
   cin >> n1 >> n2 >> n3;
   int min=n1;              // now min <= n1
   if (n2 < min) min = n2;  // now min <= n1 and  n2
   if (n3 < min) min = n3;  // now min <= n1 ,n2 ,and n3
   cout << "Their minimum is " << min << endl;
}
```

```
Enter three integers: 77 33 55
Their minimum is 33
```

سه توضیح ذکر شده در برنامه، نحوهٔ پیشرفت کار را نشان می‌دهد: ابتدا min برابر n1 فرض می‌شود، لذا min کمینهٔ مجموعهٔ {n1} می‌شود. پس از اجرای اولین if، مقدار min برابر با n2 می‌شود اگر n2 از مقدار فعلی min کوچک‌تر باشد. پس min برابر کمینهٔ مجموعهٔ {n1,n2} می‌شود. آخرین دستور if، مقدار min را برابر با n3 قرار می‌دهد اگر n3 از مقدار فعلی min کوچک‌تر باشد. بنابراین در نهایت مقدار min برابر با کمینهٔ مجموعهٔ {n1, n2, n3} خواهد شد.

3-5 بلوک‌های دستورالعمل

یک ***بلوک دستورالعمل*** زنجیره‌ای از دستورالعمل‌هاست که درون براکت {} محصور شده، مانند این:

```
{  int temp=x;
   x = y;
   y = temp;
}
```

در برنامه‌های C++ یک بلوک دستورالعمل مانند یک دستورالعمل تکی است. یعنی هر جا که یک دستورالعمل تنها بتواند استفاده شود، یک بلوک دستورالعمل نیز می‌تواند استفاده شود. به مثال بعدی توجه کنید.

× مثال 6-3 یک بلوک دستورالعمل درون یک دستور if

این برنامه دو عدد صحیح را گرفته و به ترتیب بزرگ‌تری، آن‌ها را چاپ می‌کند:

```
int main()
{  int x, y;
   cout << "Enter two integers: ";
   cin >> x >> y;
   if (x > y)  { int temp = x;
                 x = y;
                 y = temp;
                }                          //swap x and y
   cout << x << " <= " << y << endl;
}
```

```
Enter two integers: 66 44
44 <= 66
```

سه دستور درون بلوک، مقادیر x و y را به ترتیب بزرگ‌تری مرتب می‌کنند بدین شکل که اگر آن‌ها خارج از ترتیب باشند، جای آن دو را عوض می‌کنند. برای این جابجایی به سه گام متوالی و یک محل ذخیره‌سازی موقتی احتیاج داریم که در این‌جا temp نامیده شده. برنامه یا باید هر سه دستورالعمل را اجرا کند و یا هیچ یک را نباید اجرا کند. وقتی این سه دستور را درون بلوک دستورالعمل قرار دهیم، منظور فوق برآورده می‌شود. توجه کنید که متغیر temp درون بلوک تعریف شده است. این سبب می‌شود که متغیر مذکور درون بلوک، یک **متغیر محلی**[1] باشد. یعنی این متغیر فقط وقتی ایجاد می‌شود که بلوک اجرا شود. اگر شرط نادرست باشد (یعنی x<=y باشد) متغیر temp هرگز موجود نخواهد شد. این مثال، روش مناسبی برای محلی کردن اشیا را نشان می‌دهد، طوری که اشیا وقتی ایجاد می‌شوند که به آن‌ها نیاز است. همچنین توجه کنید که یک برنامهٔ C++ خودش یک بلوک دستورالعمل است که توسط تابع

1 – Local Variable

اصلی main() ساخته شده است. یادآوری می‌کنیم که حوزهٔ متغیر، قسمتی از یک برنامه است که متغیر می‌تواند در آن استفاده شود(بخش 5-1). این حوزه، از نقطه‌ای که متغیر اعلان می‌شود شروع شده و تا پایان همان بلوک ادامه می‌یابد. پس یک بلوک می‌تواند به عنوان محدودهٔ حوزهٔ متغیر استفاده شود. یکی از نتایج مهم این کار آن است که می‌توانیم از متغیرهای متفاوتی با یک نام در قسمت‌های مختلف برنامه استفاده کنیم.

× مثال 7-3 استفاده از بلوک‌ها به عنوان محدودهٔ حوزه

در این برنامه سه متغیر مختلف با نام n استفاده شده است:

```
int main()
{  int n=44;
   cout << "n = " << n << endl;
   {  int n;                          // scope extends over 4 lines
      cout << "Enter an integer: ";
      cin >> n;
      cout << "n = " << n << endl;
   }
   {  cout << " n = " << n << endl; // n that was declared first
   }
   {  int n;                          // scope extends over 2 lines
      cout << "n = " << n << endl;
   }
   cout << "n = " << n << endl;     // n that was declared first
}
```

```
n = 44
Enter an integer: 77
n = 77
n = 44
n = 4251897
n = 44
```

برنامهٔ بالا سه بلوک داخلی دارد. اولین بلوک یک n جدید اعلان می‌کند که فقط درون همان بلوک، معتبر و موجود است. این n متغیر n اصلی را پنهان می‌کند. بنابراین وقتی مقدار 77 در این بلوک از ورودی دریافت می‌شود، این مقدار درون n محلی قرار می‌گیرد و مقدار n اصلی بدون تغییر می‌ماند. در دومین بلوک n جدیدی تعریف

نمی‌شود، لذا حوزهٔ n اصلی این بلوک را نیز شامل می‌شود. پس در سومین دستور خروجی، مقدار n اصلی یعنی 44 چاپ می‌شود. بلوک سوم برنامه نیز مانند بلوک اول یک n جدید تعریف می‌کند که n اصلی را پنهان می‌نماید، اما این n جدید مقداردهی نمی‌شود. بنابراین در چهارمین خروجی، یک مقدار زباله چاپ می‌شود. در خط انتهایی برنامه، تمام بلوک‌های محلی به پایان می‌رسند. به همین خاطر وقتی در این خط دستور چاپ برای n صادر می‌شود، مقدار n اصلی یعنی 44 چاپ می‌شود.

3-6 شرط‌های مرکب

شرط‌هایی مانند n%d و x>=y می‌توانند به صورت یک شرط مرکب با هم ترکیب شوند. این کار با استفاده ازعملگرهای منطقی && (and) و || (or) و ! (not) صورت می‌پذیرد. این عملگرها به شکل زیر تعریف می‌شوند:

p && q درست است اگر و تنها اگر هم p و هم q هر دو درست باشند

p || q نادرست است اگر و تنها اگر هم p و هم q هر دو نادرست باشند

p! درست است اگر و تنها اگر p نادرست باشد

برای مثال (n%d || x>=y) نادرست است اگر و تنها اگر n%d برابر صفر و x کوچک‌تر از y باشد.

سه عملگر منطقی بالا معمولا با استفاده از ***جداول درستی*** به گونهٔ زیر بیان می‌شوند:

p	!p
T	F
F	T

p	q	p||q
T	T	T
T	F	T
F	T	T
F	F	F

p	q	P&&q
T	T	T
T	F	F
F	T	F
F	F	F

طبق جدول‌های فوق اگر p درست و q نادرست باشد، عبارت p&&q نادرست و عبارت p||q درست است.

مثال بعدی همان مسألهٔ مثال 5-3 را حل می‌کند ولی این کار را با استفاده از شرط‌های مرکب انجام می‌دهد.

× مثال 8-3 استفاده از شرط‌های مرکب

برنامهٔ زیر مانند برنامهٔ مثال 5-3 است. این نسخه برای یافتن کمینهٔ سه عدد از شرط‌های مرکب استفاده کرده است:

```
int main()
{  int n1, n2, n3;
   cout << "Enter three integers: ";
   cin >> n1 >> n2 >> n3;
   if (n1<=n2 && n1<=n3) cout << "Their minimum is "
       << n1 <<endl;
   if (n2<=n1 && n2<=n3) cout << "Their minimum is "
       << n2 <<endl;
   if (n3<=n1 && n3<=n2) cout << "Their minimum is "
       << n3 <<endl;
}
```

```
Enter three integers: 77 33 55
Their minimum is 33
```

برنامه‌ای که در این مثال آمد هیچ مزیتی بر مثال 5-3 ندارد و فقط نحوهٔ استفاده از شرط‌های مرکب را بیان می‌کند. در مثال زیر هم از شرط مرکب استفاده شده است.

× مثال 9-3 ورودی کاربر پسند

این برنامه به کاربر امکان می‌دهد که برای پاسخ مثبت "y" یا "Y" را وارد کند:

```
int main()
{  char ans;
   cout << "Are you enrolled (y/n): ";
   cin >> ans;
   if (ans= ='Y' || ans= ='y') cout << "You are enrolled.\n";
   else cout << "You are not enrolled.\n";
}
```

```
Are you enrolled (y/n): N
```

```
You are not enrolled
```

برنامهٔ بالا از کاربر پاسخی می‌خواهد و y و n را به عنوان جواب‌های ممکن پیشنهاد می‌دهد. اما هر کاراکتر دیگری را هم می‌پذیرد و اگر آن کاراکتر 'y' یا 'Y' نباشد، فرض می‌کند که پاسخ کاربر "no" است.

3-7 ارزیابی میانبری

عملگرهای && و || به دو عملوند نیاز دارند. یعنی به دو مقدار نیاز دارند تا مقایسه را روی آن دو انجام دهند. شرط‌های مرکب که از && و || استفاده می‌کنند عملوند دوم را بررسی نمی‌کنند مگر این که لازم باشد. جداول درستی نشان می‌دهد که p&&q نادرست است اگر p نادرست باشد. در این حالت دیگر نیازی نیست که q بررسی شود. همچنین p||q درست است اگر p درست باشد و در این حالت هم نیازی نیست که q بررسی شود. در هر دو حالت گفته شده، با ارزیابی عملوند اول به سرعت نتیجه معلوم می‌شود. این کار ***ارزیابی میانبری*** نامیده می‌شود.

× مثال 10-3 ارزیابی میانبری

برنامهٔ زیر بخش‌پذیری اعداد صحیح را بررسی می‌کند:

```
int main()
{  int n, d;
   cout << "Enter two positive integers: ";
   cin >> n >> d;
   if (d != 0 && n%d == 0) cout << d << " divides " << n
                                << endl;
   else cout << d << " does not divide " << n << endl;
}
```

در اجرای زیر، d مثبت و n%d صفر است. بنابراین شرط مرکب درست است:

```
Enter two integers: 300 5
5 divides 300
```

در اجرای بعدی، d مثبت است اما n%d صفر نیست. بنابراین شرط مرکب نادرست است:

```
Enter two integers: 300 7
7 does not divide 300
```

در آخرین اجرا، d صفر است. پس به سرعت برآورد می‌شود که شرط مرکب نادرست است بدون این که عبارت دوم یعنی n%d == 0 ارزیابی شود:

```
Enter two integers: 300 0
0 does not divide 300
```

ارزیابی میانبری در مثال بالا از خرابی برنامه جلوگیری می‌کند زیرا وقتی d صفر است، رایانه نمی‌تواند عبارت n%d را محاسبه کند.

3-8 عبارات منطقی

یک ***عبارت منطقی*** شرطی است که یا درست است یا نادرست. در مثال قبلی عبارات d>0 و n%d == 0 و (d>0 && n%d==0) عبارات منطقی هستند. قبلا دیدیم که عبارات منطقی با مقادیر صحیح ارزیابی می‌شوند. مقدار صفر به معنای نادرست و هر مقدار غیر صفر به معنای درست است. به عبارات منطقی «عبارات بولی» هم می‌گویند.

چون همهٔ مقادیر صحیح ناصفر به معنای درست تفسیر می‌شوند، عبارات منطقی اغلب تغییر قیافه می‌دهند. برای مثال دستور

```
if (n) cout << "n is not zero";
```

وقتی n غیر صفر است عبارت n is not zero را چاپ می‌کند زیرا عبارت منطقی (n) وقتی مقدار n غیر صفر است به عنوان درست تفسیر می‌گردد. کد زیر را نگاه کنید:

```
if (n%d) cout << "n is not a multiple of d";
```

دستور خروجی فقط وقتی که n%d ناصفر است اجرا می‌گردد و n%d وقتی ناصفر است که n بر d بخش‌پذیر نباشد. گاهی ممکن است فراموش کنیم که عبارات منطقی مقادیر صحیح دارند و این فراموشی باعث ایجاد نتایج غیر منتظره و نامتعارف شود.

× مثال 3-11 یک خطای منطقی دیگر

این برنامه خطادار است:

```
int main()
{  int n1, n2, n3;
   cout << "Enter three integers: ";
   cin >> n1 >> n2 >> n3;
   if (n1 >= n2 >= n3) cout << "max = " << n1;  // LOGICAL ERROR!
}
```

```
Enter three integers: 0 0 1
max = 0
```

منشأ خطا در برنامهٔ بالا این اصل است که عبارات منطقی مقدارهای عددی دارند. چون عبارت (n1>=n2>=n3) از چپ به راست ارزیابی می‌شود، به ازای ورودی‌های فوق اولین بخش ارزیابی یعنی n1>=n2 درست است چون 0>=0 اما «**درست**» به شکل عدد 1 در حافظه نگهداری می‌شود. سپس این مقدار با مقدار n3 که 1 می‌باشد مقایسه می‌شود. یعنی عبارت 1>=1 ارزیابی می‌شود که این هم درست است. نتیجه این است که کل عبارت به عنوان درست تفسیر می‌شود گرچه در حقیقت این طور نیست! (0 بیشینهٔ 0 و 1 نیست)

ایراد کار این‌جاست که خط اشتباه به طور نحوی صحیح است. بنابراین نه کامپایلر می‌تواند خطا بگیرد و نه سیستم‌عامل. این نوع دیگری از خطای منطقی است که با آن‌چه در مثال 3-4 مطرح شد قابل مقایسه است. نتیجهٔ این مثال آن است که: «همیشه به خاطر داشته باشید عبارات منطقی مقدار عددی دارند، بنابراین شرط‌های مرکب می‌توانند گول‌زننده باشند».

3-9 دستورهای انتخاب تودرتو

دستورهای انتخاب می‌توانند مانند دستورالعمل‌های مرکب به کار روند. به این صورت که یک دستور انتخاب می‌تواند درون دستور انتخاب دیگر استفاده شود. به این روش، ***جملات تودرتو*** می‌گویند.

× مثال 12-3 دستورهای انتخاب تودرتو

این برنامه همان اثر مثال 10-3 را دارد:

```
int main()
{  int n, d;
   cout << "Enter two positive integers: ";
   cin >> n >> d;
   if (d != 0)
      if (n%d = = 0) cout << d << " divides " << n << endl;
      else cout << d << " does not divide " << n << endl;
   else cout << d << " does not divide " << n << endl;
}
```

در برنامهٔ بالا، دستور if..else دوم درون دستور if..else اول قرار گرفته است. پس if..else دوم وقتی اجرا می‌شود که d صفر نباشد. توجه کنید که در این‌جا مجبوریم دو بار از عبارت does not divide استفاده کنیم. اولی در اولین دستور if..else قرار گرفته و زمانی اجرا می‌شود که d صفر نباشد و n%d صفر گردد. دومی هم وقتی اجرا می‌شود که d صفر باشد.

وقتی دستور **if..else** به شکل تو در تو به کار می‌رود، کامپایلر از قانون زیر جهت تجزیه این دستورالعمل مرکب استفاده می‌کند:

« هر **else** با آخرین **if** تنها جفت می‌شود.»

با به‌کارگیری این قانون، کامپایلر به راحتی می‌تواند کد پیچیدهٔ زیر را رمز گشایی کند:

```
if (a > 0) if (b > 0) ++a; else if (c > 0)        //BAD CODING STYLE
if (a > 4) ++b; else  if (b < 4) ++c; else -a;    //BAD CODING STYLE
else if (c < 4) --b; else --c; else a = 0;        //BAD CODING STYLE
```

برای این که کد بالا را خواناتر و قابل فهم کنیم، می‌توانیم آن را به شکل زیر بنویسیم:

```
if (a > 0)
   if (b > 0) ++a;
   else
      if (c > 0)
         if (a < 4) ++b;
```

```
         else
            if (b < 4) ++c;
            else –a;
      else
         if (c < 4) –b;
         else –c;
else a = 0;
```

یا به این شکل :

```
if (a > 0)
   if (b > 0) ++a;
   else if (c > 0)
      if (a < 4) ++b;
      else if (b < 4) ++c;
      else –a;
   else if (c < 4) –b;
   else –c;
else a = 0;
```

در شیوهٔ دوم عبارات `else if` زیرِ هم و در یک راستا نوشته می‌شوند.

× مثال 13–3 استفاده از دستورهای انتخاب تودرتو

این برنامه همان اثر مثال‌های 5–3 و 8–3 را دارد. در این نسخه برای یافتن کمینهٔ سه عدد صحیح از دستورهای `if..else` تودرتو استفاده می‌شود:

```
int main()
{  int n1, n2, n3;
   cout << "Enter three integers: ";
   cin >> n1 >> n2 >> n3;
   if (n1 < n2)
      if (n1 < n3) cout << "Their minimum is " << n1 << endl;
      else cout << "Their minimum is " << n3 << endl;
   else      // n1 >= n2
      if (n2 < n3) cout << "Their minimum is " << n2 << endl;
      else cout << "Their minimum is " << n3 << endl;
}
```

```
Enter two integers: 77 33 55
Their minimum is 33
```

در اجرای بالا، اولین شرط (n1<n2) نادرست است و سومین شرط (n2<n3) درست است. بنابراین گزارش می‌شود که n2 کمینه است.

این برنامه از برنامهٔ مثال 8-3 موثرتر است زیرا در هر اجرای آن فقط دو شرط سادهٔ تودرتو به جای سه شرط مرکب ارزیابی می‌شود ولی به نظر می‌رسد این برنامه ارزش کم‌تری نسبت به برنامهٔ مثال 8-3 داشته باشد زیرا منطق این برنامه پیچیده‌تر است. در مقایسه بین کارایی و سادگی، معمولا بهتر است سادگی انتخاب گردد.

× مثال 14-3 کمی پیچیده‌تر: یک بازی حدسی

برنامهٔ زیر عددی را که کاربر بین 1 تا 8 در ذهن دارد، پیدا می‌کند:

```
int main()
{  cout << "Pick a number from 1 to 8." << endl;
   char answer;
   cout << "Is it less than 5? (y|n): "; cin >> answer;
   if (answer = = 'y')            // 1 <=n <= 4
   {  cout << "Is it less than 3? (y|n): "; cin >> answer;
      if (answer = = 'y')         // 1 <= n <= 2
      {  cout << "Is it less than 2? (y|n): "; cin >> answer;
         if (answer = = 'y') cout << "Your number is 1."
             << endl;
         else cout << "Your number is 2." << endl;
      }
      else                        // 3 <= n <= 4
      {  cout << "Is it less than 4? (y|n): "; cin >> answer;
         if (answer = = 'y') cout << "Your number is 3."
             << endl;
         else cout << "Your number is 4." << endl;
      }
   }
   else                           // 5 <= n <= 8
   {  cout << "Is it less than 7? (y|n): "; cin >> answer;
      if (answer = = 'y')         // 5 <= n <= 6
```

```
      {  cout >> "Is it less than 6? (y|n): "; cin >> answer;
         if (answer = = 'y') cout << "Your number is 5."
             << endl;
         else cout << "Your number is 6." << endl;
      }
      else                          // 7 <= n <= 8
      {  cout << "Is it less than 8? (y n): "; cin >> answer;
         if (answer = = 'y') cout << "your number is 7."
             << endl;
         else cout << "Your number is 8." << endl;
      }
   }
}
```

برنامهٔ بالا با تجزیهٔ مسأله قادر است تنها با سه پرسش، هر یک از هشت عدد را پیدا کند. در اجرای زیر، کاربر عدد 6 را در نظر داشته است:

```
Pick a number from 1 to 8.
Is it less than 5? (y|n) : n
Is it less than 7? (y|n) : y
Is it less than 6? (y|n) : n
Your number is 6.
```

سعی کنید منطق برنامهٔ بالا را کشف کنید. به الگوریتم استفاده شده در مثال 3-14 الگوریتم ***جستجوی دودویی***[1] می‌گویند. این الگوریتم روی مجموعه‌های مرتب به کار می‌رود و به سرعت مشخص می‌کند آیا یک دادهٔ مفروض در این مجموعه هست یا خیر. در فصل‌های بعدی روش‌های دیگری از جستجو را خواهیم دید.

3-10 ساختار else if

دستور if..else تودرتو، اغلب برای بررسی مجموعه‌ای از حالت‌های متناوب یا موازی به کار می‌رود. در این حالات فقط عبارت else شامل دستور if بعدی خواهد بود. این قبیل کدها را معمولا با ساختار **else if** می‌سازند.

---

1 – Binary search

× مثال 15-3 استفاده از ساختار else if برای بررسی حالت‌های موازی

برنامهٔ زیر زبان کاربر را سوال می‌کند و سپس یک پیغام به همان زبان در خروجی چاپ می‌نماید:

```
int main()
{  char language;
   cout << "Engl., Fren., Ger., Ital., or Rus.? (e|f|g|i|r): ";
   cin >> language;
   if (language = = 'e') cout << "Welcome to ProjectC++.";
   else if (language = = 'f') cout << "Bon jour, ProjectC++.";
   else if (language = = 'g') cout << "Guten tag, ProjectC++.";
   else if (language = = 'i') cout << "Bon giorno, ProjectC++.";
   else if (language = = 'r') cout << "Dobre utre, ProjectC++.";
   else cout << "Sorry; we don't speak your language.";
}
```

```
Engl., Fren., Ger., Ital., or Rus.? (e|f|g|i|r): i
Bon giorno, ProjectC++.
```

این برنامه در حقیقت از دستور if..else تودرتو استفاده کرده. کد بالا را می‌توانستیم به ترتیب زیر بنویسیم:

```
if (language = = 'e') cout << "Welcome to ProjectC++.";
else
   if (language = = 'f') cout << "Bon jour, ProjectC++.";
   else
      if (language = = 'g') cout << "Guten tag, ProjectC++.";
      else
         if (language = = 'i') cout << "Bon giorno, ProjectC++.";
         else
            if (language = = 'r') cout << "Dobre utre, ProjectC++.";
            else cout << "Sorry; we don't speak your language.";
```

اما قالب قبلی به علت این که درک منطق برنامه را آسان‌تر می‌کند، بیشتر استفاده می‌شود. همچنین به تورفتگی کمتری احتیاج دارد.

× مثال 3-16 استفاده از ساختار else if برای مشخص کردن محدودهٔ نمره

برنامهٔ زیر یک نمرهٔ امتحان را به درجهٔ حرفی معادل تبدیل می‌کند:

```
int main()
{  int score;
   cout << "Enter your test score: ";  cin >> score;
   if (score > 100) cout << "Error: that score is out of range.";
   else if (score >= 90) cout << "Your grade is an A." << endl;
   else if (score >= 80) cout << "Your grade is a B." << endl;
   else if (score >= 70) cout << "Your grade is a C." << endl;
   else if (score >= 60) cout << "Your grade is a D." << endl;
   else if (score >= 0) cout << "Your grade is an F." << endl;
   else cout << "Error: that score is out of range.";
}
```

```
Enter your test score: 83
Your grade is a B.
```

مقدار متغیر score به شکل آبشاری با دستورهای انتخاب به طور متوالی بررسی می‌شود تا این که یکی از شرطها درست شود و یا به آخرین else برسیم.

3-11 دستورالعمل switch

دستور switch می‌تواند به جای ساختار else if برای بررسی مجموعه‌ای از حالت‌های متناوب و موازی به کار رود. نحو دستور switch به شکل زیر است:

```
switch (expression)
{  case constant1: statementlist1;
   case constant2: statementlist2;
   case constant3: statementlist3;
         :
         :
   case constantN: statementlistN;
   default: statementlist0;
}
```

این دستور ابتدا *expression* را برآورد می‌کند و سپس میان ثابت‌های **case** به دنبال مقدار آن می‌گردد. اگر مقدار مربوطه از میان ثابت‌های فهرست‌شده یافت شد، دستور *statementlist* مقابل آن **case** اجرا می‌شود. اگر مقدار مورد نظر میان **case**ها یافت نشد و عبارت **default** وجود داشت، دستور *statementlist* مقابل آن اجرا می‌شود. عبارت **default** یک عبارت اختیاری است. یعنی می‌توانیم در دستور **switch** آن را قید نکنیم. *expression* باید به شکل یک نوع صحیح ارزیابی شود و *constant*ها باید ثابت‌های صحیح باشند.

× مثال 17-3 نسخهٔ تغییریافته‌ای از مثال 16-3

این برنامه همان اثر برنامهٔ مثال 16-3 را دارد. در این نسخه از دستور switch استفاده شده:

```
int main()
{  int score;
   cout << "Enter your test score: ";  cin >> score;
   switch (score/10)
   {  case 10:
      case  9: cout << "Your grade is an A." << endl;  break;
      case  8: cout << "Your grade is a B." << endl;   break;
      case  7: cout << "Your grade is a C." << endl;   break;
      case  6: cout << "Your grade is a D." << endl;   break;
      case  5:
      case  4:
      case  3:
      case  2:
      case  1:
      case  0: cout << "Your grade is an F." << endl;  break;
      default: cout << "Error: score is out of range.\n";
   }
   cout << "Goodbye." << endl;
}
```

```
Enter your test score: 83
Your grade is a B.
```

```
Goodbye.
```

در برنامهٔ بالا ابتدا score بر 10 تقسیم می‌شود تا محدودهٔ اعداد بین صفر تا 10 محدود شود. بنابراین در اجرای آزمایشی، نمرهٔ 83 به 8 تبدیل می‌شود، اجرای برنامه به case 8 انشعاب می‌کند و خروجی مربوطه چاپ می‌گردد. سپس دستور break موجب می‌شود که اجرای برنامه از دستور switch خارج شده و به اولین دستور بعد از بلوک switch انشعاب کند. در آن‌جا عبارت "Goodbye." چاپ می‌شود.

لازم است در انتهای هر case دستور break قرار بگیرد. بدون این دستور، اجرای برنامه پس از این که case مربوطه را اجرا کرد از دستور switch خارج نمی‌شود، بلکه همهٔ caseهای زیرین را هم خط به خط می‌پیماید و دستورات مقابل آن‌ها را اجرا می‌کند. به این اتفاق، ***تلهٔ سقوط***[1] می‌گویند.

× مثال 18-3 تلهٔ سقوط در دستور switch

قصد بر این است که این برنامه مثل برنامهٔ 17-3 رفتار کند ولی بدون دستورهای break این برنامه دچار تلهٔ سقوط می‌شود:

```
int main()
{  int score;
   cout << "Enter your test score: ";  cin >> score;
   switch (score/10)
   {  case 10:
      case  9: cout << "Your grade is an A." << endl;  // LOGICAL ERROR
      case  8: cout << "Your grade is a B." << endl;   // LOGICAL ERROR
      case  7: cout << "Your grade is a C." << endl;   // LOGICAL ERROR
      case  6: cout << "Your grade is a D." << endl;   // LOGICAL ERROR
      case  5:
      case  4:
      case  3:
      case  2:
      case  1:
      case  0: cout << "Your grade is an F." << endl;  // LOGICAL ERROR
      default: cout << "Error: score is out of range.\n";
   }
```

1 – Fall-throw error

```
   cout << "Goodbye." << endl;
}
```

```
Enter your test score: 83
Your grade is a B.
Your grade is a C.
Your grade is a D.
Your grade is an F.
Error: score is out of range.
Goodbye.
```

در اجرای فوق پس از این که به case 8 انشعاب شد و عبارت مقابل آن چاپ شد، چون دستور break وجود ندارد، اجرای برنامه به خط بعدی یعنی case 7 می‌رود و عبارت "Your grade is a C." را نیز چاپ می‌کند و به همین ترتیب یکی یکی همهٔ عبارت‌های case را اجرا می‌نماید و سرانجام عبارت default را هم اجرا نموده و آنگاه از دستور switch خارج می‌شود.

3-12 عملگر عبارت شرطی

یکی از مزیت‌های C++ اختصار در کدنویسی است. ***عملگر عبارت شرطی*** یکی از امکاناتی است که جهت اختصار در کدنویسی تدارک دیده شده است. این عملگر را می‌توانیم به جای دستور if..else به کار ببریم. این عملگر از نشانه‌های **?** و **:** به شکل زیر استفاده می‌کند:

```
condition ? expression1 : expression2;
```

در این عملگر ابتدا شرط *condition* بررسی می‌شود. اگر این شرط درست بود، حاصل کل عبارت برابر با expression1 می‌شود و اگر شرط نادرست بود، حاصل کل عبارت برابر با expression2 می‌شود. مثلا در دستور انتساب زیر:

```
min = ( x<y ? x : y );
```

اگر x<y باشد مقدار x را درون min قرار می‌دهد و اگر x<y نباشد مقدار y را درون min قرار می‌دهد. یعنی به همین سادگی و اختصار، مقدار کمینهٔ x و y درون متغیر min قرار می‌گیرد. عملگر عبارت شرطی یک عملگر سه‌گانه است. یعنی سه عملوند را

برای تهیهٔ یک مقدار به کار می‌گیرد. بهتر است عملگر عبارت شرطی فقط در مواقع ضروری استفاده شود؛ وقتی که شرط و هر دو دستور خیلی ساده هستند.

× مثال 3-19 نسخهٔ جدیدی از برنامهٔ یافتن مقدار کمینه

این برنامه اثری شبیه برنامهٔ مثال 3-3 دارد:

```
int main()
{  int m, n;
   cout << "Enter two integers: ";
   cin >> m >> n;
   cout << ( m<n ? m : n ) << " is the minimum." << endl;
}
```

عبارت شرطی **(m<n ? m : n)** برابر با m می‌شود اگر m<n باشد و در غیر این صورت برابر با n می‌شود.

3-13 کلمات کلیدی[1]

اکنون با کلماتی مثل if و case و float آشنا شدیم. دانستیم که این کلمات برای C++ معانی خاصی دارند. از این کلمات نمی‌توان به عنوان نام یک متغیر یا هر منظور دیگری استفاده کرد و فقط باید برای انجام همان کار خاص استفاده شوند. مثلا کلمهٔ float فقط باید برای معرفی یک نوع اعشاری به کار رود. یک ***کلمهٔ کلیدی*** در یک زبان برنامه‌نویسی کلمه‌ای است که از قبل تعریف شده و برای هدف مشخصی منظور شده است. C++ استاندارد اکنون شامل 74 کلمهٔ کلیدی است:

and	**and_eq**	**asm**
auto	**bitand**	**Bitor**
bool	**break**	**case**
catch	**char**	**class**
compl	**const**	**const_cast**
continue	**default**	**delete**
dodouble	**dynamic_cast**	**else**
enum	**explicit**	**export**
extern	**dfalse**	**float**
for	**friend**	**goto**

1 – Keywords

if	**inline**	**int**
long	**mutable**	**namespace**
new	**not**	**not_eq**
operator	**or**	**or_eq**
privat	**eprotected**	**public**
register	**reinterpret_cast**	**return**
short	**signed**	**sizeof**
static	**static_cast**	**struct**
swich	**template**	**this**
throw	**true**	**try**
typedef	**typoid**	**typename**
using	**union**	**unsigned**
virtual	**void**	**volatile**
wchar_t	**while**	**xor**
xor_eq		

کلمات کلیدی مانند if و else تقریبا در هر زبان برنامه‌نویسی پیدا می‌شوند. دیگر کلمات کلیدی همچون dynamic_cast منحصر به C++ هستند. 74 کلمهٔ کلیدی در C++ هست که همهٔ 32 کلمهٔ کلیدی زبان C را نیز شامل می‌شود.

دو نوع کلمهٔ کلیدی وجود دارد: ***کلمه‌های رزرو شده*** و ***شناسه‌های استاندارد***. یک کلمهٔ رزرو شده کلمه‌ای است که یک دستور خاص از آن زبان را نشان می‌دهد. کلمهٔ کلیدی if و else کلمات رزرو شده هستند. یک شناسهٔ استاندارد کلمه‌ای است که یک نوع دادهٔ استاندارد از زبان را مشخص می‌کند. کلمات کلیدی bool و int شناسه‌های استاندارد هستند زیرا هر یک از آن‌ها یک نوع دادهٔ خاص را در زبان C++ مشخص می‌کنند. برای اطلاعات بیشتر در مورد کلمات کلیدی C++ به مراجع این زبان یا راهنمای کامپایلرتان مراجعه کنید.

پرسش‌های گزینه‌ای

1 – به ازای کد `if (k = 0) i=k;` کدام جمله صحیح است؟

الف) اگر k مساوی با صفر باشد، آنگاه مقدار k در i کپی می‌شود.

ب) اگر k مساوی با غیر صفر باشد، آنگاه مقدار k در i کپی می‌شود.

ج) کامپایلر خطا می‌گیرد زیرا عملگر برابری == است نه = .

د) به ازای همهٔ مقادیر k مقدار k در i کپی می‌شود.

2 – اگر متغیر b از نوع بولین باشد، کد `if(b) b;` چه کاری انجام می‌دهد؟

الف) اگر b برابر با true باشد، آنگاه b را false می‌کند.

ب) اگر b برابر با false باشد، آنگاه b را true می‌کند.

ج) اگر b برابر با true باشد، آنگاه b را false می‌کند وگرنه b را true می‌کند.

د) اگر b برابر با false باشد، آنگاه b را true می‌کند وگرنه b را false می‌کند.

3 – کد `if (i==0) if (j==0) k=0;` معادل کدام یک از کدهای زیر است؟

الف)
```
if ((i==0) || (j==0)) k=0;
```

ب)
```
if ((i==0) && (j==0)) k=0;
```

ج)
```
if (i==0) k=0;
```

د)
```
if (j==0) k=0;
```

4 – در ارزیابی عبارت‌های شرطی:

الف) صفر به معنای درست و هر مقدار غیر صفر به معنای نادرست است.

ب) صفر به معنای نادرست و هر مقدار غیر صفر به معنای درست است.

ج) یک به معنای درست و هر مقدار غیر یک به معنای نادرست است.

د) یک به معنای نادرست و هر مقدار غیر یک به معنای درست است.

5 – اگر m یک متغیر بولین باشد، در کد `if(m) i++ else i--;` چه روی می‌دهد؟

الف) اگر m برابر با true باشد، به i یک واحد افزوده می‌شود وگرنه از i یک واحد کاسته می‌شود

ب) اگر m برابر با false باشد، به i یک واحد افزوده می‌شود وگرنه از i یک واحد کاسته می‌شود

ج) اگر m برابر با true باشد، به i دو واحد افزوده می‌شود وگرنه از i دو واحد کاسته می‌شود

د) اگر m برابر با False باشد، به i دو واحد افزوده می‌شود وگرنه از i دو واحد کاسته می‌شود

6 – در کد `if (i<0) i++; j++;` چه رخ می‌دهد؟

الف) اگر i از صفر کوچک‌تر باشد، به j یک واحد افزوده می‌شود

ب) اگر i بزرگ‌تر یا مساوی صفر باشد، به j یک واحد افزوده می‌شود

ج) اگر به i یک واحد افزوده شود، به j هم یک واحد افزوده می‌شود.

د) مقدار i ربطی به j ندارد و در هر حال به j یک واحد افزوده می‌شود.

7 – در کد `if (i<j) {i++; j--;}` چه اتفاقی می‌افتد؟

الف) اگر i از j کوچک‌تر باشد، از j یک واحد کاسته می‌شود

ب) اگر i از j بزرگ‌تر باشد، از j یک واحد کاسته می‌شود

ج) در هر حال به i یک واحد اضافه می‌شود و ربطی به j ندارد

د) در هر حال از j یک واحد کاسته می‌شود و ربطی به i ندارد

8 – خروجی کد مقابل چیست؟

```
int n=55;
{  int n=77;
   cout << n << endl;
}
cout << n;
```

الف) روی اولین سطر 55 چاپ می‌شود و روی دومین سطر 77 چاپ می‌شود

ب) روی اولین سطر 77 چاپ می‌شود و روی دومین سطر 55 چاپ می‌شود

ج) روی هر دو سطر مقدار 55 چاپ می‌شود

د) روی هر دو سطر مقدار 77 چاپ می‌شود

9 – حاصل اجرای کد `if ((d>1) || (d/m)) d++;` به ازای d=2 و m=0 چیست؟

الف) خطای تقسیم بر صفر رخ می‌دهد و برنامه متوقف می‌شود

ب) شرط دستور if نادرست است پس دستور d++ نادیده گرفته می‌شود

ج) به d یک واحد اضافه می‌شود

د) خطای تقسیم بر صفر رخ می‌دهد پس دستور d++ نادیده گرفته می‌شود

10 – اگر a=0 و b=1 و c=2 باشد، مقدار c پس از اجرای کد زیر، چیست؟

```
if (a==1)
if (b==1) c++;
else c--;
```

الف) c = 3 ب) c = 1 ج) c = 2 د) c = 4

11 – اگر i = 5 باشد، مقدار i پس از اجرای کد زیر، چیست؟

```
switch (i)
{  case 5: i++;
   case 0: i--;
   default: i--;
}
```

الف) i = 6 ب) i = 5 ج) i = 4 د) i = 3

12 – تله سقوط وقتی رخ می‌دهد که :

الف) به جای عملگر برابری (==) از عملگر جایگزینی (=) استفاده کنیم

ب) در دستور if پرانتزهای شرط را فراموش کنیم

ج) تلاش کنیم عددی را بر صفر تقسیم کنیم

د) در دستور switch دستورهای break را فراموش کنیم

13 – کد if (x>y) p=x; else p=y; معادل کدام یک از کدهای زیر است؟

الف) p=(x>y ? y : x); ب) p=(x>y ? x : y);

ج) (x>y ? p=y : p=x); د) (x>y ? p=x : p=y);

14 – معنی جملهٔ مقابل چیست؟ «عملگر عبارت شرطی یک عملگر سه گانه است»

الف) یعنی عملگر عبارت شرطی سه شکل متفاوت دارد

ب) یعنی عملگر عبارت شرطی سه کاربرد متفاوت دارد

ج) یعنی عملگر عبارت شرطی ترکیبی از سه شرط است

د) یعنی عملگر عبارت شرطی سه عملوند را برای تهیهٔ یک مقدار به کار می‌گیرد

15 – کدام دستور زیر، یک دستور انتخاب نیست؟

الف) دستور break ب) دستور if

ج) دستور if..else د) دستور switch

پرسش‌های تشریحی

1- یک دستورالعمل منفرد در C++ بنویسید که اگر متغیر cout از 100 تجاوز کرد عبارت "Too many" را چاپ کند.

2- چه اشتباهی در کدهای زیر است؟

a.
```
cin << count;
```

b.
```
if x < y min = x
   else min = y;
```

3- چه اشتباهی در این کد برنامه وجود دارد؟

```
cout << "Enter n: ";
cin >> n;
if (n < 0)
   cout << "That is negative. Try again." << endl;
   cin >> n;
else
   cout << "o.k. n = " << n << endl;
```

4- چه تفاوتی بین کلمهٔ رزرو شده و شناسهٔ استاندارد است؟

5- مشخص کنید هر یک از عبارات زیر درست است یا نادرست. اگر نادرست است بگویید چرا؟

الف - عبارت (p||q) ! با عبارت !q || !p برابر است.

ب - عبارت !p!!! با عبارت !p برابر است.

ج - عبارت p && q || r با عبارت p && (q || r) برابر است.

6- برای هر یک از عبارت‌های بولی زیر یک جدول درستی بسازید که مقادیر درستی آن‌ها را (0 یا 1) به ازای هر مقدار از عملوندهای p و q نشان دهد:

الف - !p || !q

ب - !p&&!q || p&&q

ج - !(p&&q) && (p || q)

7- با استفاده از جدول درستی تعیین کنید که آیا دو عبارت بولی در هر یک از معادلات زیر برابرند یا خیر؟

الف – !q && !p و !(p && q)

ب – p و !p

ج – !q || p و !p || q

د – r && (p && q) و p && (q && r)

ه – r && (p || q) و p || (q && r)

8- ارزیابی میانبری چیست و چه فایده‌ای دارد؟

9- چه اشتباهی در کد زیر است؟

```
if (x = 0) cout << x << " = 0\n";
else cout << x << " != 0\n";
```

10- چه اشتباهی در کد زیر وجود دارد؟

```
if (x < y < z) cout << x << " < " << y << " < " <<
z << endl;
```

11- برای هر یک از شرط‌های زیر یک عبارت منطقی بسازید:

الف – score بزرگ‌تر یا مساوی 80 و کوچک‌تر از 90 باشد

ب – answer برابر با 'n' یا 'N' باشد

ج – n یک عدد زوج باشد ولی برابر با 8 نباشد

د – ch یک حرف بزرگ (capital) باشد

12- برای هر یک از شرط‌های زیر یک عبارت منطقی بسازید:

الف – n بین 0 و 7 باشد ولی برابر با 3 نباشد

ب – n بین 0 و 7 باشد ولی زوج نباشد

ج – n بر 3 بخش‌پذیر باشد ولی بر 30 بخش‌پذیر نباشد

د – ch یک حرف بزرگ یا کوچک باشد

13- چه اشتباهی در این کد است؟

```
if (x = = 0)
  if (y = = 0) cout << "x and y are both zero."
                    << endl;
else cout << "x is not zero." << endl;
```

14- چه تفاوتی بین دو دستورالعمل زیر است؟

a.
```
if (n > 2) { if (n < 6) cout << "OK"; }
  else cout << "NG";
```

b.
```
if (n > 2) { if (n < 6) cout << "OK" ;
                else cout << "NG"; }
```

15- تلهٔ سقوط چیست؟

16- عبارت زیر چگونه ارزیابی می‌شود؟

```
(x < y ? -1 : (x == y ? 0 : 1) );
```

17- یک دستورالعمل منفرد در C++ بنویسید که با استفاده از عملگر عبارت شرطی، قدرمطلق x را در متغیر absx قرار دهد.

18- یک دستورالعمل منفرد در C++ بنویسید که اگر متغیر count از 100 تجاوز کرد عبارت "Too many" را چاپ کند با استفاده از:

الف - یک دستورالعمل if

ب - یک عملگر عبارت شرطی

تمرین‌های برنامه‌نویسی

1- برنامهٔ مثال 1-3 را طوری تغییر دهید که تنها اگر n بر d قابل تقسیم باشد پاسخی را چاپ کند.
2- برنامهٔ مثال 5-3 را طوری تغییر دهید که کمینهٔ چهار عدد صحیح را چاپ کند.
3- برنامهٔ مثال 5-3 را طوری تغییر دهید که حد وسط سه عدد صحیح وارد شده را چاپ کند.
4- برنامهٔ مثال 6-3 را طوری تغییر دهید که همان اثر را داشته باشد اما بدون استفاده از بلوک دستورالعمل.
5- پیش‌بینی کنید خروجی برنامهٔ مثال 17-3 چیست اگر اعلان خط پنجم برنامه را پاک کنیم. برنامهٔ تغییریافته را برای بررسی پیش‌بینی خود اجرا کنید.
6- برنامه‌ای نوشته و اجرا کنید که سن کاربر را بخواند و اگر سن کوچک‌تر از 18 بود عبارت "You are a child" را چاپ کند و اگر سن بین 18 و 65 بود عبارت "You are an adult" را چاپ کند و اگر سن بزرگ‌تر یا مساوی 65 بود عبارت "you are a cenior citizen" را چاپ کند.
7- برنامه‌ای نوشته و اجرا کنید که دو عدد صحیح را می‌خواند و با استفاده از یک عملگر عبارت شرطی، با توجه به این که آیا یکی از این دو مضرب دیگری است یا خیر، عبارت "multiple" یا "not" را چاپ کند.
8- برنامه‌ای نوشته و اجرا کنید که یک ماشین‌حساب ساده را شبیه‌سازی می‌کند که دو عدد صحیح و یک کاراکتر را می‌خواند و سپس اگر کاراکتر (+) باشد مجموع را چاپ کند و اگر کاراکتر (-) باشد تفاضل را چاپ کند و اگر کاراکتر(*) باشد حاصل‌ضرب را چاپ کند و اگر کاراکتر (/) باشد حاصل‌تقسیم را چاپ کند و اگر کاراکتر (%) باشد باقیمانده تقسیم را چاپ کند. از یک دستورالعمل switch استفاده کنید.
9- برنامه‌ای نوشته و اجرا کنید که بازی "سنگ – کاغذ – قیچی" را انجام دهد. در این بازی دو نفر به طور هم‌زمان یکی از عبارات "سنگ" یا "کاغذ" یا "قیچی" را می‌گویند (و یا یکی از علامت‌های از قبل مشخص را با دست نشان می‌دهند). برنده کسی است که شیء غلبه‌کننده بر دیگری را انتخاب کرده باشد. حالات ممکن، چنین است که کاغذ

بر سنگ غلبه می‌کند (می‌پوشاند)، سنگ بر قیچی غلبه می‌کند (می‌شکند) و قیچی بر کاغذ غلبه می‌کند (می‌برد). برای اشیاء از یک نوع شمارشی استفاده کنید.

10- مسألهٔ 9 را با استفاده از دستور switch حل کنید.

11- مسألهٔ 9 را با استفاده از عبارات شرطی حل کنید.

12- برنامه‌ای را نوشته و اجرا کنید که یک معادلهٔ درجه دوم را حل می‌کند. معادلهٔ درجه دوم معادله‌ای است که به شکل $ax^2+bx+c=0$ باشد. a و b و c ضرایب هستند و x مجهول است. ضرایب، اعداد حقیقی هستند که توسط کاربر وارد می‌شوند. بنابراین باید از نوع float یا double اعلان گردند. از آن‌جا که معادلهٔ درجه دوم معمولا دو ریشه دارد، برای جواب‌ها از x1 و x2 استفاده کنید. جواب‌ها باید از نوع double اعلان گردند تا از خطای گرد کردن جلوگیری شود.

13- برنامه‌ای را نوشته و اجرا کنید که یک عدد شش رقمی را می‌خواند و مجموع شش رقم آن عدد را چاپ می‌کند. از عملگر تقسیم (/) و عملگر باقیمانده (٪) برای بیرون کشیدن رقم‌ها از عدد ورودی استفاده کنید. برای مثال اگر عدد ورودی n برابر با 876,543 باشد، آنگاه n/1000%10 برابر با رقم یکان هزار یعنی 6 است.

فصل چهارم

«تکرار»

مقدمه

تکرار[1]، اجرای پی در پی یک دستور یا بلوکی از دستورالعمل‌ها در یک برنامه است. با استفاده از تکرار می‌توانیم کنترل برنامه را مجبور کنیم تا به خطوط قبلی برگردد و آن‌ها را دوباره اجرا نماید. C++ دارای سه دستور تکرار است: دستور **while**، دستور **do_while** و دستور **for**. دستورهای تکرار به علت طبیعت چرخه‌مانندشان، ***حلقه***[2] نیز نامیده می‌شوند.

4-1 دستور while

نحو دستور **while** به شکل زیر است:

```
while (condition) statement;
```

به جای *condition*، یک شرط قرار می‌گیرد و به جای statement دستوری که باید تکرار شود قرار می‌گیرد. اگر مقدار شرط، صفر(یعنی نادرست) باشد، statement

---

1 – Iteration 2 - Loop

نادیده گرفته می‌شود و برنامه به اولین دستور بعد از **while** پرش می‌کند. اگر مقدار شرط ناصفر(یعنی درست) باشد، statement اجرا شده و دوباره مقدار شرط بررسی می‌شود. این تکرار آن قدر ادامه می‌یابد تا این که مقدار شرط صفر شود. توجه کنید که شرط باید درون پرانتز قرار بگیرد.

× مثال 1-4 محاسبهٔ حاصل جمع اعداد صحیح متوالی با حلقهٔ while

این برنامه مقدار 1 + 2 + 3 + … + n را برای عدد ورودی n محاسبه می‌کند:

```
int main()
{  int n, i=1;
   cout << "Enter a positive integer: ";
   cin >> n;
   long sum=0;
   while (i <= n)
      sum += i++;
   cout << "The sum of the first " << n << " integers is "
        << sum;
}
```

برنامهٔ بالا از سه متغیر محلی n و i و sum استفاده می‌کند. متغیر i با مقدار 1 مقداردهی اولیه می‌شود و عددی که کاربر وارد می‌کند در متغیر n قرار می‌گیرد. متغیر sum نیز با 0 مقداردهی اولیه می‌شود. سپس حلقهٔ while آغاز می‌گردد: ابتدا مقدار i با n مقایسه می‌شود. اگر i<=n بود مقدار i با مقدار sum جمع شده و حاصل در sum قرار می‌گیرد. به i یکی افزوده شده و دوباره شرط حلقه بررسی می‌شود. هنگامی که i>n شود حلقه متوقف می‌شود. پس n آخرین مقداری است که به sum افزوده می‌شود. شکل زیر پاسخ برنامه را به ازای ورودی n=8 نشان می‌دهد. همچنین مقدار متغیرها در هر گام حلقه در جدول نشان داده شده است.

```
Enter a positive integer: 8
The sum of the first 8 integers is 36
```

i	0	1	2	3	4	5	6	7	8
sum	0	1	3	6	10	15	21	28	36

در دومین اجرا، کاربر عدد 100 را وارد می‌کند، لذا حلقهٔ while نیز 100 بار تکرار می‌شود تا محاسبهٔ 5050=100+99+98+...+3+2+1 را انجام دهد:

```
Enter a positive integer: 100
The sum of the first 100 integers is 5050
```

به تورفتگی دستور داخل حلقه توجه کنید. این شکل چینش سبب می‌شود که منطق برنامه راحت‌تر دنبال شود، خصوصا در برنامه‌های بزرگ.

× مثال 2-4 استفاده از حلقهٔ while برای تکرار یک محاسبه

برنامهٔ زیر جذر هر عددی که کاربر وارد کند را محاسبه می‌نماید. در این برنامه از حلقهٔ while استفاده شده تا مجبور نباشیم برای محاسبهٔ جذر عدد بعدی، برنامه را دوباره اجرا کنیم:

```
int main()
{  double x;
   cout << "Enter a positive number: ";
   cin >> x;
   while (x > 0)
   {  cout << "sqrt(" << x << ") = " << sqrt(x) << endl;
      cout << "Enter another positive number (or 0 to quit): ";
      cin >> x;
   }
}
```

```
Enter a positive number: 49
sqrt(49) = 7
Enter another positive number (or 0 to quit): 3.14159
sqrt(3.14159) = 1.77245
Enter another positive number (or 0 to quit): 100000
sqrt(100000) = 316.228
Enter another positive number (or 0 to quit): 0
```

در این مثال، شرط کنترل حلقه عبارت **(x > 0)** است. مقدار x درون حلقه با تغییر عدد ورودی تغییر می‌کند. بنابراین فقط وقتی حلقه خاتمه می‌یابد که عدد ورودی برابر با 0 یا کم‌تر از آن باشد. متغیری که به این شکل برای کنترل حلقه استفاده شود، ***متغیر کنترل حلقه*** نامیده می‌شود.

4-2 خاتمه دادن به یک حلقه

قبلا دیدیم که چگونه دستور break برای کنترل دستورالعمل switch استفاده می‌شود (به مثال 3-17 نگاه کنید). از دستور break برای پایان دادن به حلقه‌ها نیز می‌توان استفاده کرد.

× مثال 4-3 استفاده از دستور break برای خاتمه دادن به یک حلقه

این برنامه همان تاثیر مثال 4-1 را دارد:

```
int main()
{  int n, i=1;
   cout << "Enter a positive integer: ";
   cin >> n;
   long sum=0;
   while (true)
   {  if (i > n) break;     // terminates the loop immediately
      sum += i++;
   }
   cout << "The sum of the first " << n << " integers is " << sum;
}
```

```
Enter a positive integer: 100
The sum of the first 100 integers is 5050
```

برنامهٔ بالا مانند مثال 4-1 کار می‌کند: همین که مقدار i به n برسد، حلقه خاتمه می‌یابد و دستور خروجی در پایان برنامه اجرا می‌شود.

توجه کنید که شرط کنترل حلقه true است. به این ترتیب حلقه برای همیشه تکرار می‌شود و هیچ‌گاه پایان نمی‌یابد اما در بدنهٔ حلقه شرطی هست که سبب پایان گرفتن حلقه می‌شود. به محض این که i>n شود دستور break حلقه را می‌شکند و کنترل به بیرون حلقه پرش می‌کند. وقتی قرار است حلقه از درون کنترل شود، معمولا شرط کنترل حلقه را true می‌گذارند. با این روش عملا شرط کنترل حلقه حذف می‌شود.

یکی از مزیت‌های دستور break این است که فورا حلقه را خاتمه می‌دهد بدون این که مابقی دستورهای درون حلقه اجرا شوند.

× مثال 4-4 اعداد فیبوناچی

اعداد فیبوناچی ... F_0, F_1, F_2, F_3, به شکل بازگشتی توسط معادله‌های زیر تعریف می‌شوند:

$$F_0 = 0 \quad , \quad F_1 = 1 \quad , \quad F_n = F_{n-1} + F_{n-2}$$

مثلا برای n=2 داریم:

$$F_2 = F_{2-1} + F_{2-2} = F_1 + F_0 = 0 + 1 = 1$$

یا برای n=3 داریم:

$$F_3 = F_{3-1} + F_{3-2} = F_2 + F_1 = 1 + 1 = 2$$

و برای n=4 داریم:

$$F_4 = F_{4-1} + F_{4-2} = F_3 + F_2 = 2 + 1 = 3$$

برنامهٔ زیر، همهٔ اعداد فیبوناچی را تا یک محدودهٔ مشخص که از ورودی دریافت می‌شود، محاسبه و چاپ می‌کند:

```
int main()
{  long bound;
   cout << "Enter a positive integer: ";
   cin >> bound;
   cout << "Fibonacci numbers < " << bound << ":\n0, 1";
   long f0=0, f1=1;
   while (true)
   {  long f2 = f0 + f1;
      if (f2 > bound) break;   // terminates the loop immediately
      cout << ", " << f2;
      f0 = f1;
      f1 = f2;
   }
}
```

```
Enter a positive integer: 1000
Fibonacci numbers < 1000:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987
```

حلقهٔ while شامل بلوکی از پنج دستور است. وقتی شرط (f2 > bound) درست باشد، دستور break اجرا شده و بدون این که سه دستور آخر حلقه اجرا شوند، حلقه فورا پایان می‌یابد.

توجه کنید که از کاراکتر خط جدید '\n' در رشتهٔ "\n0,1:" استفاده شده. این باعث می‌شود که علامت کولن : در پایان خط فعلی چاپ شود و سپس مکان‌نما به خط بعدی روی صفحه‌نمایش پرش نماید و رشتهٔ 0,1 را در شروع آن خط چاپ کند.

× مثال 5-4 استفاده از تابع exit(0)

تابع exit(0) روش دیگری برای خاتمه دادن به یک حلقه است. هرچند که این تابع بلافاصله اجرای کل برنامه را پایان می‌دهد:

```
int main()
{  long bound;
   cout << "Enter a positive integer: ";
   cin >> bound;
   cout << "Fibonacci numbers < " << bound << ":\n0, 1";
   long f0=0, f1=1;
   while (true)
   {  long f2 = f0 + f1;
      if (f2 > bound) exit(0); // terminates the program immediately
      cout << ", " << f2;
      f0 = f1;
      f1 = f2;
   }
}
```

```
Enter a positive integer: 1000
Fibonacci numbers < 1000:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987
```

برنامهٔ بالا پس از بدنهٔ حلقه هیچ دستور دیگری ندارد. پس خاتمه دادن حلقه به معنی پایان دادن برنامه است. به همین دلیل این برنامه مانند مثال 4-4 اجرا می‌شود.

این مثال یک راه برای خروج از حلقهٔ نامتناهی را نشان داد. مثال بعدی روش دیگری را نشان می‌دهد. اما برنامه‌نویسان ترجیح می‌دهند از break برای خاتمه دادن به حلقه‌های نامتناهی استفاده کنند زیرا قابلیت انعطاف بیشتری دارد.

× مثال 6-4 متوقف کردن یک حلقهٔ نامتناهی

اگر از راهکارهای خاتمهٔ حلقه استفاده نکنید، حلقه برای همیشه ادامه پیدا می‌کند و به طبع آن برنامه هم هیچ‌گاه به پایان نمی‌رسد. ممکن است شرط کنترلی که برای حلقه می‌نویسید هنگام اجرای برنامه هیچ‌گاه «نادرست» نشود و حلقه تا بی‌نهایت ادامه یابد. در چنین مواردی از سیستم عامل کمک بگیرید. با فشردن کلیدهای Ctrl+C سیستم عامل یک برنامه را به اجبار خاتمه می‌دهد. کلید Ctrl را پایین نگه داشته و کلید C روی صفحه‌کلید خود را فشار دهید تا برنامهٔ فعلی خاتمه پیدا کند. به کد زیر نگاه کنید:

```
int main()
{  long bound;
   cout << "Enter a positive integer: ";
   cin >> bound;
   cout << "Fibonacci numbers < " << bound << ":\n0, 1";
   long f0=0, f1=1;
   while (true)       // ERROR: INFINITE LOOP! Press <Ctrl>+c.)
   {  long f2 = f0 + f1;
      cout << ", " << f2;
      f0 = f1;
      f1 = f2;
   }
}
```

```
Enter a positive integer: 1000
Fibonacci numbers < 1000:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,
159781, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418,
317811, 5040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352,
24157817, 63245986, 102334155, 165580141, 267914296, 433494437,
```

چون هیچ شرط پایان حلقه‌ای در این برنامه وجود ندارد، اجرای برنامه تا بی‌نهایت ادامه خواهد یافت (تا وقتی حافظه سرریز شود). پس کلیدهای Ctrl+C را فشار دهید تا برنامه خاتمه یابد.

4-3 دستور do..while

ساختار do..while روش دیگری برای ساختن حلقه است. نحو آن به صورت زیر است:

```
do statement while (condition);
```

به جای *condition* یک شرط قرار می‌گیرد و به جای statement دستور یا بلوکی قرار می‌گیرد که قرار است تکرار شود. این دستور ابتدا statement را اجرا می‌کند و سپس شرط *condition* را بررسی می‌کند. اگر شرط درست بود حلقه دوباره تکرار می‌شود وگرنه حلقه پایان می‌یابد.

دستور **do..while** مانند دستور **while** است. با این فرق که شرط کنترل حلقه به جای این که در ابتدای حلقه ارزیابی گردد، در انتهای حلقه ارزیابی می‌شود. یعنی هر متغیر کنترلی به جای این که قبل از شروع حلقه تنظیم شود، می‌تواند درون آن تنظیم گردد. نتیجهٔ دیگر این است که حلقهٔ do..while همیشه بدون توجه به مقدار شرط کنترل، لااقل یک بار اجرا می‌شود اما حلقهٔ while می‌تواند اصلا اجرا نشود.

× مثال 4-7 محاسبهٔ حاصل جمع اعداد صحیح متوالی با حلقهٔ do..while

این برنامه همان تأثیر مثال 4-1 را دارد:

```
int main()
{  int n, i=0;
   cout << "Enter a positive integer: ";
   cin >> n;
   long sum=0;
   do
      sum += i++;
   while (i <= n);
   cout << "The sum of the first " << n << " integers is " << sum;
```

```
}
```

× مثال 8-4 اعداد فاکتوریال

اعداد فاکتوریل !0 و !1 و !2 و !3 و ... با استفاده از رابطه‌های بازگشتی زیر تعریف می‌شوند:

$$0! = 1 \quad , \quad n! = n(n-1)!$$

برای مثال، به ازای n = 1 در معادلهٔ دوم داریم:

$$1! = 1((1-1)!) = 1(0!) = 1(1) = 1$$

همچنین برای n = 2 داریم:

$$2! = 2((2-1)!) = 2(1!) = 2(1) = 2$$

و به ازای n = 3 داریم:

$$3! = 3((3-1)!) = 3(2!) = 3(2) = 6$$

برنامهٔ زیر همهٔ اعداد فاکتوریال را که از عدد داده شده کوچک‌ترند، چاپ می‌کند:

```
int main()
{  long bound;
   cout << "Enter a positive integer: ";
   cin >> bound;
   cout << "Factorial numbers < " << bound << ":\n1";
   long f=1, i=1;
   do
   {  cout << ", " << f;
      f *= ++i;
   }
   while (f < bound);
}
```

```
Enter a positive integer: 100000
Factorial numbers < 100000:
1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880
```

حلقهٔ do..while تا وقتی که شرط کنترل (**f < bound**) نادرست شود، تکرار می‌گردد.

4-4 دستور for

نحو دستورالعمل **for** به صورت زیر است:

for (initialization; *condition*; update**)** statement;

سه قسمت داخل پرانتز، حلقه را کنترل می‌کنند. عبارت initialization برای اعلان یا مقداردهی اولیه به متغیر کنترل حلقه استفاده می‌شود. این عبارت اولین عبارتی است که ارزیابی می‌شود پیش از این که نوبت به تکرارها برسد. عبارت *condition* برای تعیین این که آیا حلقه باید تکرار شود یا خیر به کار می‌رود. یعنی این عبارت، شرط کنترل حلقه است. اگر این شرط درست باشد دستور statement اجرا می‌شود. عبارت update برای پیش‌بردن متغیر کنترل حلقه به کار می‌رود. این عبارت پس از اجرای statement ارزیابی می‌گردد. بنابراین زنجیرهٔ وقایعی که تکرار را ایجاد می‌کنند عبارتند از:

1 – ارزیابی عبارت initialization

2 – بررسی شرط *condition* . اگر نادرست باشد، حلقه خاتمه می‌یابد.

3 – اجرای statement

4 – ارزیابی عبارت update

5 – تکرار گام‌های 2 تا 4

عبارت‌های initialization و *condition* وupdate عبارت‌های اختیاری هستند. یعنی می‌توانیم آن‌ها را در حلقه ذکر نکنیم.

× مثال 9-4 استفاده از حلقهٔ for برای محاسبهٔ مجموع اعداد صحیح متوالی

این برنامه همان تأثیر مثال 1-4 را دارد:

```
int main()
{  int n;
   cout << "Enter a positive integer: ";
```

```
   cin >> n;
   long sum=0;
   for (int i=1; i <= n; i++)
      sum += I;
   cout << "The sum of the first " << n << " integers is " << sum;
}
```

در حلقهٔ برنامهٔ فوق، عبارت مقداردهی اولیه **int i=1** است. شرط کنترل حلقه **i<=n** می‌باشد و عبارت پیش‌بری متغیر کنترل هم **i++** است. دقت کنید که این‌ها همان عباراتی هستند که در برنامهٔ مثال‌های 1-4 و 3-4 و 7-4 استفاده شده است.

در C++ استاندارد وقتی یک متغیر کنترل درون یک حلقهٔ for اعلان می‌شود (مانند i در مثال بالا) حوزهٔ آن متغیر به همان حلقهٔ for محدود می‌گردد. یعنی آن متغیر نمی‌تواند بیرون از آن حلقه استفاده شود. نتیجهٔ دیگر این است که می‌توان از نام مشابهی در خارج از حلقهٔ for برای یک متغیر دیگر استفاده نمود.

× مثال 10-4 استفادهٔ مجدد از اسامی متغیرهای کنترل حلقهٔ for

برنامهٔ زیر همان اثر برنامهٔ مثال 1-4 را دارد:

```
int main()
{  int n;
   cout << "Enter a positive integer: ";
   cin >> n;
   long sum=0;
   for (int i=1; i < n/2; i++)
      // the scope of this i is this loop
      sum += i;
   for (int i=n/2; i <= n; i++)
      // the scope of this i is this loop
      sum += i;
   cout << "The sum of the first " << n << " integers is " << sum ;
}
```

دو حلقهٔ for در برنامهٔ بالا همان محاسبات حلقهٔ for در برنامهٔ مثال 9-4 را انجام می‌دهند. این دو حلقه، کار را به دو قسمت تقسیم می‌کنند: n/2 محاسبه در حلقهٔ اول

انجام می‌گیرد و مابقی در حلقهٔ دوم. هر حلقه به طور مستقل متغیر کنترلی i خودش را دارد.

اخطار: بیشتر کامپایلرهای قبل از C++ استاندارد، حوزهٔ متغیر کنترلی حلقهٔ for را تا بعد از پایان حلقه نیز گسترش می‌دهند.

× مثال 4-11 دوباره اعداد فیبوناچی

این برنامه همان تأثیر برنامهٔ مثال 4-8 را دارد:

```
int main()
{  long bound;
   cout << "Enter a positive integer: ";
   cin >> bound;
   cout << "Factorial numbers < " << bound << ":\n1";
   long f=1;
   for (int i=2; f <= bound; i++)
   {  cout << ", " << f;
      f *= i;
   }
}
```

```
Enter a positive integer: 100000
Factorial numbers < 100000:
1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880
```

برنامهٔ بالا را با مثال 4-8 مقایسه کنید. هر دو کارهای مشابهی را انجام می‌دهند. در هر دو برنامه، کارهای زیر انجام می‌شود: مقدار اولیهٔ 1 در f قرار می‌گیرد. مقدار اولیهٔ 2 در i قرار داده می‌شود و سپس پنج گام تکرار رخ می‌دهد: چاپ f، ضرب f در i، افزایش i، بررسی شرط (f <= bound) و پایان دادن به حلقه در صورت نادرست بودن شرط. این برنامه با حلقهٔ for همان تأثیر برنامه با حلقهٔ do..while را دارد.

دستور for انعطاف‌پذیری بیشتری به برنامه می‌دهد. مثال‌های زیر این مطلب را آشکار می‌کنند.

× مثال 4-12 یک حلقهٔ for نزولی

برنامهٔ زیر ده عدد صحیح مثبت را به ترتیب نزولی چاپ می‌کند:

```
int main()
{  for (int i=10; i > 0; i--)
      cout << " " << i;
}
```

```
10 9 8 7 6 5 4 3 2 1
```

× مثال 13-4 استفاده از حلقهٔ for با گام‌های بزرگ‌تر از یک

برنامهٔ زیر مشخص می‌کند که آیا یک عدد ورودی اول هست یا خیر:

```
int main()
{  long n;
   cout << "Enter a positive integer: ";
   cin >> n;
   if (n < 2) cout << n << " is not prime." << endl;
   else if (n < 4) cout << n << " is prime." << endl;
   else if (n%2 == 0) cout << n << " = 2*" << n/2 << endl;
   else
   {  for (int d=3; d <= n/2; d+=2)
         if (n%d == 0)
         {  cout << n << " = " << d << "*" << n/d << endl;
            exit(0);
         }
      cout << n << " is prime." << endl;
   };
}
```

```
Enter a positive integer: 101
101 is prime.
```

```
Enter a positive integer: 975313579
975313579 = 17*57371387
```

توجه کنید که حلقهٔ **for** در برنامهٔ بالا متغیر کنترلی خود یعنی d را دو واحد دو واحد افزایش می‌دهد. سعی کنید منطق برنامهٔ بالا را توضیح دهید.

× مثال 14-4 استفاده از نگهبان برای کنترل حلقهٔ for

این برنامه مقدار بیشینهٔ یک رشته از اعداد ورودی را پیدا می‌کند:

```
int main()
{  int n, max;
   cout << "Enter positive integers (0 to quit): ";
   cin >> n;
   for (max = n; n > 0; )
   {  if (n > max) max = n;
      cin >> n;
   }
   cout << "max = " << max << endl;
}
```

```
Enter positive integers (0 to quit): 44 77 55 22 99 33 11 66 88 0
max = 99
```

حلقهٔ for در برنامهٔ بالا به وسیلهٔ متغیر ورودی n کنترل می‌شود. این حلقه ادامه می‌یابد تا زمانی که n<=0 بشود. متغیر ورودی که به این شیوه برای کنترل حلقه نیز استفاده شود، ***نگهبان*** نامیده می‌شود.

به بخش کنترلی این حلقه که به صورت **(max = n; n > 0;)** است دقت کنید. بخش پیش‌بری در آن وجود ندارد و بخش مقداردهی آن نیز متغیر جدیدی را تعریف نمی‌کند بلکه از متغیرهایی که قبلا در برنامه تعریف شده استفاده می‌برد. علت این است که حلقهٔ مذکور نگهبان دارد و نگهبان از طریق ورودی پیش برده می‌شود و دیگر نیازی به بخش پیش‌بری در حلقه نیست. متغیر max نیز باید مقدار خود را پس از اتمام حلقه حفظ کند تا در خروجی چاپ شود. اگر متغیر max درون حلقه اعلان می‌شد، پس از اتمام حلقه از بین می‌رفت و دیگر قابل استفاده نبود.

× مثال 15-4 بیشتر از یک متغیر کنترل در حلقهٔ for

حلقهٔ for در برنامهٔ زیر دو متغیر کنترل دارد:

```
int main()
{  for (int m=95, n=11, m%n > 0; m -= 3, n++)
      cout << m << "%" << n << " = " << m%n << endl;
}
```

```
95%11 = 7
92%12 = 8
89%13 = 11
```

```
86%14 = 2
83%15 = 8
```

در بخش کنترل این حلقه، دو متغیر m و n به عنوان متغیر کنترل اعلان و مقداردهی شده‌اند. در هر تکرار حلقه، m سه واحد کاسته شده و n یک واحد افزایش می‌یابد. در نتیجه زوج‌های (m,n) به شکل (95,11) و (92,12) و (89,13) و (86,14) و (83,15) و (80,16) تولید می‌شوند. چون 80 بر 16 بخش‌پذیر است، حلقه با زوج (80,16) پایان می‌یابد.

× مثال 16-4 حلقه‌های for تودرتو

برنامهٔ زیر یک جدول ضرب چاپ می‌کند:

```
#include <iomanip>    // defines setw()
#include <iostream>   // defines cout
using namespace std;
int main()
{  for (int x=1; x <= 10; x++)
   {  for (int y=1; y <= 10; y++)
         cout << setw(4) << x*y;
      cout << endl;
   }
}
```

```
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
  4   8  12  16  20  24  28  32  36  40
  5  10  15  20  25  30  35  40  45  50
  6  12  18  24  30  36  42  48  54  60
  7  14  21  28  35  42  49  56  63  70
  8  16  24  32  40  48  56  64  72  80
  9  18  27  36  45  54  63  72  81  90
 10  20  30  40  50  60  70  80  90 100
```

در اولین تکرار از حلقهٔ بیرونی، وقتی که x=1 است، حلقهٔ درونی ده مرتبه تکرار می‌شود و به ازای y=1 تا 10 مقادیر 1*y را روی یک ردیف چاپ می‌کند. وقتی حلقهٔ درونی پایان یافت، با دستور cout << endl; مکان‌نما به سطر بعدی روی صفحه‌نمایش منتقل می‌شود. حالا دومین تکرار حلقهٔ بیرونی به ازای x=2 آغاز

می‌شود. دوباره حلقهٔ درونی ده مرتبه تکرار می‌شود و این دفعه مقادیر 2*y روی یک سطر چاپ می‌شود. دوباره با دستور cout << endl; مکان‌نما به سطر بعدی می‌رود و تکرار سوم حلقهٔ بیرونی شروع می‌شود. این رویه ادامه می‌یابد تا این که حلقهٔ بیرونی برای بار دهم تکرار شده و آخرین سطر جدول هم چاپ می‌شود و سپس برنامه خاتمه می‌یابد.

در این برنامه از شکل‌دهندهٔ فرایند setw استفاده شده. عبارت setw(4) به این معنی است که طول ناحیهٔ چاپ را برای خروجی بعدی به اندازهٔ چهار کاراکتر تنظیم کن. به این ترتیب اگر خروجی کم‌تر از چهار کاراکتر باشد، فضای خالی به خروجی مربوطه پیوند زده می‌شود تا طول خروجی به اندازهٔ چهار کاراکتر شود. نتیجه این است که خروجی نهایی به شکل یک جدول مرتب روی ده سطر و ده ستون زیر هم چاپ می‌شود. شکل‌دهنده‌های فرایند در سرفایل <iomanip> تعریف شده‌اند. بنابراین برای استفاده از شکل‌دهنده‌های فرآیند باید راهنمای پیش‌پردازندهٔ #include <iomanip> را به ابتدای برنامه بیافزایید. همچنین برنامه باید دارای راهنمای پیش‌پردازندهٔ #include<iostream> نیز باشد.

4-5 دستور break

دستور **break** یک دستور آشناست. قبلا از آن برای خاتمه دادن به دستور switch و همچنین حلقه‌های while و do..while استفاده کرده‌ایم. از این دستور برای خاتمه دادن به حلقهٔ for نیز می‌توانیم استفاده کنیم. دستور break انعطاف‌پذیری بیشتری را برای حلقه‌ها ایجاد می‌کند. معمولا یک حلقهٔ while ، یک حلقهٔ do..while یا یک حلقهٔ for فقط در شروع یا پایان مجموعهٔ کامل دستورالعمل‌های موجود در بلوک حلقه، خاتمه می‌یابد. دستور break در هر جایی درون حلقه می‌تواند جا بگیرد و در همان جا حلقه را خاتمه دهد.

× مثال 4-17 کنترل ورودی با یک نگهبان

این برنامه یک رشته اعداد صحیح مثبت را تا زمانی که صفر وارد شود، خوانده و معدل آن‌ها را محاسبه می‌کند:

```
int main()
{  int n, count=0, sum=0;
   cout << "Enter positive integers (0 to quit):" << endl;
   for (;;)    // "forever"
   {  cout << "\t" << count + 1 << ": ";
      cin >> n;
      if (n <= 0) break;
      ++count;
      sum += n;
   }
   cout << "The average of those " << count << " positive
            numbers is "<< float(sum)/count << endl;
}
```

```
Enter positive integers (0 to quit):
        1: 4
        2: 7
        3: 1
        4: 5
        5: 2
        6: 0
The average of those 5 positive numbers is 3.8
```

در برنامهٔ بالا وقتی که 0 وارد شود، دستور break اجرا شده و حلقه فورا خاتمه می‌یابد و اجرای برنامه به اولین دستور بعد از حلقه پرش می‌کند. به نحوهٔ نوشتن دستور for در این برنامه دقت کنید. هر سه بخش کنترلی در این حلقه، خالی است: **for(; ;)**. این ترکیب به معنای بی‌انتهایی است. یعنی بدون دستور break این حلقه یک حلقهٔ نامتناهی می‌شود.

وقتی دستور break درون حلقه‌های تودرتو استفاده شود، فقط روی حلقه‌ای که مستقیما درون آن قرار گرفته تاثیر می‌گذارد. حلقه‌های بیرونی بدون هیچ تغییری ادامه می‌یابند.

× مثال 18-4 استفاده از دستور break در حلقه‌های تودرتو

چون عمل ضرب جابجایی‌پذیر است (یعنی 3×4 = 4×3)، برای ایجاد یک جدول ضرب فقط کافی است اعداد قطر پایینی مشخص شوند. این برنامه، مثال 16-4 را برای چاپ یک جدول ضرب مثلثی تغییر می‌دهد:

```
int main()
{  for (int x=1; x <= 10; x++)
   {  for (int y=1; y <= 10; y++)
         if (y > x) break;
         else cout << setw(4) << x*y;
      cout << endl;
   }
}
```

```
 1
 2   4
 3   6   9
 4   8  12  16
 5  10  15  20  25
 6  12  18  24  30  36
 7  14  21  28  35  42  49
 8  16  24  32  40  48  56  64
 9  18  27  36  45  54  63  72  81
10  20  30  40  50  60  70  80  90 100
```

وقتی y>x باشد، اجرای حلقهٔ y درونی خاتمه می‌یابد و تکرار بعدی حلقهٔ خارجی x شروع می‌شود. مثلا وقتی x=3 باشد، حلقهٔ y سه بار تکرار می‌شود و خروجی‌های 3 و 6 و 9 چاپ می‌شوند. در تکرار چهارم، شرط (y>x) برابر با درست ارزیابی می‌شود. پس دستور break اجرا شده و کنترل فورا به خط cout<<endl; منتقل می‌شود (زیرا این خط اولین دستور خارج از حلقهٔ درونی y است). آنگاه حلقهٔ بیرونی x تکرار چهارم را با x=4 آغاز می‌کند.

4-6 دستور continue

دستور break بقیهٔ دستورهای درون بلوک حلقه را نادیده گرفته و به اولین دستور بیرون حلقه پرش می‌کند. دستور **continue** نیز شبیه همین است اما به جای

این که حلقه را خاتمه دهد، اجرا را به تکرار بعدی حلقه منتقل می‌کند. این دستور، ادامهٔ چرخهٔ فعلی را لغو کرده و اجرای دور بعدی حلقه را آغاز می‌کند.

× مثال 19-4 استفاده از دستورهای break و continue

این برنامهٔ کوچک، دستورهای break و continue را شرح می‌دهد:

```
int main()
{  int n = 1;
   char c;
   for( ; ;n++ )
   {  cout << "\nLoop no: " << n << endl;
      cout << "Continue? <y|n> ";
      cin >> c;
      if (c = = 'y') continue;
      break;
   }
   cout << "\nTotal of loops: " << n;
}
```

```
Loop no: 1
Continue? y
Loop no: 2
Continue? y
Loop no: 3
Continue? n
Total of loops: 3
```

برنامهٔ بالا تعداد تکرار حلقه را می‌شمارد. در ابتدای هر حلقه با چاپ n مشخص می‌شود که چندمین دور حلقه در حال اجراست. سپس از کاربر درخواست می‌شود تا یک کاراکتر را به عنوان انتخاب، وارد کند. اگر کاراکتر وارد شده 'y' باشد، شرط (c=='y') برابر با درست ارزیابی می‌شود و لذا دستور continue اجرا شده و دور جدید حلقه شروع می‌شود. اگر کاراکتر وارد شده هر چیزی غیر از 'y' باشد، دستور break این حلقه را خاتمه می‌دهد و کنترل اجرا به اولین دستور بیرون حلقه پرش می‌کند. سپس مجموع دفعاتی که حلقه تکرار شده چاپ می‌گردد و برنامه پایان می‌گیرد.

4-7 دستور goto

دستورهای break و continue و switch باعث می‌شوند که اجرای برنامه به مکان دیگری از جایی که به طور طبیعی باید می‌رفت، منتقل شود. مقصد انتقال را نوع دستور تعیین می‌کند: break به خارج از حلقه می‌رود، continue به شرط ادامهٔ حلقه(دور بعدی حلقه) می‌رود و switch به یکی از ثابت‌های case می‌رود. هر سهٔ این دستورها ***دستور پرش***[1] هستند زیرا باعث می‌شوند اجرای برنامه از روی دستورهای دیگر پرش کند.

دستور**goto** نوع دیگری از دستورهای پرش است. مقصد این پرش توسط یک برچسب معین می‌شود. ***برچسب***[2] شناسه‌ای است که جلوی آن علامت کولن(:) می‌آید و جلوی یک دستور دیگر قرار می‌گیرد. برچسب‌ها شبیه caseها در دستور switch هستند، یعنی مقصد پرش را مشخص می‌کنند. یک مزیت دستور **goto** این است که با استفاده از آن می‌توان از همهٔ حلقه‌های تودرتو خارج شد و به مکان دلخواهی در برنامه پرش نمود.

مثال زیر نشان می‌دهد که دستور break فقط درونی‌ترین حلقه را خاتمه می‌دهد ولی با دستور **goto** می‌توان چند حلقه یا همهٔ حلقه‌ها را یک‌جا خاتمه داد.

× مثال 20-4 استفاده از دستور goto برای خارج شدن از حلقه‌های تودرتو

```
int main()
{  const int N=5;
   for (int i=0; i<N; i++)
   {  for (int j=0; j<N; j++)
      {  for (int k=0; k<N; k++)
            if (i+j+k>N) goto esc;
            else cout << i+j+k << " ";
         cout << "* ";
      }
   esc: cout << "." << endl;   // inside the i loop, outside the j loop
   }
}
```

```
0 1 2 3 4 * 1 2 3 4 5 * 2 3 4 5 .
```

```
1 2 3 4 5 * 2 3 4 5 .
2 3 4 5 .
3 4 5 .
4 5 .
```

1 – Jump 2 - Label

اگر شرط (i+j+k>N) در درونی‌ترین حلقه درست شود، به دستور goto می‌رسیم. وقتی این دستور اجرا شود، اجرای برنامه به سطری که برچسب **esc** دارد منتقل می‌شود. این خط بیرون از حلقه‌های j و k است. پس با این پرش هر دوی این حلقه‌ها پایان می‌گیرند. وقتی i و j صفر هستند، حلقهٔ k پنج بار تکرار می‌گردد و 0 1 2 3 4 به همراه ستاره × چاپ می‌شود. آنگاه j به 1 افزایش می‌یابد و حلقهٔ k پنج بار دیگر تکرار شده و این بار 1 2 3 4 5 همراه یک ستاره × چاپ می‌شود. سپس j به 2 افزایش می‌یابد و حلقهٔ k چهار بار دیگر تکرار می‌شود و 2 3 4 5 چاپ می‌شود. اما در تکرار بعدی حلقهٔ k شرط (i+j+k>0) درست می‌شود زیرا حالا i+j+k = 6 است. پس دستور **goto** برای اولین بار اجرا شده و کنترل برنامه به سطر برچسب‌دار که یک دستور خروجی است پرش می‌کند، یک نقطه چاپ شده و مکان‌نما به سطر بعد منتقل می‌شود. توجه کنید که حلقه‌های j و k بدون این که تکرارهایشان را کامل کنند، ناتمام رها می‌شوند. دستور برچسب خورده، خود جزو بدنهٔ حلقهٔ i است. لذا پس از پایان گرفتن اجرای این سطر، تکرار بعدی حلقهٔ i آغاز می‌شود. حالا i=1 است و حلقهٔ j دوباره با j=0 شروع می‌شود و این خود باعث می‌گردد حلقهٔ k نیز با k=0 دوباره شروع شود. برنامه به همین ترتیب ادامه می‌یابد و خروجی نهایی حاصل می‌شود.

با استفاده از دستور **goto** می‌توان از هر قسمت برنامه به هر قسمت دیگری پرش کرد. گرچه این دستور باعث می‌شود راحت‌تر بتوانیم از تکرار حلقه‌ها خلاص شویم یا آسان‌تر به سطر دلخواه انشعاب کنیم اما تجربه نشان داده که استفادهٔ بی‌مهابا از دستور **goto** سبب افزایش خطاهای زمان اجرا و کاهش پایداری برنامه می‌شود.

× مثال 21-4 خارج شدن از چند حلقهٔ تودرتو بدون استفاده از goto

این برنامه همان اثر برنامهٔ مثال 20-4 را دارد:

```
int main()
{  const int N=5;
   bool done=false;
   for (int i=0; i<N; i++)
   {  for (int j=0; j<N && !done; j++)
      {  for (int k=0; k<N && !done; k++)
            if (i+j+k>N) done = true;
            else cout << i+j+k << " ";
         cout << "* ";
      }
      cout << "." << endl;    // inside the i loop, outside the j loop
      done = false;
   }
}
```

در برنامهٔ بالا از یک پرچم به نام done استفاده شده. وقتی done برابر با true شود، هر دو حلقهٔ درونی k و j خاتمه می‌یابد و حلقهٔ خارجی i تکرارش را با چاپ یک نقطه ادامه می‌دهد و پرچم را دوباره false می‌کند و دور جدید را آغاز می‌نماید. گرچه منطق این برنامه کمی پیچیده‌تر است اما قابلیت اطمینان بیشتری نسبت به برنامهٔ مثال 20-4 دارد زیرا هیچ حلقه‌ای نیمه‌کاره نمی‌ماند و هیچ متغیری بلاتکلیف رها نمی‌شود.

4-8 تولید اعداد شبه تصادفی[1]

یکی از کاربردهای بسیار مهم رایانه‌ها، «شبیه‌سازی[2]» سیستم‌های دنیای واقعی است. تحقیقات و توسعه‌های بسیار پیشرفته به این راهکار خیلی وابسته است. به وسیلهٔ شبیه‌سازی می‌توانیم رفتار سیستم‌های مختلف را مطالعه کنیم بدون این که لازم باشد واقعا آن‌ها را پیاده‌سازی نماییم. در شبیه‌سازی نیاز است «اعداد تصادفی» توسط رایانه‌ها تولید شود تا نادانسته‌های دنیای واقعی مدل‌سازی شود. البته رایانه‌ها «ثابت‌کار» هستند یعنی با دادن داده‌های مشابه به رایانه‌های مشابه، همیشه خروجی یکسان تولید می‌شود. با وجود این می‌توان اعدادی تولید کرد که به ظاهر تصادفی هستند؛ اعدادی که

به طور یکنواخت در یک محدودهٔ خاص گسترده‌اند و برای هیچ‌کدام الگوی مشخصی وجود ندارد. چنین اعدادی را «اعداد شبه‌تصادفی» می‌نامیم.

سرفایل <cstdlib> در c استاندارد دارای تابعی به نام rand() است که این تابع اعداد صحیح شبه‌تصادفی در محدودهٔ صفر تا RAND_MAX تولید می‌نماید.

---

1 – Pseudo Random 2 - Simulation

RAND_MAX ثابتی است که آن هم در سرفایل <cstdlib> تعریف شده. هر بار که تابع rand() فراخوانی شود، یک عدد صحیح متفاوت از نوع unsigned تولید می‌کند که این عدد در محدودهٔ ذکر شده قرار دارد.

× مثال 22-4 تولید اعداد شبه تصادفی

این برنامه از تابع rand() برای تولید اعداد شبه‌تصادفی استفاده می‌کند:

```
#include <cstdlib>  // defines the rand() and RAND_MAX
#include <iostream>

int main()
{   // prints pseudo-random numbers:
   for (int i = 0; i < 8; i++)
      cout << rand() << endl;
   cout << "RAND_MAX = " << RAND_MAX << endl;
}
```

```
1103527590
377401575
662824084
1147902781
2035015474
368800899
1508029952
486256185
RAND_MAX = 2147483647
```

هر بار که برنامهٔ بالا اجرا شود، رایانه هشت عدد صحیح unsigned تولید می‌کند که به طور یکنواخت در فاصلهٔ 0 تا RAND_MAX گسترده شده‌اند. RAND_MAX در این

رایانه برابر با 2,147,483,647 است. خروجی زیر، اجرای دیگری از برنامهٔ بالا را نشان می‌دهد:

```
1103527590
377401575
662824084
1147902781
2035015474
368800899
1508029952
486256185
RAND_MAX = 2147483647
```

متاسفانه هر بار که برنامه اجرا می‌شود، همان اعداد قبلی تولید می‌شوند زیرا این اعداد از یک هستهٔ مشترک ساخته می‌شوند.

هر عدد شبه‌تصادفی از روی عدد قبلی خود ساخته می‌شود. اولین عدد شبه‌تصادفی از روی یک مقدار داخلی که «هسته[1]» گفته می‌شود ایجاد می‌گردد. هر دفعه که برنامه اجرا شود، هسته با یک مقدار پیش‌فرض بارگذاری می‌شود. برای حذف این اثر نامطلوب که از تصادفی بودن اعداد می‌کاهد، می‌توانیم با استفاده از تابع srand() خودمان مقدار هسته را انتخاب کنیم.

× مثال 23-4 کارگذاری هسته به طور محاوره‌ای

این برنامه مانند برنامهٔ مثال 22-4 است بجز این که می‌توان هستهٔ تولیدکنندهٔ اعداد تصادفی را به شکل محاوره‌ای وارد نمود:

```
#include <cstdlib>  // defines the rand() and srand()
#include <iostream>

int main()
{   // prints pseudo-random numbers:
   unsigned seed;
   cout << "Enter seed: ";
   cin >> seed;
   srand(seed);      // initializes the seed
   for (int i = 0; i < 8; i++)
      cout << rand() << endl;
}
```

سه اجرای متفاوت از برنامهٔ بالا نشان داده شده است:

```
Enter seed: 0
12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
```

1 – Seed

```
Enter seed: 1
1103527590
377401575
662824084
1147902781
2035015474
368800899
1508029952
486256185
```

```
Enter seed: 12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
794471793
```

خط `srand(seed);` مقدار متغیر `seed` را به هستهٔ داخلی تخصیص می‌دهد. این مقدار توسط تابع `rand()` برای تولید اعداد شبه‌تصادفی استفاده می‌شود. هسته‌های متفاوت، نتایج متفاوتی را تولید می‌کنند.

توجه کنید که مقدار متغیر `seed` که در سومین اجرای برنامه استفاده شده (`12345`) اولین عددی است که توسط تابع `rand()` در اجرای اول تولید شده بود. در نتیجه اعداد اول تا هشتم که در اجرای سوم تولید شده با اعداد دوم تا نهم که در اجرای اول تولید شده بود برابر است. همچنین دقت کنید که رشته اعداد تولید شده در

اجرای دوم مانند رشته اعداد تولید شده در مثال 22-4 است. این موضوع القا می‌کند که مقدار پیش‌فرض هسته در این رایانه، عدد یک است.

این که مقدار هسته باید به طور محاوره‌ای وارد شود مشکلی است که با استفاده از ساعت سیستم حل می‌شود. «ساعت سیستم[1]» زمان فعلی را بر حسب ثانیه نگه می‌دارد. تابع time() که در سرفایل <ctime> تعریف شده زمان فعلی را به صورت یک عدد صحیح unsigned برمی‌گرداند. این مقدار می‌تواند به عنوان هسته برای تابع rand() استفاده شود.

1 – System timer

× مثال 24-4 کارگذاری هسته از ساعت سیستم

برنامهٔ زیر، همان برنامهٔ مثال 23-4 است با این فرق که هستهٔ تولیدکنندهٔ اعداد شبه‌تصادفی را با استفاده از ساعت سیستم تنظیم می‌کند.

توجه: اگر کامپایلر شما سرفایل <ctime> را تشخیص نمی‌دهد، به جای آن از سرفایل <time.h> استفاده کنید.

```
#include <cstdlib>
#include <ctime>      // defines the time() function
#include <iostream>
//#include <time.h>   // use this if <ctime> is not recognized
int main()
{   // prints pseudo-random numbers:
   unsigned seed = time(NULL);      // uses the system clock
   cout << "seed = " << seed << endl;
   srand(seed);                     // initializes the seed
   for (int i = 0; i < 8; i++)
      cout << rand() << endl;
}
```

```
seed = 808148157
1877361330
352899587
1443923328
1857423289
```

```
200398846
1379699551
1622702508
715548277
```

```
seed = 808148160
892939769
1559273790
1468644255
952730860
1322627253
844657339
440402904
```

در اولین اجرا، تابع time() عدد صحیح 808,148,157 را برمی‌گرداند که به عنوان هستهٔ تولید کنندهٔ اعداد تصادفی استفاده شده است. دومین اجرا 3 ثانیه بعد انجام شده، بنابراین تابع time() عدد صحیح 808,148,160 را برمی‌گرداند که این مقدار، رشته اعداد کاملا متفاوتی را تولید می‌کند.

دو اجرای زیر روی یک pc با پردازندهٔ intel انجام شده است:

```
seed = 943364015
2948
15841
72
25506
30808
29709
13155
2527
```

```
seed = 943364119
17427
20464
13149
5702
12766
1424
16612
31746
```

در بیشتر برنامه‌های کاربردی، نیاز است که اعداد تصادفی در محدودهٔ مشخصی پخش شده باشند. مثال بعدی طریقهٔ انجام این کار را نشان می‌دهد.

× مثال 25-4 تولید اعداد تصادفی در یک محدودهٔ مشخص

برنامهٔ زیر مانند مثال 24-4 است به جز این که اعدادی که برنامهٔ زیر تولید می‌کند در یک ناحیهٔ مشخص محدود شده:

```
#include <cstdlib>
#include <ctime>      // defines the time() function
#include <iostream>
//#include <time.h>   // use this if <ctime> is not recognized
int main()
{  // prints pseudo-random numbers:
   unsigned seed = time(NULL);     // uses the system clock
   cout << "seed = " << seed << endl;
   srand(seed);                    // initializes the seed
   int min, max;
   cout << "Enter minimum and maximum: ";
   cin >> min >> max;           // lowest and highest numbers
   int range = max - min + 1;   // number of numbers in rsnge
   for (int i = 0; i < 20; i++)
   {  int r = rand()/100%range + min;
      cout << r << " ";
   }
   cout << endl;
}
```

```
seed = 808237677
Enter minimum and maximum: 1 100
85 57 1 10 5 73 81 43 46 42 17 44 48 9 3 74 41 4 30 68
```

```
seed = 808238101
Enter minimum and maximum: 22 66
63 29 56 22 53 57 39 56 43 36 62 30 41 57 26 61 59 26 28
```

اولین اجرا 20 عدد صحیح تصادفی بین 1 و 100 تولید می‌کند. دومین اجرا 20 عدد صحیح که بین 22 و 66 گسترش یافته را تولید می‌نماید.

در حلقهٔ for ابتدا مقدار rand() بر 100 تقسیم می‌شود تا دو رقم سمت راست عدد تصادفی حذف شود زیرا این مولد به طور متناوب اعداد زوج و فرد تولید

می‌کند. با حذف دو رقم سمت راست عدد تولید شده، این مشکل برطرف می‌شود. سپس عبارت rand()/100%range اعداد تصادفی در محدودهٔ 0 تا range-1 تولید نموده و عبارت rand()/100%range + min اعدادی تصادفی در محدودهٔ min تا max تولید می‌نماید.

پرسش‌های گزینه‌ای

1 – کدام یک از دستورات زیر، یک حلقه نیست؟

الف – while ب – switch

ج – for د – do..while

2 – در مورد کد `while (false) i++;` **کدام گزینه صحیح است؟**

الف – حلقهٔ فوق تا بی‌نهایت ادامه می‌یابد و مقدار i بی‌نهایت می‌شود.

ب – حلقهٔ فوق آن قدر ادامه می‌یابد تا این که i سرریز شود.

ج – حلقهٔ فوق فقط یک بار اجرا می‌شود و i فقط یک واحد اضافه می‌شود.

د – حلقهٔ فوق اصلا اجرا نمی‌شود.

3 – اگر i از نوع صحیح و مقدار آن 5 باشد، پس از خاتمهٔ حلقهٔ زیر مقدار i برابر با کدام گزینه است؟

```
While (i<10) i+=2;
```

الف – 10 ب – 11 ج – 9 د – 12

4 – اگر n از نوع صحیح و مقدار آن 0 باشد، پس از خاتمهٔ حلقهٔ زیر مقدار n برابر با کدام گزینه است؟

```
While (n<5)
{  if (n>3) break;
   n++;
}
```

الف – 4 ب – 5 ج – 6 د – 3

5 – اگر k از نوع صحیح و مقدار آن 5 باشد، پس از خاتمهٔ حلقهٔ زیر مقدار k برابر با کدام گزینه است؟

```
do
   k++
while (k<5);
```

الف – 5 ب – 6 ج – k سرریز می شود د – k پاریز می شود.

6 – کدام گزینه صحیح نیست؟

الف– دستور break حلقه را خاتمه می‌دهد.

ب – دستور exit(0) حلقه را خاتمه می‌دهد.

ج – دستور continue حلقه را خاتمه می‌دهد.

د – غلط بودن شرط کنترل، حلقه را خاتمه می‌دهد.

7 – حلقهٔ مقابل چند بار تکرار می‌شود؟ `for (j=0; true; j++) {…}`

الف – اصلا تکرار نمی‌شود
ب – تا بی‌نهایت ادامه می‌یابد
ج – فقط یک بار تکرار می‌شود
د – اگر در بدنهٔ حلقه، دستور خاتمهٔ حلقه وجود نداشته باشد تا بی‌نهایت ادامه می‌یابد.

8 – در پایان حلقه‌های مقابل مقدار k چقدر است؟

```
k = 0;
for (i=0; i<5; i++)
   for (j=0; j<5; j++)
      k++;
```

الف – 5 ب – 10 ج – 25 د – 50

9 – کدام گزینه صحیح است؟

الف – حلقهٔ do..while دست کم یک بار اجرا می‌شود.
ب – حلقهٔ while دست کم یک بار اجرا می‌شود.
ج – حلقهٔ for دست کم یک بار اجرا می‌شود
د – حلقهٔ while و حلقهٔ for دست کم یک بار اجرا می‌شوند.

10 – در کدام حلقه، شرط کنترل حلقه در انتهای هر تکرار بررسی می‌شود؟

الف – حلقهٔ while ب - حلقهٔ do..while ج – حلقهٔ for د – هیچ‌کدام

11 – کدام حلقه نمی‌تواند تا بی‌نهایت ادامه یابد؟

الف – حلقهٔ while ب – حلقهٔ do..while ج – حلقهٔ for د – هیچ‌کدام

12 – در مورد حلقه‌های تودرتو کدام عبارت صحیح است؟

الف – دستور break فقط درونی‌ترین حلقه را خاتمه می‌دهد
ب – دستور break فقط بیرونی‌ترین حلقه را خاتمه می‌دهد
ج – دستور break فقط حلقه‌ای که این دستور در بدنهٔ آن قرار دارد را خاتمه می‌دهد
د – دستور break حلقه‌های درونی و بیرونی را یک‌جا خاتمه می‌دهد.

13 – کدام عبارت صحیح است؟

الف – حلقهٔ for را می‌توان به حلقهٔ while یا حلقهٔ do..while تبدیل کرد.
ب – حلقهٔ for را نمی‌توان به حلقهٔ while یا حلقهٔ do..while تبدیل کرد.
ج – حلقهٔ for را فقط می‌توان به حلقهٔ while تبدیل کرد.

د – حلقهٔ for را فقط می‌توان به حلقهٔ do..while تبدیل کرد.

14 – اگر به جای شرط کنترل اجرای یک حلقه، عبارت true قرار دهیم آنگاه:

الف – حلقه اصلا اجرا نمی‌شود

ب – حلقه حتما تا بی‌نهایت ادامه می‌یابد

ج – تعداد تکرارها بستگی به دستورات بدنه دارد

د – کامپایلر خطا می‌گیرد

15 – اگر i متغیری از نوع bool با مقدار false باشد آنگاه حلقهٔ مقابل چند بار تکرار می‌شود؟

```
while (!i) i=true;
```

الف – اصلا اجرا نمی‌شود ب – یک بار اجرا می‌شود

ج – تا بی‌نهایت ادامه می‌یابد د – تا وقتی حافظه سرریز شود ادامه می‌یابد

16 – دستور continue در حلقه‌ها چه کاری انجام می‌دهد؟

الف – حلقه را در همان محل خاتمه می‌دهد

ب – مابقی دستورات بدنهٔ حلقه را نادیده گرفته و تکرار بعدی حلقه را آغاز می‌کند

ج – مابقی دستورات بدنهٔ حلقه را نادیده گرفته و حلقه را خاتمه می‌دهد

د – تمام دستورات تکرار فعلی را اجرا نموده و سپس حلقه را خاتمه می‌دهد

17 – در رابطه با بخش initializing یا مقداردهی اولیه در حلقهٔ for کدام عبارت صحیح نیست؟

الف – این بخش فقط یک بار ارزیابی می‌شود

ب – این بخش قبل از این که تکرارها آغاز شوند ارزیابی می‌شود

ج – این بخش می‌تواند در حلقهٔ for قید نشود

د – در حلقه‌های for تودرتو این بخش حذف می‌شود

پرسش‌های تشریحی

1- در یک حلقهٔ while اگر شرط کنترل در ابتدا با مقدار false (یعنی صفر) مقداردهی شود، چه اتفاقی می‌افتد؟

2- چه وقت باید متغیر کنترل در حلقهٔ for قبل از حلقه اعلان گردد (به جای این که داخل بخش کنترلی آن اعلان گردد)؟

3- دستور break چگونه باعث کنترل بهتر روی حلقه‌ها می‌شود؟

4- حداقل تکرار در:

الف - یک حلقهٔ while چقدر است؟

ب - یک حلقهٔ do..while چقدر است؟

5- چه اشتباهی در حلقهٔ زیر است؟

```
while (n <= 100)
 sum += n*n;
```

6- چه خطایی در برنامهٔ زیر است؟

```
int main()
{  const double PI;
   int n;
   PI = 3.14159265358979
   n = 22;
}
```

7- «حلقهٔ بی‌پایان» چیست و چه فایده‌ای دارد؟

8- چطور می‌توان حلقه‌ای ساخت که با یک دستور در وسط بلوکش پایان یابد؟

9- چرا از به‌کارگیری متغیرهای ممیز شناور در مقایسه‌های برابری باید اجتناب شود؟

تمرین‌های برنامه‌نویسی

1- قطعه برنامهٔ زیر را دنبال نمایید و مقدار هر متغیر را در هر گام مشخص کنید:

```
float x = 4.15;
for (int i=0; i < 3; i++)
   x *= 2;
```

2- حلقهٔ for زیر را به حلقهٔ while تبدیل کنید:

```
for (int i=1; i <= n; i++)
 cout << i*i << " ";
```

3- خروجی این برنامه را توضیح دهید:

```
int main()
{  for (int i = 0; i < 8; i++)
      if ( i%2 == 0) cout << i + 1 << "\t";
      else if (i%3 == 0) cout << i*i << "\t";
      else if (i%5 == 0) cout << 2*i - 1 << "\t";
      else cout << i << "\t";
}
```

4- خروجی برنامهٔ زیر را توضیح دهید:

```
int main()
{  for (int i=0; i < 8; i++)
   {  if (i%2 == 0) cout << i + 1 << endl;
      else if (i%3 == 0) continue;
      else if (i%5 == 0) break;
      cout << "End of program.\n";
   }
   cout << "End of program.\n";
}
```

5- برنامه‌ای نوشته و اجرا کنید که عددی را از ورودی گرفته و با استفاده از یک حلقهٔ while مجموع مربعات اعداد متوالی تا آن عدد را پیدا کند. برای مثال اگر 5 وارد شود، برنامه مذکور عدد 55 را چاپ کند که معادل $5^2+4^2+3^2+2^2+1^2$ است.

6- پاسخ سوال 5 را با یک حلقهٔ for نوشته و اجرا کنید.

7- پاسخ سوال 5 را با یک حلقهٔ do..while نوشته و اجرا کنید.
8- برنامه‌ای را نوشته و اجرا کنید که اعمال تقسیم و باقیمانده را بدون استفاده از عملگرهای / و % برای تقسیم اعداد صحیح مثبت پیاده‌سازی می‌کند.
9- برنامه‌ای را نوشته و اجرا کنید که ارقام یک عدد مثبت داده شده را معکوس می‌کند. (به تمرین 13 فصل سوم نگاه کنید)
10- برنامه‌ای بنویسید که ریشهٔ صحیح یک عدد داده شده را پیدا کند. ریشهٔ صحیح، بزرگ‌ترین عدد صحیحی است که مربع آن کوچک‌تر یا مساوی عدد داده شده باشد.
11 – با استفاده از الگوریتم اقلیدس، بزرگ‌ترین مقسوم‌علیه مشترک دو عدد صحیح داده شده را بیابید. این الگوریتم به وسیلهٔ تقسیم‌های متوالی، زوج (m,n) را به زوج (n,0) تبدیل می‌کند. به این صورت که عدد صحیح بزرگ‌تر را بر عدد کوچک‌تر تقسیم کرده و سپس به جای عدد بزرگ‌تر، عدد کوچک‌تر را قرار می‌دهد و به جای عدد کوچک‌تر، باقیمانده تقسیم را قرار می‌دهد و دوباره تقسیم را روی این زوج جدید تکرار می‌کند. وقتی باقیمانده برابر با صفر شود، عدد دیگر از آن زوج، بزرگ‌ترین مقسوم‌علیه مشترک دو عدد صحیح اولیه است (و همچنین بزرگ‌ترین مقسوم‌علیه مشترک تمام زوج‌های میانی). برای مثال اگر m برابر با 532 و n برابر با 112 باشد، الگوریتم اقلیدس زوج (532,112) را به ترتیب زیر به زوج (28,0) تبدیل می‌کند:

(532,112) ⇨ (112,84) ⇨ (84,28) ⇨ (28,0).

برنامه‌ای بنویسید که با استفاده از الگوریتم اقلیدس، بزرگ‌ترین مقسوم‌علیه مشترک دو عدد صحیح داده شده را بیابد.

فصل پنجم

« توابع»

5-1 مقدمه

برنامه‌های واقعی و تجاری بسیار بزرگ‌تر از برنامه‌هایی هستند که تاکنون بررسی کردیم. برای این که برنامه‌های بزرگ قابل مدیریت باشند، برنامه‌نویسان این برنامه‌ها را به زیربرنامه‌هایی بخش‌بندی می‌کنند. این زیربرنامه‌ها «تابع» نامیده می‌شوند. توابع را می‌توان به طور جداگانه کامپایل و آزمایش نمود و در برنامه‌های مختلف دوباره از آن‌ها استفاده کرد. این بخش‌بندی در موفقیت یک نرم‌افزار شی‌گرا بسیار موثر است.

5-2 توابع کتابخانه‌ای C++ استاندارد

«کتابخانهٔ C++ استاندارد» مجموعه‌ای است که شامل توابع از پیش تعریف شده و سایر عناصر برنامه است. این توابع و عناصر از طریق «سرفایل‌ها» قابل دستیابی‌اند. قبلا برخی از آن‌ها را استفاده کرده‌ایم: ثابت INT_MAX که در <climits> تعریف شده (مثال 1-2)، تابع sqrt() که در <cmath> تعریف شده (مثال 15-2)، تابع rand() که در <cstdlib> تعریف شده (مثال 22-4) و تابع time() که در

<ctime> تعریف شده (مثال 24-4). اولین مثال این بخش، استفاده از یک تابع ریاضی را نشان می‌دهد.

× مثال 1-5 تابع جذر ()sqrt

ریشهٔ دوم یک عدد مثبت، جذر آن عدد است. ریشهٔ دوم 9، عدد 3 است. می‌توانیم تابع جذر را به شکل یک جعبهٔ سیاه تصور کنیم که وقتی عدد 9 درون آن قرار گیرد، عدد 3 از آن خارج می‌شود و وقتی عدد 2 در آن قرار گیرد، عدد 1/41421 از آن خارج می‌شود. تابع مانند یک برنامهٔ کامل، دارای روند ورودی - پردازش - خروجی است هرچند که پردازش، مرحله‌ای پنهان است. یعنی نمی‌دانیم که تابع روی عدد 2 چه اعمالی انجام می‌دهد که 1/41421 حاصل می‌شود. تنها چیزی که لازم است بدانیم این است که عدد 1/41421 جذر است و مجذور آن، عدد ورودی 2 بوده است.

برنامهٔ سادهٔ زیر، تابع از پیش تعریف شدهٔ جذر را به کار می‌گیرد:

```
#include <cmath>      // defines the sqrt() function
#include <iostream>   // defines the cout object
using namespace std;
int main()
{ //tests the sqrt() function:
   for (int x=0; x < 6; x++)
      cout << "\t" << x << "\t" << sqrt(x) << endl;
}
```

```
      0       0
      1       1
      2       1.41421
      3       1.73205
      4       2
      5       2.23607
```

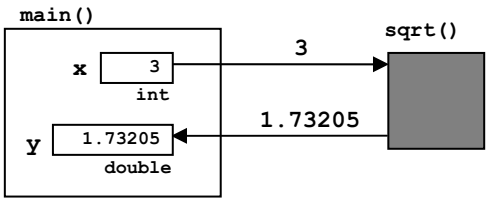
این برنامه، ریشهٔ دوم اعداد صفر تا پنج را چاپ می‌کند. هر وقت اجرای برنامه به عبارت sqrt(x) می‌رسد، تابع sqrt() اجرا می‌گردد. گرچه کد اصلی تابع مذکور درون کتابخانهٔ C++ پنهان شده اما می‌توانیم مطمئن باشیم که به جای عبارت sqrt(x) مقدار جذر x قرار می‌گیرد. به دستور #include <cmath> در اولین

خط برنامه توجه کنید. کامپایلر به این خط نیاز دارد تا بتواند تعریف تابع sqrt() را پیدا کند. این خط به کامپایلر می‌گوید که تعریف تابع جذر در سرفایل <cmath> وجود دارد.

برای اجرای یک تابع مانند تابع sqrt() کافی است نام آن تابع به صورت یک متغیر در دستورالعمل مورد نظر استفاده شود، مانند زیر:

```
y=sqrt(x).
```

این کار «فراخوانی تابع[1]» یا «احضار تابع» گفته می‌شود. بنابراین وقتی کد sqrt(x) اجرا شود، تابع sqrt() فراخوانی می‌گردد. عبارت x درون پرانتز «آرگومان[2]» یا «پارامتر واقعی» فراخوانی نامیده می‌شود. در چنین حالتی می‌گوییم که x توسط «مقدار»[3] به تابع فرستاده می‌شود. لذا وقتی x=3 است، با اجرای کد sqrt(x) تابع sqrt() فراخوانی شده و مقدار 3 به آن فرستاده می‌شود. تابع مذکور نیز حاصل 1.73205 را به عنوان پاسخ برمی‌گرداند. این فرایند در نمودار مقابل نشان داده شده. متغیرهای x و y در تابع main() تعریف شده‌اند. مقدار x که برابر با 3 است به تابع sqrt() فرستاده می‌شود و این تابع مقدار 1.73205 را به تابع main() برمی‌گرداند. جعبه‌ای که تابع sqrt() را نشان می‌دهد به رنگ تیره است، به این معنا که فرایند داخلی و نحوهٔ کار آن قابل رویت نیست.



× مثال 2-5 آزمایش یک رابطهٔ مثلثاتی

این برنامه هم از سرفایل <cmath> استفاده می‌کند. هدف این است که صحت رابطهٔ Sin2x=2SinxCosx به شکل تجربی بررسی شود.

```
int main()
{  // tests the identity sin 2x = 2 sin x cos x:
   for (float x=0; x < 2; x += 0.2)
      cout << x << "\t\t" << sin(2*x) << "\t"
```

1 – Function call 2 – Argument 3 – Pass by value

```
            << 2*sin(x)*cos(x) << endl;
}
```

```
0               0       0
0.2             0.389418        0.389418
0.4             0.717356        0.717356
0.6             0.932039        0.932039
0.8             0.999574        0.999574
1               0.909297        0.909297
1.2             0.675463        0.675463
1.4             0.334988        0.334988
1.6             -0.0583744      -0.0583744
1.8             -0.442521       -0.442521
```

برنامهٔ بالا مقدار x را در ستون اول، مقدار Sin2x را در ستون دوم و مقدار 2SinxCosx را در ستون سوم چاپ می‌کند. خروجی نشان می‌دهد که برای هر مقدار آزمایشی x، مقدار Sin2x با مقدار 2SinxCosx برابر است. البته این نتایج به طور کلی اثبات نمی‌کند که رابطهٔ مذکور صحیح است اما به طور تجربی نشان می‌دهد که این رابطه درست می‌باشد. توجه کنید که x از نوع float تعریف شده است. این امر سبب می‌شود که کد x += 0.2 به درستی کار کند و خطای گردکردن رخ ندهد.

حاصل تابع را می‌توانیم مانند یک متغیر معمولی در هر عبارتی به کار ببریم. یعنی می‌توانیم بنویسیم:

```
y = sqrt(2);
cout << 2*sin(x)*cos(x);
```

همچنین می‌توانیم توابع را به شکل تودرتو فراخوانی کنیم:

```
y = sqrt(1 + 2*sqrt(3 + 4*sqrt(5)))
```

بیشتر توابع معروف ریاضی که در ماشین‌حساب‌ها هم وجود دارد در سرفایل <cmath> تعریف شده است. بعضی از این توابع در جدول زیر نشان داده شده:

بعضی از توابع تعریف شده در سرفایل <cmath>

تابع	شرح	مثال
acos(x)	کسینوس معکوس x (به رادیان)	acos(0.2) مقدار 1.36944 را برمی‌گرداند
asin(x)	سینوس معکوس x (به رادیان)	asin(0.2) مقدار 0.201358 را برمی‌گرداند
atan(x)	تانژانت معکوس x (به رادیان)	atan(0.2) مقدار 0.197396 را برمی‌گرداند

ceil(x)	مقدار سقف x (گرد شده)	ceil(3.141593) مقدار 4.0 را برمی‌گرداند
cos(x)	کسینوس x (به رادیان)	cos(2) مقدار 0.416147- را برمی‌گرداند
exp(x)	تابع نمایی x (در پایه e)	exp(2) مقدار 7.38906 را برمی‌گرداند
fabs(x)	قدر مطلق x	fabs(-2) مقدار 2.0 را برمی‌گرداند
floor(x)	مقدار کف x (گرد شده)	floor(3.141593) مقدار 3.0 را برمی‌گرداند
log(x)	لگاریتم طبیعی x (در پایه e)	log(2) مقدار 0.693147 را برمی‌گرداند
log10(x)	لگاریتم عمومی x (در پایه 10)	log10(2) مقدار 0.30103 را برمی‌گرداند
pow(x,p)	x به توان p	pow(2,3) مقدار 8.0 را برمی‌گرداند
sin(x)	سینوس x (به رادیان)	sin(2) مقدار 0.909297 را برمی‌گرداند
sqrt(x)	جذر x	sqrt(2) مقدار 1.41421 را برمی‌گرداند
tan(x)	تانژانت x (به رادیان)	tan(2) مقدار 2.18504- را برمی‌گرداند

توجه داشته باشید که هر تابع ریاضی یک مقدار از نوع double را برمی‌گرداند. اگر یک نوع صحیح به تابع فرستاده شود، قبل از این که تابع آن را پردازش کند، مقدارش را به نوع double ارتقا می‌دهد.

بعضی از سرفایل‌های کتابخانهٔ C++ استاندارد که کاربرد بیشتری دارند در جدول زیر آمده است:

بعضی از سرفایل‌های کتابخانهٔ C++ استاندارد

سرفایل	شرح
<assert>	تابع <assert> را تعریف می‌کند
<ctype>	توابعی را برای بررسی کاراکترها تعریف می‌کند
<cfloat>	ثابت‌های مربوط به اعداد ممیز شناور را تعریف می‌کند
<climits>	محدودهٔ اعداد صحیح را روی سیستم موجود تعریف می‌کند
<cmath>	توابع ریاضی را تعریف می‌کند
<cstdio>	توابعی را برای ورودی و خروجی استاندارد تعریف می‌کند
<cstdlib>	توابع کاربردی را تعریف می‌کند
<cstring>	توابعی را برای پردازش رشته‌ها تعریف می‌کند
<ctime>	توابع تاریخ و ساعت را تعریف می‌کند

این سرفایل‌ها از کتابخانهٔ C استاندارد گرفته شده‌اند. استفاده از آن‌ها شبیه استفاده از سرفایل‌های C++ استاندارد (مانند <iostream>) است. برای مثال اگر

بخواهیم تابع اعداد تصادفی rand() را از سرفایل <cstdlib> به کار ببریم، باید دستور پیش‌پردازندهٔ زیر را به ابتدای فایل برنامهٔ اصلی اضافه کنیم:

```
#include <cstdlib>
```

5-3 توابع ساخت کاربر

گرچه توابع بسیار متنوعی در کتابخانهٔ C++ استاندارد وجود دارد ولی این توابع برای بیشتر وظایف برنامه‌نویسی کافی نیستند. علاوه بر این برنامه‌نویسان دوست دارند خودشان بتوانند توابعی را بسازند و استفاده نمایند.

× مثال 5-3 تابع cube()

یک مثال ساده از توابع ساخت کاربر:

```
int cube(int x)
{  // returns cube of x:
   return x*x*x;
}
```

این تابع، مکعب یک عدد صحیح ارسالی به آن را برمی‌گرداند. بنابراین فراخوانی cube(2) مقدار 8 را برمی‌گرداند.

یک تابع ساخت کاربر دو قسمت دارد: عنوان و بدنه. عنوان یک تابع به صورت زیر است:

نوع بازگشتی **نام** (*فهرست پارامترها*)

نوع بازگشتی تابع cube() که در بالا تعریف شد، int است. نام آن cube می‌باشد و یک پارامتر از نوع int به نام x دارد. یعنی تابع cube() یک مقدار از نوع int می‌گیرد و پاسخی از نوع int تحویل می‌دهد. پس عنوان تابع فوق عبارت است از:

```
int cube(int x)
```

بدنهٔ تابع، یک بلوک کد است که در ادامهٔ عنوان آن می‌آید. بدنه شامل دستوراتی است که باید انجام شود تا نتیجهٔ مورد نظر به دست آید. بدنه شامل دستور return

است که پاسخ نهایی را به مکان فراخوانی تابع برمی‌گرداند. بدنهٔ تابع cube عبارت است از:

```
{  // returns cube of x:
   return x*x*x;
}
```

این تقریبا ساده‌ترین بدنه‌ای است که یک تابع می‌تواند داشته باشد. توابع مفیدتر معمولا بدنهٔ بزرگ‌تری دارند اما عنوان تابع اغلب روی یک سطر جا می‌شود.

دستور return دو وظیفهٔ عمده دارد. اول این که اجرای تابع را خاتمه می‌دهد و دوم این که مقدار نهایی را به برنامهٔ فراخوان باز می‌گرداند. دستور return به شکل زیر استفاده می‌شود:

return *expression*;

به جای *expression* هر عبارتی قرار می‌گیرد که بتوان مقدار آن را به یک متغیر تخصیص داد. نوع آن عبارت باید با نوع بازگشتی تابع یکی باشد.

عبارت int main() که در همهٔ برنامه‌ها استفاده کرده‌ایم یک تابع به نام «تابع اصلی» را تعریف می‌کند. نوع بازگشتی این تابع از نوع int است. نام آن main است و فهرست پارامترهای آن خالی است؛ یعنی هیچ پارامتری ندارد.

4-5 برنامهٔ آزمون

وقتی یک تابع مورد نیاز را ایجاد کردید، فورا باید آن تابع را با یک برنامهٔ ساده امتحان کنید. چنین برنامه‌ای ***برنامهٔ آزمون***[1] نامیده می‌شود. تنها هدف این برنامه، امتحان کردن تابع و بررسی صحت کار آن است. برنامهٔ آزمون یک برنامهٔ موقتی است که باید «سریع و کثیف» باشد؛ یعنی لازم نیست در آن تمام ظرافت‌های برنامه‌نویسی – مثل پیغام‌های خروجی، برچسب‌ها و راهنماهای خوانا – را لحاظ کنید. وقتی با استفاده از برنامهٔ آزمون، تابع را آزمایش کردید دیگر به آن احتیاجی نیست و می‌توانید برنامهٔ آزمون را دور بریزید.

1 – Testing program

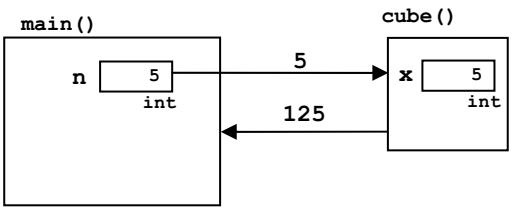
× مثال 4-5 یک برنامهٔ آزمون برای تابع cube()

کد زیر شامل تابع cube() و برنامهٔ آزمون آن است:

```
int cube(int x)
{  // returns cube of x:
   return x*x*x;
}


int main()
{  // tests the cube() function:
   int n=1;
   while (n != 0)
   {  cin >> n;
      cout << "\tcube(" << n << ") = " << cube(n) << endl;
   }
}
```

```
5
        cube(5) = 125
-6
        cube(-6) = -216
0
        cube(0) = 0
```

برنامهٔ بالا اعداد صحیح را از ورودی می‌گیرد و مکعب آن‌ها را چاپ می‌کند تا این که کاربر مقدار 0 را وارد کند. هر عدد صحیحی که خوانده می‌شود، با استفاده از کد cube(n) به تابع cube() فرستاده می‌شود. مقدار بازگشتی از تابع، جایگزین عبارت cube(n) گشته و با استفاده از cout در خروجی چاپ می‌شود.



می‌توان رابطهٔ بین تابع main() و تابع cube() را شبیه این شکل تصور نمود:

تابع main() مقدار 5 را به تابع cube() می‌فرستد و تابع

cube() مقدار 125 را به تابع main() بازمی‌گرداند. آرگومان n به وسیلهٔ مقدار به پارامتر صوری x فرستاده می‌شود. به بیان ساده‌تر وقتی تابع فراخوانی می‌شود، x مقدار n را می‌گیرد.

دقت کنید که تابع cube() در بالای تابع main() تعریف شده زیرا قبل از این که تابع cube() در تابع main() به کار رود، کامپایلر C++ باید در بارهٔ آن اطلاع حاصل کند.

مثال بعدی یک تابع ساخت کاربر به نام max() را نشان می‌دهد که این تابع از دو عدد ارسال شده به آن، عدد بزرگ‌تر را برمی‌گرداند. این تابع دو پارامتر دارد.

× مثال 5-5 یک برنامهٔ آزمون برای تابع max()

تابع زیر دو پارامتر دارد. این تابع از دو مقدار فرستاده شده به آن، مقدار بزرگ‌تر را برمی‌گرداند:

```
int max(int x, int y)
{  // returns larger of the two given integers:
   int z;
   z = (x > y) ? x : y ;
   return z;
}

int main()
{   // tests the max() function:
   int m, n;
   do
   {  cin >> m >> n;
      cout << "\tmax(" << m << "," << n << ") = "
           << max(m,n) << endl;
   }
   while (m != 0);
}
```

```
3 9
      max(3,9) = 9
2 -2
```

```
        max(2,-2) = 2
0 0
        max(0,0) = 0
```

تابع max() یک متغیر محلی به نام z دارد که مقدار بزرگ‌تر در آن نگهداری شده و سپس این مقدار با استفاده از دستور return به تابع main() می‌گردد.

توابع می‌توانند بیش از یک دستور return داشته باشند. مثلا تابع max() را مانند این نیز می‌توانستیم بنویسیم:

```
int max(int x, int y)
{  // returns larger of the two given integers:
   if (x < y) return y;
   else return x;
}
```

در این کد هر دستور return که زودتر اجرا شود مقدار مربوطه‌اش را بازگشت داده و تابع را خاتمه می‌دهد.

دستور return نوعی دستور پرش است (شبیه دستور break) زیرا اجرا را به بیرون از تابع هدایت می‌کند. اگرچه معمولا return در انتهای تابع قرار می‌گیرد، می‌توان آن را در هر نقطهٔ دیگری از تابع قرار داد.

5-5 اعلان‌ها و تعاریف تابع

در دو مثال آخر، تعریف کامل تابع در ابتدای برنامه آمد و زیر آن متن برنامهٔ اصلی قرار گرفت. این یک روش تعریف توابع است که اغلب برای برنامه‌های آزمون از آن استفاده می‌شود. راه دیگری که بیشتر رواج دارد این گونه است که ابتدا تابع ***اعلان***[1] شود، سپس متن برنامهٔ اصلی بیاید، پس از برنامهٔ اصلی ***تعریف***[2] کامل تابع قرار بگیرد. این روش در مثال بعدی نشان داده شده است.

اعلان تابع با تعریف تابع تفاوت دارد. اعلان تابع، فقط عنوان تابع است که یک سمیکولن در انتهای آن قرار دارد ولی تعریف تابع، متن کامل تابع است که هم شامل عنوان است و هم شامل بدنه. اعلان تابع شبیه اعلان متغیرهاست. یک متغیر قبل از این

1 – Declaration 2 - Definition

که به کار گرفته شود باید اعلان شود. تابع هم همین طور است با این فرق که متغیر را در هر جایی از برنامه می‌توان اعلان کرد اما تابع را باید قبل از برنامهٔ اصلی اعلان نمود. در اعلان تابع فقط بیان می‌شود که نوع بازگشتی تابع چیست، نام تابع چیست و نوع پارامترهای تابع چیست. همین‌ها برای کامپایلر کافی است تا بتواند کامپایل برنامه را آغاز کند. بعدا در زمان اجرا به تعریف بدنهٔ تابع نیز احتیاج می‌شود که این بدنه در انتهای برنامه و پس از تابع main() قرار می‌گیرد.

اکنون باید فرق بین «آرگومان[1]» و «پارامتر[2]» را بدانیم. ***پارامترها*** متغیرهایی هستند که در فهرست پارامتر یک تابع نام برده می‌شوند. در مثال قبلی x و y پارامترهای تابع max() هستند. پارامترها متغیرهای محلی برای تابع محسوب می‌شوند؛ یعنی فقط در طول اجرای تابع وجود دارند. ***آرگومان‌ها*** متغیرهایی هستند که از برنامهٔ اصلی به تابع فرستاده می‌شوند. در مثال قبلی m و n آرگومان‌های تابع max() هستند. وقتی یک تابع فراخوانی می‌شود، مقدار آرگومان‌ها درون پارامترهای تابع قرار می‌گیرد تا تابع پردازش را شروع کند. به این ترتیب می‌گوییم که آرگومان‌ها «به روش مقدار» ارسال شده اند. یعنی مقدار آرگومان‌ها جایگزین پارامترهای متناظرشان می‌شوند. در مثال بالا وقتی تابع max() فراخوانی می‌شود، مقدار آرگومان‌های m و n به ترتیب جایگزین پارامترهای x و y می‌شود و سپس تابع کارش را شروع می‌کند. می‌توان به جای آرگومان‌ها، یک مقدار ثابت را به تابع فرستاد (مثل max(22,44)) یا می‌توان یک عبات را به تابع فرستاد (مثل max(2*m, 3-n) که مقدار 2*m در x قرار می‌گیرد و مقدار 3-n در y قرار می‌گیرد).

× مثال 6-5 تابع max() با اعلان جدا از تعریف آن

این برنامه همان برنامهٔ آزمون تابع max() در مثال 5-5 است. اما این‌جا اعلان تابع بالای تابع اصلی ظاهر شده و تعریف تابع بعد از برنامهٔ اصلی آمده است:

```
int max(int,int);
// returns larger of the two given integers:

int main()
```

1 – Argument 2 - Parameter

```
{  // tests the max() function:
   int m, n;
   do
   {  cin >> m >> n;
      cout << "\tmax(" << m << "," << n << ") = "
           << max(m,n) << endl;
   }
   while (m != 0);
}

int max(int x, int y)
{  if (x < y) return y;
   else return x;
}
```

توجه کنید که پارامترهای x و y در بخش عنوان تعریف تابع آمده‌اند (طبق معمول) ولی در اعلان تابع وجود ندارند.

6-5 کامپایل جداگانهٔ توابع

اغلب این طور است که تعریف و بدنهٔ توابع در فایل‌های جداگانه‌ای قرار می‌گیرد. این فایل‌ها به طور مستقل کامپایل[1] می‌شوند و سپس به برنامهٔ اصلی که آن توابع را به کار می‌گیرد الصاق[2] می‌شوند. توابع کتابخانهٔ C++ استاندارد به همین شکل پیاده‌سازی شده‌اند و هنگامی که یکی از آن توابع را در برنامه‌هایتان به کار می‌برید باید با دستور راهنمای پیش‌پردازنده، فایل آن توابع را به برنامه‌تان ضمیمه کنید. این کار چند مزیت دارد. اولین مزیت «مخفی‌سازی اطلاعات» است. یعنی این که توابع لازم را در فایل جداگانه‌ای تعریف و کامپایل کنید و سپس آن فایل را به همراه مشخصات توابع به برنامه‌نویس دیگری بدهید تا برنامهٔ اصلی را تکمیل کند. به این ترتیب آن برنامه‌نویس از جزییات توابع و نحوهٔ اجرای داخلی آن‌ها چیزی نمی‌داند (نباید هم بداند) و فقط می‌داند که چطور می‌تواند از آن‌ها استفاده کند. در نتیجه اطلاعاتی که دانستن آن‌ها برای برنامه‌نویس ضروری نیست از دید او مخفی می‌ماند. تجربه نشان

1 – Compiling 2 - Linking

داده که پنهان‌سازی اطلاعات، فهمیدن برنامهٔ اصلی را آسان می‌کند و پروژه‌های بزرگ با موفقیت اجرا می‌شوند.

مزیت دیگر این است که توابع مورد نیاز را می‌توان قبل از این که برنامهٔ اصلی نوشته شود، جداگانه آزمایش نمود. وقتی یقین کردید که یک تابع مفروض به درستی کار می‌کند، آن را در یک فایل ذخیره کنید و جزییات آن تابع را فراموش کنید و هر وقت که به آن تابع نیاز داشتید با خیالی راحت از آن در برنامه‌هایتان استفاده نمایید. نتیجه این است که تولید توابع مورد نیاز و تولید برنامهٔ اصلی، هم‌زمان و مستقل از هم پیش می‌رود بدون این که یکی منتظر دیگری بماند. به این دیدگاه «بسته‌بندی نرم‌افزار» می‌گویند.

سومین مزیت این است که در هر زمانی به راحتی می‌توان تعریف توابع را عوض کرد بدون این که لازم باشد برنامهٔ اصلی تغییر یابد. فرض کنید تابعی برای مرتب کردن فهرستی از اعداد ایجاد کرده‌اید و آن را جداگانه کامپایل و ذخیره نموده‌اید و در یک برنامهٔ کاربردی هم از آن استفاده برده‌اید. حالا هرگاه که الگوریتم سریع‌تری برای مرتب‌سازی یافتید، فقط کافی است فایل تابع را اصلاح و کامپایل کنید و دیگر نیازی نیست که برنامهٔ اصلی را دست‌کاری نمایید.

چهارمین مزیت هم این است که می‌توانید یک بار یک تابع را کامپایل و ذخیره کنید و از آن پس در برنامه‌های مختلفی از همان تابع استفاده ببرید. وقتی شروع به نوشتن یک برنامهٔ جدید می‌کنید، شاید برخی از توابع مورد نیاز را از قبل داشته باشید. بنابراین دیگر لازم نیست که آن توابع را دوباره نوشته و کامپایل کنید. این کار سرعت تولید نرم افزار را افزایش می‌دهد.

تابع `max()` را به خاطر بیاورید. برای این که این تابع را در فایل جداگانه‌ای قرار دهیم، تعریف آن را در فایلی به نام `max.cpp` ذخیره می‌کنیم. فایل `max.cpp` شامل کد مقابل است:

max.cpp

```
int max(int x, int y)
{  if (x < y) return y;
   else return x;
}
```

حالا سراغ برنامهٔ اصلی می‌رویم. متن برنامهٔ اصلی را در فایلی به نام test.cpp ذخیره می‌نماییم. این فایل شامل کد زیر است:

test.cpp

```
int max(int,int);
// returns larger of the two given integers:

int main()
{  // tests the max() function:
   int m, n;
   do
   {  cin >> m >> n;
      cout << "\tmax(" << m << "," << n << ") = "
           << max(m,n) << endl;
  }
   while (m != 0);
}
```

در برنامهٔ اصلی، تابع max() فقط اعلان شده است. در اولین خط از برنامهٔ اصلی اعلان شده که تابع max() دو پارامتر از نوع int دارد و یک مقدار از نوع int برمی‌گرداند.

نحوهٔ کامپایل کردن فایل‌ها و الصاق آن‌ها به یکدیگر به نوع سیستم عامل و نوع کامپایلر بستگی دارد. در سیستم عامل ویندوز معمولا توابع را در فایل‌هایی از نوع DLL[1] کامپایل و ذخیره می‌کنند و سپس این فایل را در برنامهٔ اصلی احضار می‌نمایند. فایل‌های DLL را به دو طریق ایستا و پویا می‌توان مورد استفاده قرار داد. برای آشنایی بیشتر با فایل‌های DLL به مرجع ویندوز و کامپایلرهای c++ مراجعه کنید.

6-5 متغیرهای محلی، توابع محلی

متغیر محلی[2]، متغیری است که در داخل یک بلوک اعلان گردد. این گونه متغیرها فقط در داخل همان بلوکی که اعلان می‌شوند قابل دستیابی هستند. چون بدنهٔ تابع، خودش یک بلوک است پس متغیرهای اعلان شده در یک تابع متغیرهای محلی

1 – Data Link Library

1 – Local variable

برای آن تابع هستند. این متغیرها فقط تا وقتی که تابع در حال کار است وجود دارند. پارامترهای تابع نیز متغیرهای محلی محسوب می‌شوند.

× مثال 7-5 تابع فاکتوریل

اعداد فاکتوریل را در مثال 8-4 دیدیم. فاکتوریل عدد صحیح n برابر است با:

```
n! = n(n-1)(n-2)..(3)(2)(1)
```

تابع زیر، فاکتوریل عدد n را محاسبه می‌کند:

```
long fact(int n)
{  //returns n! = n*(n-1)*(n-2)*...*(2)*(1)
   if (n < 0) return 0;
   int f = 1;
   while (n > 1)
      f *= n--;
   return f;
}
```

این تابع دو متغیر محلی دارد: n و f. پارامتر n یک متغیر محلی است زیرا در فهرست پارامترهای تابع اعلان شده. متغیر f نیز محلی است زیرا درون بدنهٔ تابع اعلان شده. تابع فاکتوریل را با استفاده از برنامهٔ آزمون زیر می‌توان آزمایش کرد:

```
long fact(int);
// returns n! = n*(n-1)*(n-2)*...*(2)*(1)

int main()
{  // tests the factorial() function:
   for (int i=-1; i < 6; i++)
      cout << " " << fact(i);
   cout << endl;
}
```

```
0 1 1 2 6 24 120
```

برای این که برنامهٔ بالا قابل اجرا باشد، یا باید فایل تابع فاکتوریل را به آن الصاق کنیم و یا این که تعریف تابع فاکتوریل را به انتهای آن اضافه نماییم.

همان گونه که متغیرها می‌توانند محلی باشند، توابع نیز می‌توانند محلی باشند. یک ***تابع محلی***[1] تابعی است که درون یک تابع دیگر به کار رود. با استفاده از چند تابع ساده و ترکیب آن‌ها می‌توان توابع پیچیده‌تری ساخت. به مثال زیر نگاه کنید.

× مثال 8-5 تابع جایگشت

در ریاضیات، تابع جایگشت را با `p(n,k)` نشان می‌دهند. این تابع بیان می‌کند که به چند طریق می‌توان `k` عنصر دلخواه از یک مجموعهٔ `n` عنصری را کنار یکدیگر قرار داد. برای این محاسبه از رابطهٔ زیر استفاده می‌شود:

$$P(n,k)=\frac{n!}{(n-k)!}$$

برای مثال:

$$P(4,2)=\frac{4!}{(4-2)!}=\frac{4!}{2!}=\frac{24}{2}=12$$

پس به 12 طریق می‌توانیم دو عنصر دلخواه از یک مجموعهٔ چهار عنصری را کنار هم بچینیم. برای دو عنصر از مجموعهٔ `{1, 2, 3, 4}` حالت‌های ممکن عبارت است از:

```
12, 13, 14, 21, 23, 24, 31, 32, 34, 41, 42, 43
```

کد زیر تابع جایگشت را پیاده‌سازی می‌کند:

```
long perm(int n, int k)
{  // returns P(n,k), the number of the permutations of k from n:
   if (n < 0) || k < 0 || k > n) return 0;
   return fact(n)/fact(n-k);
}
```

این تابع، خود از تابع دیگری که همان تابع فاکتوریل است استفاده کرده. شرط به کار رفته در دستور `if` برای محدود کردن حالت‌های غیر ممکن استفاده شده است. در این حالت‌ها، تابع مقدار 0 را برمی‌گرداند تا نشان دهد که یک ورودی اشتباه وجود داشته است. برنامهٔ آزمون برای تابع `perm()` در ادامه آمده است:

---

1 – Local function

```
long perm(int,int);
// returns P(n,k), the number of permutations of k from n:

int main()
{  // tests the perm() function:
   for (int i = -1; i < 8; i++)
   {  for (int j= -1; j <= i+1; j++)
         cout << " " << perm(i,j);
      cout << endl;
  }
}
```

```
0 0
0 1 0
0 1 1 0
0 1 2 2 0
0 1 3 6 6 0
0 1 4 12 24 24 0
0 1 5 20 60 120 120 0
0 1 6 30 120 360 720 720 0
0 1 7 42 210 840 2520 5040 5040 0
```

البته ضروری است که تعریف دو تابع perm() و fact() در یک فایل باشد.

5-7 تابع void

لازم نیست یک تابع حتما مقداری را برگرداند. در C++ برای مشخص کردن چنین توابعی از کلمهٔ کلیدی void به عنوان نوع بازگشتی تابع استفاده می‌کنند. یک تابع void تابعی است که هیچ مقدار بازگشتی ندارد.

× مثال 5-9 تابعی که به جای شمارهٔ ماه‌ها ، نام آن‌ها را می‌نویسد

```
void PrintDate(int,int,int);
// prints the given date in literal form:

int main()
{  // tests the PrintDate() function:
   int day, month, year;
   do
```

```
   {  cin >> day >> month >> year;
      PrintDate(day,month,year);
   }
   while (month > 0);
}

void PrintDate(int d, int m, int y)
{  // prints the given date in literal form:
 if (d < 1 || d > 31 || m < 1 || m > 12 || y < 0)
   {  cout << "Error: parameter out of range.\n";
      return;
    }
   Cout << d;
   switch (m)
   {  case  1:  cout << "Farvardin ";   break;
      case  2:  cout << "Ordibehesht "; break;
      case  3:  cout << "Khordad ";     break;
      case  4:  cout << "Tir ";         break;
      case  5:  cout << "Mordad ";      break;
      case  6:  cout << "Shahrivar ";   break;
      case  7:  cout << "Mehr ";        break;
      case  8:  cout << "Aban ";        break;
      case  9:  cout << "Azar ";       break;
      case 10:  cout << "Dey ";        break;
      case 11:  cout << "Bahman ";     break;
      case 12:  cout << "Esfnad ";     break;
   }
   cout << y << endl;
}
```

```
7 12 1383
7 Esfand 1383
15 8 1384
15 Aban 1384
0 0 0
Error: parameter out of range.
```

تابع PrintDate() هیچ مقداری را برنمی‌گرداند. تنها هدف این تابع، چاپ تاریخ است. بنابراین نوع بازگشتی آن void است. اگر پارامترها خارج از محدوده باشند، تابع بدون این که چیزی چاپ کند خاتمه می‌یابد. با این وجود باز هم احتمال دارد مقادیر غیرممکنی مانند 31 Esfand 1384 چاپ شوند. اصلاح این ناهنجاری را به عنوان تمرین به دانشجو وا می‌گذاریم.

از آن‌جا که یک تابع void مقداری را برنمی‌گرداند، نیازی به دستور return نیست ولی اگر قرار باشد این دستور را در تابع void قرار دهیم، باید آن را به شکل تنها استفاده کنیم بدون این که بعد از کلمهٔ return هیچ چیز دیگری بیاید:

```
return;
```

در این حالت دستور return فقط تابع را خاتمه می‌دهد.

توابع void معمولا برای انجام یک کار مشخص استفاده می‌شوند مثل تابع بالا که تاریخ عددی را گرفته و شکل حرفی آن را چاپ می‌کند. به همین دلیل برنامه‌نویسان معمولا اسم این توابع را به شکل یک گزارهٔ فعلی انتخاب می‌کنند. مثلا نام تابع فوق PrintDate است که نشان می‌دهد این تابع کار چاپ تاریخ را انجام می‌دهد. رعایت این قرارداد به خوانایی و درک بهتر برنامه‌تان کمک می‌کند.

5-8 توابع بولی

در بسیاری از اوقات لازم است در برنامه، شرطی بررسی شود. اگر بررسی این شرط به دستورات زیادی نیاز داشته باشد، بهتر است که یک تابع این بررسی را انجام دهد. این کار مخصوصا هنگامی که از حلقه‌ها استفاده می‌شود بسیار مفید است. توابع بولی فقط دو مقدار را برمی‌گردانند: true یا false .

× مثال 10-5 تابعی که اول بودن اعداد را بررسی می‌کند

کد زیر یک تابع بولی است که تشخیص می‌دهد آیا عدد صحیح ارسال شده به آن، اول است یا خیر:

```
bool isPrime(int n)
{  // returns true if n is prime, false otherwise:
```

```
   float sqrtn = sqrt(n);
   if (n < 2) return false;       // 0 and 1 are not primes
   if (n < 4) return true;        // 2 and 3 are the first primes
   if (n%2 == 0) return false;    // 2 is the only even prime
   for (int d=3; d <= sqrtn; d += 2)
      if (n%d == 0) return false; // n has a nontrivial divisor
   return true;                   // n has no nontrivial divisors
}
```

تابع فوق برای عدد n به دنبال یک مقسوم‌علیه می‌گردد. اگر پیدا شد مقدار false را برمی‌گرداند یعنی n اول نیست. اگر هیچ مقسوم‌علیهی یافت نشد مقدار true را برمی‌گرداند که یعنی n اول است. این تابع برای یافتن مقسوم‌علیه فرض‌های زیر را در نظر می‌گیرد: 1 – اعداد کوچک‌تر از دو اول نیستند. 2- عدد دو اول است. 3 – هر عدد زوج غیر از دو اول نیست. 4 – حداقل یکی از مقسوم‌علیه‌های عدد از جذر آن عدد کوچک‌تر است. این فرض‌ها یکی یکی بررسی می‌شوند. اگر n از دو کوچک‌تر باشد مقدار false برمی‌گردد. اگر n برابر با 2 یا 3 باشد مقدار true برمی‌گردد یعنی n اول است. در غیر این صورت اگر n زوج باشد باز هم false برمی‌گردد زیرا هیچ عدد زوجی غیر از دو اول نیست. اگر n زوج هم نبود آنگاه حلقهٔ for شروع می‌شود و در این حلقه بررسی می‌شود که آیا عددی هست که از جذر n کوچک‌تر بوده و مقسوم‌علیه n باشد یا خیر. دقت کنید که در حلقهٔ for فقط کافی است اعداد فرد کوچک‌تر از جذر n را بررسی کنیم (چرا؟)

یک برنامهٔ آزمون و خروجی آن در ادامه آمده است:

```
#include <cmath>      // defines the sqrt() function
#include <iostream>   // defines the cout object
using namespace std;

bool isPrime(int);
// returns true if n is prime, false otherwise;
int main()
{  for (int n=0; n < 80; n++)
      if (isPrime(n)) cout << n << " ";
   cout << endl;
}
```

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
```

بهتر است بدانید که این تابع، بهینه نیست. هر عدد مرکب(غیر اول) را می‌توان به صورت ضرب یک عدد اول در عدد دیگری نوشت. به همین دلیل برای تشخیص اول بودن یک عدد کافی است بررسی شود که آیا این عدد به اعداد اول قبل از خودش قابل تقسیم است یا خیر. برای این منظور هم باید تمامی اعداد اول یافته شده را در آرایه‌ای ذخیره کنیم. آرایه‌ها را در فصل بعدی بررسی می‌کنیم.

اسم توابع بولی را معمولا به شکل سوالی انتخاب می‌کنند زیرا توابع بولی همیشه به یک سوال مفروض پاسخ بلی یا خیر می‌دهند. تابعی که در مثال بالا توضیح داده شد `isPrime` نام گرفته زیرا پاسخ می‌دهد که آیا عدد مذکور اول است یا خیر. این نحو نام‌گذاری گرچه اجباری نیست اما درک برنامه را آسان‌تر می‌کند و به یادآوری وظیفهٔ تابع نیز کمک می‌نماید. در کتابخانهٔ C++ استاندارد توابع بولی مثل `isLower()` یا `isUpper()` به همین شیوه نام‌گذاری شده‌اند.

5-9 توابع ورودی/خروجی[1] (I/O)

بخش‌هایی از برنامه که به جزییات دست و پا گیر می‌پردازد و خیلی به هدف اصلی برنامه مربوط نیست را می‌توان به توابع سپرد. در چنین شرایطی سودمندی توابع محسوس‌تر می‌شود. فرض کنید نرم‌افزاری برای سیستم آموزشی دانشگاه طراحی کرده‌اید که سوابق تحصیلی دانشجویان را نگه می‌دارد. در این نرم‌افزار لازم است که سن دانشجو به عنوان یکی از اطلاعات پروندهٔ دانشجو وارد شود. اگر وظیفهٔ دریافت سن را به عهدهٔ یک تابع بگذارید، می‌توانید جزییاتی از قبیل کنترل ورودی معتبر، یافتن سن از روی تاریخ تولد و ... را در این تابع پیاده‌سازی کنید بدون این که از مسیر برنامهٔ اصلی منحرف شوید.

قبلا نمونه‌ای از توابع خروجی را دیدیم. تابع `PrintDate()` در مثال 9-5 هیچ چیزی به برنامهٔ اصلی برنمی‌گرداند و فقط برای چاپ نتایج به کار می‌رود. این تابع نمونه‌ای از توابع خروجی است؛ یعنی توابعی که فقط برای چاپ نتایج به کار می‌روند و هیچ مقدار بازگشتی ندارند. توابع ورودی نیز به همین روش کار می‌کنند اما

1 – Input/Output functions

در جهت معکوس. یعنی توابع ورودی فقط برای دریافت ورودی و ارسال آن به برنامهٔ اصلی به کار می‌روند و هیچ پارامتری ندارند. مثال بعد یک تابع ورودی را نشان می‌دهد.

× مثال 11-5 تابعی برای دریافت سن کاربر

تابع سادهٔ زیر، سن کاربر را درخواست می‌کند و مقدار دریافت شده را به برنامهٔ اصلی می‌فرستد. این تابع تقریبا هوشمند است و هر عدد صحیح ورودی غیر منطقی را رد می‌کند و به طور مکرر درخواست ورودی معتبر می‌کند تا این که یک عدد صحیح در محدودهٔ 7 تا 120 دریافت دارد:

```
int age()
{  // prompts the user to input his/her age and returns that value:
   int n;
   while (true)
   {  cout << "How old are you: ";
      cin >> n;
      if (n < 0) cout << "\a\tYour age could not
                         be negative.";
      else if (n > 120) cout << "\a\tYou could not
                         be over 120.";
      else return n;
      cout << "\n\tTry again.\n";
  }
}
```

شرط کنترل حلقه، true است و این حلقه به ظاهر بی‌پایان به نظر می‌رسد. اما دستور return درون حلقه نه تنها مقدار ورودی معتبر را به برنامهٔ اصلی می‌فرستد بلکه هم حلقه را خاتمه می‌دهد و هم تابع را. به محض این که ورودی دریافت شده از cin معتبر باشد، دستور return اجرا شده و مقدار مذکور به برنامهٔ اصلی ارسال می‌شود و تابع خاتمه می‌یابد. اگر ورودی قابل قبول نباشد (n<7 یا n>120) آنگاه یک بوق اخطار پخش می‌شود (که این بوق حاصل چاپ کاراکتر \a است) و سپس یک توضیح روی صفحه‌نمایش درج می‌شود که کاربر می خواهد دوباره تلاش کند.

توجه کنید که این مثالی است که دستور return در انتهای تابع قرار نگرفته. علاوه بر این فهرست پارامترهای تابع خالی است زیرا از برنامهٔ اصلی چیزی دریافت نمی‌کند و فقط یک عدد صحیح را به برنامهٔ اصلی برمی‌گرداند. با این وجود لازم است که پرانتز هم در اعلان تابع و هم در فراخوانی تابع قید شود.

یک برنامهٔ آزمون و خروجی حاصل از آن در ادامه آمده است:

```
int age()

int main()
{  // tests the age() function:
   int a = age();
   cout << "\nYou are " << a << " years old.\n";
}
```

```
How old are you? 125
  You could not be over 120
  Try again.
How old are you? -3
  Your age could not be negative
  Try again.
How old are you? 99

You are 99 years old.
```

14-5 ارسال به طریق ارجاع[1] (آدرس)

تا این لحظه تمام پارامترهایی که در توابع دیدیم به طریق ***مقدار*** ارسال شده‌اند. یعنی ابتدا مقدار متغیری که در فراخوانی تابع ذکر شده برآورد می‌شود و سپس این مقدار به پارامترهای محلی تابع فرستاده می‌شود. مثلا در فراخوانی cube(x) ابتدا مقدار x برآورد شده و سپس این مقدار به متغیر محلی n در تابع فرستاده می‌شود و پس از آن تابع کار خویش را آغاز می‌کند. در طی اجرای تابع ممکن است مقدار n تغییر کند اما چون n محلی است هیچ تغییری روی مقدار x نمی‌گذارد. پس خود x به تابع نمی‌رود بلکه مقدار آن درون تابع کپی می‌شود. تغییر دادن این مقدار کپی شده درون تابع هیچ تاثیری بر x اصلی ندارد. به این ترتیب تابع می‌تواند مقدار x را بخواند

1 – Reference

اما نمی‌تواند مقدار x را تغییر دهد. به همین دلیل به x یک پارامتر «فقط خواندنی» می‌گویند. وقتی ارسال به وسیلهٔ مقدار باشد، هنگام فراخوانی تابع می‌توان از عبارات استفاده کرد. مثلا تابع cube() را می‌توان به صورت cube(2*x-3) فراخوانی کرد یا به شکل cube(2*sqrt(x)-cube(3)) فراخوانی نمود. در هر یک از این حالات، عبارت درون پرانتز به شکل یک مقدار تکی برآورد شده و حاصل آن مقدار به تابع فرستاده می‌شود.

ارسال به طریق مقدار باعث می‌شود که متغیرهای برنامهٔ اصلی از تغییرات ناخواسته در توابع مصون بمانند. اما گاهی اوقات عمدا می‌خواهیم این اتفاق رخ دهد. یعنی می‌خواهیم که تابع بتواند محتویات متغیر فرستاده شده به آن را دست‌کاری کند. در این حالت از ***ارسال به طریق ارجاع*** استفاده می‌کنیم.

برای این که مشخص کنیم یک پارامتر به طریق ارجاع ارسال می‌شود، علامت **&** را به نوع پارامتر در فهرست پارامترهای تابع اضافه می‌کنیم. این باعث می‌شود که تابع به جای این که یک کپی محلی از آن آرگومان ایجاد کند، خود آرگومان محلی را به کار بگیرد. به این ترتیب تابع هم می‌تواند مقدار آرگومان فرستاده شده را بخواند و هم می‌تواند مقدار آن را تغییر دهد. در این حالت آن پارامتر یک پارامتر «خواندنی–نوشتنی» خواهد بود. هر تغییری که روی پارامتر خواندنی–نوشتنی در تابع صورت گیرد به طور مستقیم روی متغیر برنامهٔ اصلی اعمال می‌شود. به مثال زیر نگاه کنید.

× مثال 12-5 تابع swap()

تابع کوچک زیر در مرتب کردن داده‌ها کاربرد فراوان دارد:

```
void swap(float& x, float& y)
{  // exchanges the values of x and y:
   float temp = x;
   x = y;
   y = temp;
}
```

هدف این تابع جابجا کردن دو عنصری است که به آن فرستاده می‌شوند. برای این منظور پارامترهای x و y به صورت پارامترهای ارجاع تعریف شده‌اند:

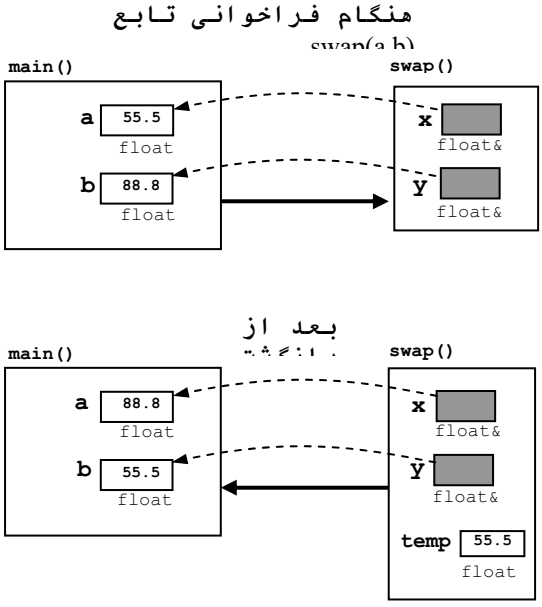
float& x, float& y

عملگر ارجاع **&** موجب می‌شود که به جای x و y آرگومان‌های ارسالی قرار بگیرند. برنامهٔ آزمون و اجرای آزمایشی آن در زیر آمده است:

```
void swap(float&, float&)
// exchanges the values of x and y:

int main()
{  // tests the swap() function:
   float a = 55.5, b = 88.8;
   cout << "a = " << a << ", b = " << b << endl;
   swap(a,b);
   cout << "a = " << a << ", b = " << b << endl;
}
```

```
a = 55.5, b = 88.8
a = 88.8, b = 55.5
```

وقتی فراخوانی swap(a,b) اجرا می‌شود، x به a اشاره می‌کند و y به b. سپس متغیر محلی temp اعلان می‌شود و مقدار x (که همان a است) درون آن قرار می‌گیرد. پس از آن مقدار y (که همان b است) درون x (یعنی a) قرار می‌گیرد و آنگاه مقدار temp درون y (یعنی b) قرار داده می‌شود. نتیجهٔ نهایی این است که مقادیر a و b با یکدیگر جابجا می شوند. شکل مقابل نشان می‌دهد که چطور این جابجایی رخ می‌دهد:



به اعلان تابع swap() دقت کنید:

```
void swap(float&, float&)
```

این اعلان شامل عملگر ارجاع & برای هر پارامتر است. برنامه‌نویسان C عادت دارند که عملگر ارجاع & را به عنوان پیشوند نام متغیر استفاده کنند (مثل float &x) در C++ فرض می‌کنیم عملگر ارجاع & پسوند نوع است (مثل float& x) به هر حال کامپایلر هیچ فرقی بین این دو اعلان نمی‌گذارد و شکل نوشتن عملگر ارجاع کاملا اختیاری و سلیقه‌ای است.

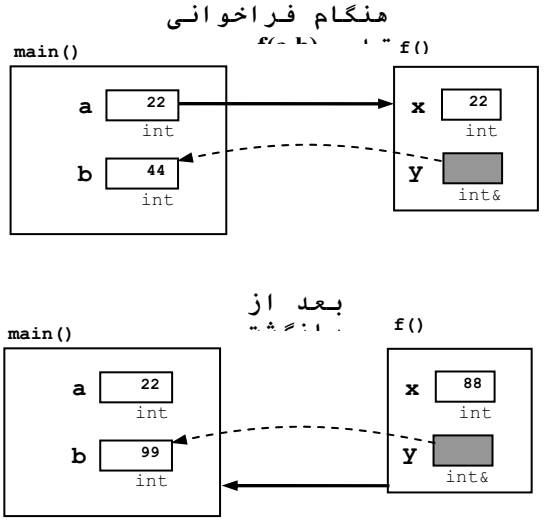
× مثال 13-5 ارسال به طریق مقدار و ارسال به طریق ارجاع

این برنامه، تفاوت بین ارسال به طریق مقدار و ارسال به طریق ارجاع را نشان می‌دهد:

```
void f(int,int&);
// changes reference argument to 99:

int main()
{  // tests the f() function:
   int a = 22, b = 44;
   cout << "a = " << a << ", b = " << b << endl;
   f(a,b);
   cout << "a = " << a << ", b = " << b << endl;
   f(2*a-3,b);
   cout << "a = " << a << ", b = " << b << endl;
}

void f(int x , int& y)
{  // changes reference argument to 99:
   x = 88;
   y = 99;
}
```

```
a = 22, b = 44
a = 22, b = 99
a = 22, b = 99
```

تابع f() دو پارامتر دارد که اولی به طریق مقدار و دومی به طریق ارجاع ارسال می‌شود. فراخوانی f(a,b) باعث می‌شود که a از طریق مقدار به x ارسال شود و b از طریق ارجاع به y فرستاده شود. بنابراین x یک متغیر محلی است که مقدار 22 به آن فرستاده می‌شود در حالی که y یک ارجاع به متغیر b است که مقدار فعلی آن 44 می‌باشد. در تابع f() مقدار 88 در x قرار می‌گیرد که این تاثیری بر a ندارد. همچنین مقدار 99 در y قرار می‌گیرد که چون y در حقیقت یک نام مستعار برای b است، مقدار b به 99 تغییر می‌کند. هنگامی که تابع خاتمه یابد، a هنوز مقدار 22 را دارد ولی مقدار b به 99 تغییر یافته است. آرگومان a فقط خواندنی است و آرگومان b خواندنی-نوشتنی است. شکل زیر نحوهٔ کار تابع f() را نشان می‌دهد.



در جدول زیر خلاصهٔ تفاوت‌های بین ارسال از طریق مقدار و ارسال از طریق ارجاع آمده است.

ارسال از طریق مقدار در مقایسه با ارسال از طریق ارجاع

ارسال از طریق مقدار	ارسال از طریق ارجاع
int x;	int& x;
پارامتر x یک متغیر محلی است	پارامتر x یک ارجاع است
x یک کپی از آرگومان است	x مترادف با آرگومان است
تغییر محتویات آرگومان ممکن نیست	می‌تواند محتویات آرگومان را تغییر دهد

آرگومان ارسال شده از طریق مقدار می‌تواند یک ثابت، یک متغیر یا یک عبارت باشد	آرگومان ارسال شده از طریق ارجاع فقط باید یک متغیر باشد
آرگومان فقط خواندنی است	آرگومان خواندنی-نوشتنی است

یکی از مواقعی که پارامترهای ارجاع مورد نیاز هستند جایی است که تابع باید بیش از یک مقدار را بازگرداند. دستور return فقط می‌تواند یک مقدار را برگرداند. بنابراین اگر باید بیش از یک مقدار برگشت داده شود، این کار را پارامترهای ارجاع انجام می‌دهند.

× مثال 14-5 بازگشت بیشتر از یک مقدار

تابع زیر از طریق دو پارامتر راجاع، دو مقدار را بازمی‌گرداند: area و circumference (محیط و مساحت) برای دایره‌ای که شعاع آن عدد مفروض r است:

```
void ComputeCircle(double& area, double& circumference, double r)
{  // returns the area and circumference of a circle with radius r:
   const double PI = 3.141592653589793;
   area = PI*r*r;
   circumference = 2*PI*r;
}
```

برنامهٔ آزمون تابع فوق و یک اجرای آزمایشی آن در شکل زیر نشان داده شده است:

```
void ComputerCircle(double&, double&, double);
// returns the area and  circumference of a circle with radius r;

int main()
{  // tests the ComputeCircle() function:
   double r, a, c;
   cout << "Enter radius: ";
   cin >> r;
   ComputeCircle(a, c, r);
   cout << "area = " << a << ", circumference = "
        << c << endl;
}
```

```
Enter radius: 100
area = 31415.9, circumference = 628.319
```

در اعلان و تعریف تابع فوق، پارامترهایی که از طریق ارجاع ارسال می‌شوند در ابتدای فهرست پارامترها قرار داده شده‌اند. رعایت این قاعده باعث می‌شود که نظم برنامه حفظ شود و به سادگی بتوانید پارامترهای تابع را از یکدیگر تمیز دهید. البته این فقط یک قرارداد است و رعایت آن اجباری نیست.

11-5 ارسال از طریق ارجاع ثابت

ارسال پارامترها به طریق ارجاع دو خاصیت مهم دارد: اول این که تابع می‌تواند روی آرگومان واقعی تغییراتی بدهد و دوم این که از اشغال بی‌مورد حافظه جلوگیری می‌شود. وقتی یک آرگومان از طریق مقدار به تابع فرستاده شود، یک کپی محلی از آن آرگومان ایجاد شده و در اختیار تابع قرار می‌گیرد. این کپی به اندازهٔ آرگومان اصلی حافظه اشغال می‌کند. حال اگر آرگومان اصلی خیلی حجیم باشد (مثل یک تصویر گرافیکی) آنگاه ارسال از طریق مقدار باعث می‌شود که حافظه به میزان دوبرابر مصرف شود؛ بخشی برای آرگومان اصلی و بخشی دیگر برای نسخه محلی که در تابع به کار می‌رود. حال اگر این شیء حجیم را از طریق ارجاع به تابع ارسال کنیم دیگر نسخهٔ محلی ساخته نمی‌شود و حافظه‌ای هم هدر نمی‌رود. اما این کار یک عیب بزرگ دارد: تابع می‌تواند مقدار پارامتر ارجاع را دست‌کاری کند. اگر تابع نمی‌بایست پارامتر مذکور را دست‌کاری کند، آنگاه ارسال از طریق ارجاع مخاطره‌آمیز خواهد بود.

برای این که عیب مذکور برطرف شود و شیء اصلی از تغییرات ناخواسته درون تابع مصون باشد، C++ روش سومی را برای ارسال آرگومان پیشنهاد می‌کند: ارسال از طریق ***ارجاع ثابت***[1]. این روش مانند ارسال از طریق ارجاع است با این فرق که تابع نمی‌تواند محتویات پارامتر ارجاع را دست‌کاری نماید و فقط اجازهٔ خواندن آن را دارد. برای این که پارامتری را از نوع ارجاع ثابت اعلان کنیم باید عبارت `const` را به ابتدای اعلان آن اضافه نماییم.

---

1 – Constant reference

× مثال 15-5 ارسال از طریق ارجاع ثابت

سه طریقه ارسال پارامتر در تابع زیر به کار رفته است:

```
void f(int x, int& y, const int& z)
{  x += z;
   y += z;
   cout << "x = " << x << ", y = " << y << ", z = "
        << z << endl;
}
```

در تابع فوق اولین پارامتر یعنی x از طریق مقدار ارسال می‌شود، دومین پارامتر یعنی y از طریق ارجاع و سومین پارامتر نیز از طریق ارجاع ثابت. برنامهٔ آزمون و یک اجرای آزمایشی از آن در ذیل آمده است:

```
void f(int, int&, const int&);
int main()
{  // tests the f() function:
   int a = 22, b = 33, c = 44;
   cout << "a = " << a << ", b = " << b << ", c = "
        << c << endl;
   f(a,b,c);
   cout << "a = " << a << ", b = " << b << ", c = "
        << c << endl;
   f(2*a-3,b,c);
   cout << "a = " << a << ", b = " << b << ", c = "
        << c << endl;
}
```

```
a = 22, b = 33, c = 44
x = 66, y = 77, z = 44
a = 22, b = 77, c = 44
x = 85, y = 121, z = 44
a = 22, b = 121, c = 44
```

تابع فوق پارامترهای x و y را می‌تواند تغییر دهد ولی قادر نیست پارامتر z را تغییر دهد. تغییراتی که روی x صورت می‌گیرد اثری روی آرگومان a نخواهد داشت زیرا a از طریق مقدار به تابع ارسال شده. تغییراتی که روی y صورت می‌گیرد روی آرگومان b هم تاثیر می‌گذارد زیرا b از طریق ارجاع به تابع فرستاده شده.

ارسال به طریق ارجاع ثابت بیشتر برای توابعی استفاده می‌شود که عناصر بزرگ را ویرایش می‌کنند مثل آرایه‌ها یا نمونهٔ کلاس‌ها که در فصل‌های بعدی توضیح آن‌ها آمده است. عناصری که از انواع اصلی هستند (مثل int یا float) به طریق مقدار ارسال می‌شوند به شرطی که قرار نباشد تابع محتویات آن‌ها را دست‌کاری کند.

5-12 توابع بی‌واسطه[1]

وقتی تابعی درون یک برنامه فراخوانی می‌شود، ابتدا باید مکان فعلی اجرای برنامهٔ اصلی و متغیرهای فعلی آن در جایی نگهداری شود تا پس از اتمام تابع، ادامه برنامه پی‌گیری شود. همچنین باید متغیرهای محلی تابع ایجاد شوند و حافظه‌ای برای آن‌ها تخصیص یابد و همچنین آرگومان‌ها به این متغیرها ارسال شوند تا درنهایت تابع شروع به کار کند. پس از پایان کار تابع نیز باید همین مسیر به شکل معکوس پیموده شود تا برنامهٔ اصلی ادامه یابد. انجام همهٔ این کارها هم زمان‌گیر است و هم حافظهٔ اضافی می‌طلبد. در اصطلاح می‌گویند که فراخوانی و اجرای تابع «سربار» دارد. در بعضی حالت‌ها بهتر است با تعریف تابع به شکل ***بی‌واسطه*** از سربار اجتناب کنیم. تابعی که به شکل بی‌واسطه تعریف می‌شود، ظاهری شبیه به توابع معمولی دارد با این فرق که عبارت inline در اعلان و تعریف آن قید شده است.

× مثال 5-16 تابع cube () به شکل بی‌واسطه

این همان تابع cube () مثال 5-3 است:

```
inline int cube(int x)
{  // returns cube of x:
   return x*x*x;
}
```

تنها تفاوت این است که کلمهٔ کلیدی inline در ابتدای عنوان تابع ذکر شده. این عبارت به کامپایلر می‌گوید که در برنامه به جای cube(n) کد واقعی (n) * (n) * (n) را قرار دهد. به برنامهٔ آزمون زیر نگاه کنید:

---

1 – Inline

```
int main()
{  // tests the cube() function:
   cout << cube(4) << endl;
   int x, y;
   cin >> x;
   y = cube(2*x-3);
}
```

این برنامه هنگام کامپایل به شکل زیر درمی‌آید، گویی اصلا تابعی وجود نداشته:

```
int main()
{  // tests the cube() function:
   cout << (4) * (4) * (4) << endl;
   int x, y;
   cin >> x;
   y = (2*x-3) * (2*x-3) * (2*x-3);
}
```

وقتی کامپایلر کد واقعی تابع را جایگزین فراخوانی آن می‌کند، می‌گوییم که تابع بی‌واسطه، ***باز می‌شود***.

احتیاط: استفاده از توابع بی‌واسطه می‌تواند اثرات منفی داشته باشد. مثلا اگر یک تابع بی‌واسطه دارای 40 خط کد باشد و این تابع در 26 نقطه مختلف از برنامهٔ اصلی فراخوانی شود، هنگام کامپایل بیش از هزار خط کد به برنامهٔ اصلی افزوده می‌شود. همچنین تابع بی‌واسطه می‌تواند قابلیت انتقال برنامهٔ شما را روی سیستم‌های مختلف کاهش دهد.

5-13 چندشکلی توابع

در C++ می‌توانیم چند تابع داشته باشیم که همگی یک نام دارند. در این حالت می‌گوییم که تابع مذکور، ***چندشکلی*** دارد. شرط این کار آن است که فهرست پارامترهای این توابع با یکدیگر تفاوت داشته باشد. یعنی تعداد پارامترها متفاوت باشد یا دست کم یکی از پارامترهای متناظر هم نوع نباشند.

× مثال 17-5 چندشکلی تابع ()max

در مثال 3-5 تابع ()max را تعریف کردیم. حالا توابع دیگری با همان نام ولی شکلی متفاوت تعریف می‌کنیم و همه را در یک برنامه به کار می‌گیریم:

```
int max(int, int);
int max(int, int, int);
int max(double, double);

int main()
{  cout << max(99,77) << " " << max(55,66,33) << " " <<
max(44.4,88.8);
}

int max(int x, int y)
{  // returns the maximum of the two given integers:
   return (x > y ? x : y);
}

int max(int x, int y, int z)
{  // returns the maximum of the three given integers:
   int m = (x > y ? x : y);  // m = max(x , y)
   return ( z > m ? z : m);
}

int max(double x, double y)
{  // return the maximum of the two given doubles:
   return (x>y ? x : y);
}
```

```
99 66 88.0
```

در این برنامه سه تابع با نام ()max تعریف شده است. وقتی تابع ()max در جایی از برنامه فراخوانی می‌شود، کامپایلر فهرست آرگومان آن را بررسی می‌کند تا بفهمد که کدام نسخه از max باید احضار شود. مثلا در اولین فراخوانی تابع ()max دو آرگومان int ارسال شده، پس نسخه‌ای که دو پارامتر int در فهرست پارامترهایش

دارد فراخوانی می‌شود. اگر این نسخه وجود نداشته باشد، کامپایلر intها را به double ارتقا می‌دهد و سپس نسخه‌ای که دو پارامتر double دارد را فرا می‌خواند.

توابعی که چندشکلی دارند بسیار فراوان در C++ استفاده می‌شوند. چندشکلی در کلاس‌ها اهمیت فراوانی دارد که این موضوع در فصل 12 بحث خواهد شد.

5-14 تابع main()

اکنون که با توابع آشنا شده‌ایم، نگاه دقیق‌تری به برنامه‌ها بیاندازیم. برنامه‌هایی که تا کنون نوشتیم همه دارای تابعی به نام main() هستند. منطق C++ این طور است که هر برنامه باید دارای تابعی به نام main() باشد. در حقیقت هر برنامه کامل، از یک تابع main() به همراه توابع دیگر تشکیل شده است که هر یک از این توابع به شکل مستقیم یا غیر مستقیم از درون تابع main() فراخوانی می‌شوند. خود برنامه با فراخوانی تابع main() شروع می‌شود. چون این تابع یک نوع بازگشتی int دارد، منطقی است که بلوک تابع main() شامل دستور return 0; باشد هرچند که در برخی از کامپایلرهای C++ این خط اجباری نیست و می‌توان آن را ذکر نکرد. مقدار صحیحی که با دستور return به سیستم عامل برمی‌گردد باید تعداد خطاها را شمارش کند. مقدار پیش‌فرض آن 0 است به این معنا که برنامه بدون خطا پایان گرفته است. با استفاده از دستور return می‌توانیم برنامه را به طور غیرمعمول خاتمه دهیم.

× مثال 5-18 استفاده از دستور return برای پایان دادن به یک برنامه

```
int main()
{  // prints the quotient of two input integers:
   int n, d;
   cout << "Enter two integers: ";
   cin >> n >> d;
   if (d = = 0) return 0;
   cout << n << "/" << d << " = " << n/d << endl;
}
```

```
Enter two integers: 99 17
99/17 = 5
```

اگر کاربر برای ورودی دوم 0 را وارد کند، برنامه بدون چاپ خروجی پایان می‌یابد:

```
Enter two integers: 99 0
```

دستور return تابع فعلی را خاتمه می‌دهد و کنترل را به فراخواننده بازمی‌گرداند. به همین دلیل است که اجرای دستور return در تابع main() کل برنامه را خاتمه می‌دهد.

چهار روش وجود دارد که بتوانیم برنامه را به شکل غیرمعمول (یعنی قبل از این که اجرا به پایان بلوک اصلی برسد) خاتمه دهیم:

1 – استفاده از دستور return

2 – فراخوانی تابع exit()

3 – فراخوانی تابع abort()

4 – ایجاد یک حالت استثنا[1]

طریقهٔ به‌کارگیری تابع exit() در مثال زیر شرح داده شده. این تابع در سرفایل <cstdlib> تعریف شده است. تابع exit() برای خاتمه دادن به کل برنامه در هر تابعی غیر از تابع main() مفید است. به مثال بعدی توجه کنید.

× مثال 19-5 استفاده از تابع exit() برای پایان دادن به برنامه

```
#include <cstdlib>     // defines the exit() function
#include <iostream>    // defines thi cin and cout objects
using namespace std;
double reciprocal(double x);

int main()
{  double x;
   cin >> x;
   cout << reciprocal(x);
}

double reciprocal(double x)
{  // returns the reciprocal of x:
```

1 – Exception

```
   if (x = = 0) exit(1);     // terminate the program
   return 1.0/x;
}
```

در برنامهٔ بالا اگر کاربر عدد 0 را وارد کند، تابع reciprocal() خاتمه می‌یابد و برنامه بدون هیچ مقدار چاپی به پایان می‌رسد.

15-5 آرگومان‌های پیش‌فرض[1]

در C++ می‌توان تعداد آرگومان‌های یک تابع را در زمان اجرا به دلخواه تغییر داد. این امر با استفاده از آرگومان‌های اختیاری و مقادیر پیش‌فرض امکان‌پذیر است.

× مثال 20-5 آرگومان‌های پیش‌فرض

برنامهٔ زیر حاصل چند جمله‌ای درجه سوم $a_0 + a_1x + a_2x^2 + a_3x^3$ را پیدا می‌کند. برای محاسبهٔ این مقدار از الگوریتم هورنر استفاده شده. به این شکل که برای کارایی بیشتر، محاسبه به صورت $a_0 + (a_1 + (a_2 + a_3x)x)x$ دسته‌بندی می‌شود:

```
double p(double, double, double=0, double=0, double=0);

int main()
{  // tests the p() function:
   double x = 2.0003;
   cout << "p(x,7) = " << p(x,7) << endl;
   cout << "p(x,7,6) = " << p(x,7,6) << endl;
   cout << "p(x,7,6,5) = " << p(x,7,6,5) << endl;
   cout << "p(x,7,6,5,4) = " << p(x,7,6,5,4) << endl;
}
double p(double x, double a0, double a1=0, double a2=0, double a3=0)
{  // returns a0 + a1*x + a2*x^2 + a3*x^3:
   return a0 + (a1 + (a2 + a3*x)*x)*x;
}
```

```
p(x,7) = 7
p(x,7,6) = 19.0018
p(x,7,6,5) = 39.0078
p(x,7,6,5,4) = 71.0222
```

1 – Default

هنگامی که p(x, a0, a1, a2, a3) فراخوانی شود، چندجمله‌ای درجه سوم $a_0 + a_1x + a_2x^2 + a_3x^3$ محاسبه می‌شود اما چون a1 و a2 و a3 مقدار پیش‌فرض 0 را دارند، تابع مذکور را می‌توان به صورت p(x,a0) نیز فراخوانی نمود. این فراخوانی معادل فراخوانی p(x,a0,0,0,0) است که برابر با a0 ارزیابی خواهد شد. همچنین می‌توان تابع فوق را به صورت p(x,a0,a1) فراخوانی نمود . این فراخوانی معادل فراخوانی p(x,a0,a1,0,0) است که چندجمله‌ای درجه اول $a_0 + a_1x$ را محاسبه می‌نماید. به همین ترتیب فراخوانی p(x,a0,a1,a2) چندجمله‌ای درجه دوم $a_0 + a_1x + a_2x^2$ را محاسبه می‌کند و همچنین فراخوانی p(x,a0,a1,a2,a3) چند جمله‌ای درجه سوم $a_0 + a_1x + a_2x^2 + a_3x^3$ را محاسبه می‌کند. پس این تابع را می‌توان با 2 یا 3 یا 4 یا 5 آرگومان فراخوانی کرد.

برای این که به یک پارامتر مقدار پیش‌فرض بدهیم باید آن مقدار را در فهرست پارامترهای تابع و جلوی پارامتر مربوطه به همراه علامت مساوی درج کنیم. به این ترتیب اگر هنگام فراخوانی تابع، آن آرگومان را ذکر نکنیم، مقدار پیش‌فرض آن در محاسبات تابع استفاده می‌شود. به همین خاطر به این گونه آرگومان‌ها، آرگومان اختیاری می‌گویند.

دقت کنید که پارامترهایی که مقدار پیش‌فرض دارند باید در فهرست پارامترهای تابع بعد از همهٔ پارامترهای اجباری قید شوند مثل:

```
void f( int a, int b, int c=4, int d=7, int e=3);     // OK
void g(int a, int b=2, int c=4, int d, int e=3);      // ERROR
```

همچنین هنگام فراخوانی تابع، آرگومان‌های ذکر شده به ترتیب از چپ به راست تخصیص می‌یابند و پارامترهای بعدی با مقدار پیش‌فرض پر می‌شوند. مثلا در تابع p() که در بالا قید شد، فراخوانی p(8.0,7,6) باعث می‌شود که پارامتر x مقدار 8.0 را بگیرد سپس پارامتر a0 مقدار 7 را بگیرد و سپس پارامتر a1 مقدار 6 را بگیرد. پارامترهای a2 و a3 مقدار پیش‌فرض‌شان را خواهند داشت. این ترتیب را نمی‌توانیم به هم بزنیم. مثلا نمی‌توانیم تابع را طوری فرا بخوانیم که پارامترهای x و a0 و a3 مستقیما مقدار بگیرند ولی پارامترهای a1 و a2 مقدار پیش‌فرض‌شان را داشته باشند.

پرسش‌های گزینه‌ای

1 – توابع ریاضی C++ استاندارد در کدام سرفایل تعریف شده‌اند؟

الف – سرفایل <iostream> ب – سرفایل <cmath>

ج – سرفایل <cstdlib> د – سرفایل <iomanip>

2 – در تعریف `int f(float a)` کدام گزینه صحیح نیست؟

الف – این کد تابعی به نام f را تعریف می‌کند

ب – تابع فوق متغیری از نوع float دارد

ج – نوع بازگشتی این تابع از نوع int است

د – پارامتر این تابع از نوع int است.

3 – دستور `return` در تابع چه کاری انجام می‌دهد؟

الف – تابع را خاتمه می‌دهد

ب – مقدار نهایی را به فراخواننده برمی‌گرداند

ج – نوع بازگشتی تابع را مشخص می‌کند

د – الف و ب

4 – کدام عبارت صحیح نیست؟

الف – پارامترهای تابع، متغیرهای محلی برای آن تابع محسوب می‌شوند

ب – متغیرهای اعلان شده در یک تابع، متغیرهای محلی آن تابع محسوب می‌شوند

ج – متغیرهای محلی تابع، فقط در طول اجرای تابع موجودند

د – متغیرهای محلی تابع در سراسر برنامه معتبرند

5 – تابعی که مقداری را برنمی‌گرداند، نوع بازگشتی آن چگونه اعلان می‌شود؟

الف – از نوع void ب – از نوع null

ج – از نوع پیش فرض int د – لازم نیست نوع بازگشتی قید شود

6 – نوع بازگشتی یک تابع بولی چیست؟

الف – int ب – void ج – bool د – const

7 – عملگر ارجاع کدام یک از گزینه‌های زیر است؟

الف – * ب – & ج – -> د – :

8 – چه زمانی یک آرگومان «خواندنی–نوشتنی» است؟

الف – وقتی از طریق مقدار ارسال شود

ب – وقتی از طریق ارجاع ثابت ارسال شود

ج – وقتی از طریق ارجاع ارسال شود

د – وقتی با پیشوند `const` اعلان شود

9 – کدام گزینه صحیح است؟

الف – فقط یک مقدار را می‌توان به تابع فرستاد و تابع فقط می‌تواند یک مقدار را بازگرداند

ب – فقط یک مقدار را می‌توان به تابع فرستاد ولی تابع می‌تواند چند مقدار را بازگرداند

ج – چند مقدار را می‌توان به تابع فرستاد ولی تابع می‌تواند فقط یک مقدار را بازگرداند

د – چند مقدار را می‌توان به تابع فرستاد و تابع می‌تواند چند مقدار را بازگرداند

10 – برای تعریف یک تابع به شکل بی‌واسطه از چه کلمهٔ کلیدی استفاده می‌کنیم؟

الف – `const` ب – `inline` ج – `void` د – `int`

11 – کدام عبارت در رابطه با چندشکلی توابع صحیح است؟

الف – یک تابع چندشکلی باید نام‌های متفاوت ولی بدنه‌های یکسان داشته باشد

ب – یک تابع چندشکلی باید نام‌های یکسان ولی فهرست پارامترهای متفاوت داشته باشد

ج – یک تابع چندشکلی باید نام‌های متفاوت ولی فهرست پارامترهای یکسان داشته باشد

د – یک تابع چندشکلی باید نام‌های یکسان و فهرست پارامترهای یکسان داشته باشد

12 – اگر تابع `f` به شکل `void f(int k, int x=0, int y=1)` اعلان شده باشد آنگاه:

الف – پارامتر `k` دارای مقدار پیش‌فرض نیست.

ب – پارامتر `x` دارای مقدار پیش‌فرض `0` است.

ج – پارامتر `y` دارای مقدار پیش‌فرض `1` است.

د – همه موارد فوق صحیح است.

13 – چرا در برنامه‌های بزرگ تعریف توابع را در فایل جداگانه‌ای قرار می‌دهند؟

الف – به این دلیل که مدیریت برنامه آسان شود

ب – به این دلیل که اصل پنهان‌سازی اطلاعات رعایت شود

ج – به این دلیل که بتوان در برنامه‌های دیگر هم از آن توابع استفاده کرد

د – همه موارد فوق

14 – اگر تابع g به شکل `void g(int m, int& n)` اعلان شده باشد آنگاه:

الف – پارامتر m به طریق ارجاع ارسال شده

ب – پارامتر n به طریق ارجاع ارسال شده

ج – پارامتر m به طریق ارجاع ثابت ارسال شده

د – پارامتر n به طریق ارجاع ثابت ارسال شده

15 – اگر تابع سوال 14 به شکل `g(x, y);` فراخوانی شود آنگاه کدام عبارت صحیح است؟

الف – تابع مقدار x را می‌تواند تغییر دهد

ب – تابع مقدار y را می‌تواند تغییر دهد

ج – تابع مقدار x و مقدار y را می‌تواند تغییر دهد

د – تابع مقدار هیچ کدام را نمی‌تواند تغییر دهد.

پرسش‌های تشریحی

1- استفاده از تابع برای بخش‌بندی برنامه چه مزایایی دارد؟
2- چه تفاوتی بین اعلان یک تابع و تعریف آن است؟
3- اعلان یک تابع کجا می‌تواند قرار بگیرد؟
4- برای استفاده از چه توابعی به دستور include نیاز است؟
5- گذاشتن تعریف یک تابع در یک فایل جداگانه چه مزیتی دارد؟
6- کامپایل کردن یک تابع به طور جداگانه چه مزیتی دارد؟
7- چه تفاوت‌هایی بین ارسال یک پارامتر از طریق مقدار و ارسال آن از طریق ارجاع وجود دارد؟
8- چه تفاوت‌هایی بین ارسال یک پارامتر از طریق ارجاع و ارسال آن از طریق ارجاع ثابت وجود دارد؟
9- چرا به پارامتری که از طریق مقدار ارسال می‌شود «فقط خواندنی» گفته می‌شود؟ چرا به پارامتری که از طریق ارجاع ارسال می‌شود «خواندنی-نوشتنی» گفته می‌شود؟
10- چه اشتباهی در اعلان زیر هست؟

```
int f(int a, int b=0, int c);
```

تمرین‌های برنامه‌نویسی

1- توضیح دهید چگونه یک تابع void با یک پارامتر ارجاع می‌تواند به یک تابع غیر void با یک پارامتر مقدار تبدیل گردد.
2- برنامه‌ای شبیه مثال 2-5 بنویسید که صحت رابطهٔ مثلثاتی $\cos 2x = 2\cos^2 x - 1$ را تحقیق کند.
3- برنامه‌ای شبیه مثال 2-5 بنویسید که صحت رابطهٔ مثلثاتی $\cos^2 x + \sin^2 x = 1$ را تحقیق کند.
4- برنامه‌ای شبیه مثال 2-5 بنویسید که صحت تساوی $b^n = e^{(n \log b)}$ را تحقیق کند.
5- تابع min() را که به شکل زیر اعلان می‌شود، نوشته و آزمایش کنید. این تابع از میان چهار عدد صحیح ارسال شده، کوچک‌ترین عدد را برمی‌گرداند:

```
int min(int,int,int,int);
```

6- تابع max() را که به شکل زیر اعلان می‌شود نوشته و آزمایش کنید. این تابع با استفاده از تابع max(int,int) مثال 5-5 بزرگ‌ترین عدد در بین چهار عدد صحیح داده شده را برمی‌گرداند.

```
int max(int,int,int,int);
```

7- تابع min() که به شکل زیر اعلان می‌شود را نوشته و آزمایش کنید. این تابع با استفاده از تابع min(int,int) کوچک‌ترین عدد را از میان چهار عدد صحیح ارسال شده به آن، پیدا کرده و برمی‌گرداند.

```
int main(int,int,int,int);
```

8- تابع average() را که میانگین چهار عدد را برمی‌گرداند، نوشته و آزمایش کنید:

```
float average(float x1, float x2, float x3, float x4)
```

9- تابع average() را که میانگین حداکثر چهار عدد را بر می‌گرداند، نوشته و آزمایش کنید:

```
float average(float x1, float x2 =0, float x3=0, float x4=0)
```

10- تابع فاکتوریال fact() را با یک حلقهٔ for پیاده‌سازی کنید(مثال 9-4 را ببینید). مشخص کنید که چه مقداری از n موجب می‌شود که fact(n) سرریز شود.

11- موثرترین طریق برای محاسبهٔ جایگشت تابع p(n,k) فرمول زیر است:

$$P(n,k) = n(n-1)(n-2)...(n-k+2)(n-k+1)$$

یعنی حاصل‌ضرب k عدد صحیح از n تا n-k+1 . با استفاده از این رابطه، تابع perm() مثال 10-5 را بازنویسی و آزمایش کنید.

12- تابع ترکیب c(n,k) تعداد زیرمجموعه‌های متفاوت (نامرتب) kعنصری که ممکن است از یک مجموعهٔ n عنصری ساخته شود را نشان می‌دهد. این تابع با رابطهٔ زیر بیان می‌شود:

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

این تابع را پیاده‌سازی و آزمایش کنید.

13- تابع ترکیب c(n,k) را می‌توان با استفاده از رابطهٔ زیر بیان نمود:

$$C(n,k) = \frac{P(n,k)}{k!}$$

با استفاده از این رابطه، برنامهٔ مسالهٔ 13-5 را بازنویسی کرده و آزمایش کنید.

14- روش موثرتر برای محاسبهٔ c(n,k) رابطهٔ زیر است:

$$C(n,k) = ((((((n/1)(n-1))/2)(n-2))/3)...(n-k+2))/(k-1))(n-k+1))/k$$

این تابع به طور متناوب ضرب و تقسیم می‌شود، هر دفعه با یک واحد کم‌تر از مقدار فعلی n ضرب می‌شود و بر یک واحد بیشتر از مقدار قبلی با شروع از یک، تقسیم می‌شود. با استفاده از رابطهٔ فوق، تابع مسالهٔ 13-5 را بازنویسی و آزمایش کنید.

راهنمایی: مانند مسالهٔ 12-5 از حلقهٔ for استفاده کنید.

15- مثلث خیام یک آرایهٔ سه‌گوش از اعداد به شکل زیر است:

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5  10  10   5   1
    1   6  15  20  15   6   1
  1   7  21  35  35  21   7   1
1   8  28  56  70  56  28   8   1
```

هر عدد در مثلث خیام یکی از ترکیبات c(n,k) است (به مساله 13-5 نگاه کنید). اگر ردیف‌ها وستون‌ها را با شروع از 0 شمارش کنیم، عدد واقع در ردیف n و ستون k برابر با c(n,k) است. به عنوان مثال عدد15 = c(6,2) در ردیف شماره 6 و ستون شماره 2 است. برنامه‌ای بنویسید که با استفاده از تابع مساله 14-5 یک مثلث خیام دوازده ردیفی چاپ کند.

16- تابع digit() که به شکل زیر اعلان می‌شود را نوشته و آزمایش کنید:

```
int digit(int n, int k);
```

این تابع رقم kام عدد صحیح n را برمی‌گرداند. برای مثال اگر n عدد صحیح 29415 باشد، تابع digit(n,0) رقم 5 را بازمی‌گرداند و فراخوانی digit(n,2) رقم 4 را برمی‌گرداند. توجه کنید که رقم‌ها از راست به چپ و با شروع از 0 شمارش می‌شوند.

17- تابعی را نوشته و آزمایش کنید که الگوریتم اقلیدس را برای بازگرداندن بزرگ‌ترین مقسوم‌علیه مشترک دو عدد صحیح مثبت داده شده به کار می‌گیرد (به مسألهٔ 11 از فصل چهارم نگاه کنید)

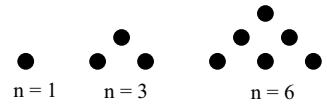
18- تابعی را نوشته و آزمایش کنید که با استفاده از تابع بزرگ‌ترین مقسوم‌علیه مشترک (مسألهٔ 17) کوچک‌ترین مضرب مشترک دو عدد صحیح مثبت را برگرداند.

19- تابعی به نام power() که به شکل زیر اعلان می‌شود را نوشته و آزمایش کنید:

```
double power(double x, int p);
```

این تابع x را به توان p می‌رساند که p می‌تواند هر عدد صحیحی باشد. از الگوریتمی استفاده کنید که برای محاسبهٔ x^{20} مقدار x را 20 مرتبه در خودش ضرب می‌کند.

20- یونانی‌های باستان اعداد را به صورت هندسی طبقه‌بندی می‌کردند. برای مثال به یک عدد مثلثی می‌گفتند اگر آن عدد می‌توانست با ریگ‌ها در یک تقارن مثلثی چیده شود. ده عدد مثلثی اول اعداد 0 و 1 و 3 و 6 و 10 و15 و 21 و 28 و 36 و 45 هستند. تابع بولی زیر را نوشته و آزمایش کنید. اگر n یک عدد مثلثی باشد این تابع مقدار 1 را برمی‌گرداند وگرنه 0 برگشت داده می‌شود:




```
int isTriangular(int n);
```

21- تابع issquare() را نوشته و آزمایش کنید. این تابع تشخیص می‌دهد که آیا عدد داده شده یک عدد مربعی است یا خیر:


```
int isSquare(int n);
```

اولین ده عدد مربعی اعداد 0 و 1 و 4 و 9 و 16 و 25 و 36 و 49 و 64 و 81 هستند.

22- تابع ComputeCircle() که مساحت a و محیط c یک دایره با شعاع داده شدهٔ r را برمی‌گرداند، نوشته وامتحان کنید:

```
void computeCircle(float& a, float& c, float r);
```

23- تابع ComputeTriangle() که مساحت a و محیط p از یک مثلث با اضلاع به طول a و b و c را محاسبه می‌نماید، نوشته و آزمایش کنید:

```
void computeTriangle(float& a, float& p, float a, float b, float c);
```

24- تابع computeSphere() که حجم v و مساحت سطح s را برای یک کره با شعاع داده شدهٔ r برمی‌گرداند، نوشته و آزمایش کنید:

```
void ComputeSphere(float& v, float& s, float r);
```

فصل ششم

«آرایه‌ها»

6-1 مقدمه

یک متغیر بخشی از حافظه است که یک نام دارد و می‌توان مقداری را در آن ذخیره کرد. با استفاده از متغیرها می‌توان به پردازش داده‌ها پرداخت. در برنامه‌های کوچک ممکن است بتوانیم کل پردازش را با استفاده از متغیرها عملی کنیم ولی در برنامه‌هایی که داده‌های فراوانی را پردازش می‌کنند استفاده از متغیرهای معمولی کار عاقلانه‌ای نیست زیرا در بسیاری از این برنامه‌ها «پردازش دسته‌ای» صورت می‌گیرد به این معنی که مجموعه‌ای از داده‌های مرتبط با هم در حافظه قرار داده می‌شود و پس از پردازش، کل این مجموعه از حافظه خارج می‌شود و مجموعهٔ بعدی در حافظه بارگذاری می‌شود. اگر قرار باشد برای این کار از متغیرهای معمولی استفاده شود بیشتر وقت برنامه‌نویس صرف پر و خالی کردن انبوهی از متغیرها می‌شود. به همین دلیل در بیشتر زبان‌های برنامه‌نویسی «آرایه‌ها[1]» تدارک دیده شده‌اند. آرایه را می‌توان متغیری تصور کرد که یک نام دارد ولی چندین مقدار را به طور هم‌زمان نگهداری می‌نماید.

---

1 – Arrays

چنانچه بعد خواهیم دید، ویرایش محتویات آرایه‌ها بسیار آسان است و پردازش داده‌ها با استفاده از آرایه‌ها سریع‌تر و راحت‌تر صورت می‌گیرد.

یک آرایه، یک زنجیره از متغیرهایی است که همه از یک نوع هستند. به این متغیرها «اعضای آرایه» می‌گویند. هر عضو آرایه با یک شماره مشخص می‌شود که به این شماره «***ایندکس***[2]» یا «زیرنویس» می‌گویند (نام زیرنویس از نمایش ریاضی آرایه‌ها اقتباس شده). ایندکس محل قرار گرفتن هر عضو آرایه را نشان می‌دهد. مثلا اگر نام آرایه‌ای a باشد، آنگاه a[0] نام عنصری است که در موقعیت صفر آرایه قرار گرفته و a[1] نام عنصری است که در موقیت 1 آرایه قرار دارد. پس عنصر nام آرایه در محل a[n-1] قرار دارد. می‌بینید که شماره‌گذاری عناصر آرایه از صفر شروع می‌شود و برای یک آرایهٔ n عنصری این شماره تا n-1 ادامه می‌یابد. علت این که شماره‌گذاری از صفر شروع می‌شود این است که به این روش، اندیس یک عنصر از آرایه، فاصلهٔ آن عنصر از عنصر اول را نشان می‌دهد. مثلا a[3] به عنصری اشاره دارد که سه خانه از عنصر اول یعنی a[0] فاصله دارد. این فاصله‌یابی بعدها به کار می‌آید.

عناصر یک آرایه در خانه‌های پشت سر هم در حافظه ذخیره می‌شوند. به این ترتیب آرایه را می‌توان بخشی از حافظه تصور کرد که این بخش خود به قسمت‌های مساوی تقسیم شده و هر قسمت به یک عنصر تعلق دارد. شکل مقابل آرایهٔ a که پنج عنصر دارد را نشان می‌دهد. عنصر a[0] حاوی مقدار 17.5 و عنصر a[1] حاوی 19.0 و عنصر a[4] حاوی مقدار 18.0 است. این مقدارها می‌توانند نمرات یک دانشجو در یک نیم‌سال تحصیلی را نشان دهند.

0	17.50
1	19.00
2	16.75
3	15.00
4	18.00

6-2 پردازش آرایه‌ها

آرایه‌ها را می‌توان مثل متغیرهای معمولی تعریف و استفاده کرد. با این تفاوت که آرایه یک متغیر مرکب است و برای دستیابی به هر یک از خانه‌های آن باید از ایندکس استفاده نمود.

---

1 – Index

× مثال 6-1 دستیابی مستقیم به عناصر آرایه

برنامهٔ سادهٔ زیر یک آرایهٔ سه عنصری را تعریف می‌کند و سپس مقادیری را در آن قرار داده و سرانجام این مقادیر را چاپ می‌کند:

```
int main()
{  int a[3];
   a[2] = 55;
   a[0] = 11;
   a[1] = 33;
   cout << "a[0] = " << a[0] << endl;
   cout << "a[1] = " << a[1] << andl;
   cout << "a[2] = " << a[2] << endl;
}
```

```
a[0] = 11
a[1] = 33
a[2] = 55
```

خط دوم، یک آرایهٔ سه عنصری از نوع int تعریف می‌کند. سه خط بعدی، مقادیری را به این سه عنصر تخصیص می‌دهد و سه خط آخر هم مقدار هر عنصر آرایه را چاپ می‌کند.

× مثال 6-2 چاپ ترتیبی عناصر یک آرایه

برنامهٔ زیر پنج عدد را می‌خواند و سپس آن‌ها را به ترتیب معکوس چاپ می‌کند:

```
int main()
{  const int SIZE=5; // defines the size N for 5 elements
   double a[SIZE];   // declares the array's elements as type double
   cout << "Enter " << SIZE << " numbers:\t";
   for (int i=0; i<SIZE; i++)
      cin >> a[i];
   cout << "In reverse order: ";
   for (int i=SIZE-1; i>=0; i--)
      cout << "\t" << a[i];
}
```

```
Enter 5 numbers:    11.11    33.33    55.55    77.77    99.99
In reverse order:   99.99    77.77    55.55    33.33    11.11
```

دومین خط برنامه، ثابتی به نام SIZE از نوع int تعریف می‌کند و مقدار 5 را درون آن قرار می‌دهد. خط سوم یک آرایه به نام a و با پنج عنصر از نوع float تعریف می‌نماید. سپس اولین حلقهٔ for پنج عنصر را به داخل آرایه می‌خواند و دومین حلقهٔ for آن پنج عنصر را به ترتیب معکوس چاپ می‌کند.

مثال فوق نشان داد که یک آرایه چطور اعلان می‌شود. نحو کلی برای اعلان آرایه به شکل زیر است:

type *array_name*[array_size];

عبارت **type** نوع عناصر آرایه را مشخص می‌کند. *array_name* نام آرایه است و array_size تعداد عناصر آرایه را نشان می‌دهد. این مقدار باید یک عدد ثابت صحیح باشد و حتما باید داخل کروشه [] قرار بگیرد.

در خط دوم از مثال 1-6 آرایه‌ای به نام a با سه عنصر اعلان شده که این عناصر از نوع int هستند. در خط سوم از مثال 2-6 آرایه‌ای به نام a با تعداد عناصری که ثابت SIZE مشخص می‌نماید اعلان شده که عناصر آن از نوع double هستند. معمولا بهتر است تعداد عناصر آرایه را با استفاده از ثابت‌ها مشخص کنیم تا بعد بتوانیم با استفاده از همان ثابت در حلقهٔ for آرایه را پیمایش نماییم.

6-3 مقداردهی آرایه‌ها

در C++ می‌توانیم یک آرایه را با استفاده از فهرست مقداردهی، اعلان و مقدارگذاری کنیم:

```
float a[] = {22.2,44.4.66.6};
```

به این ترتیب مقادیر داخل فهرست به همان ترتیبی که چیده شده‌اند درون عناصر آرایه قرار می‌گیرند. اندازه آرایه نیز برابر با تعداد عناصر موجود در فهرست خواهد بود. پس همین خط مختصر، آرایه‌ای از نوع float و با نام a و با تعداد سه عنصر اعلان کرده و هر سه عنصر را با مقدارهای درون فهرست، مقداردهی می‌کند.

	a
0	22.2
1	44.4
2	66.6

× مثال 3-6 مقداردهی آرایه با استفاده از فهرست مقداردهی

برنامهٔ زیر، آرایهٔ a را مقداردهی کرده و سپس مقدار هر عنصر را چاپ می‌کند:

```
int main()
{  float a[] = { 22.2, 44.4, 66.6 };
   int size = sizeof(a)/sizeof(float);
   for (int i=0; i<size; i++)
      cout << "\ta[" << i << "] = " << a[i] << endl;
}
```

```
  a[0] = 22.2
  a[1] = 44.4
  a[2] = 66.6
```

دومین خط، آرایهٔ a را به همان صورتی که در بالا آمد اعلان و مقداردهی می‌کند. در خط سوم، از تابع sizeof() استفاده شده. این تابع اندازهٔ آرگومان ارسالی به آن را بر حسب بایت برمی‌گرداند. مقدار sizeof(float) برابر با 4 است زیرا در این رایانه هر متغیر float چهار بایت از حافظه را اشغال می‌کند. همچنین مقدار sizeof(a) برابر با 12 است زیرا آرایهٔ مذکور دوازده بایت از حافظه را اشغال نموده (سه خانه که هر کدام چهار بایت است). حاصل تقسیم این دو مقدار، تعداد عناصر آرایه را مشخص می‌نماید. با استفاده از این روش همیشه می‌توانیم تعداد عناصر یک آرایه را فقط با دانستن نوع آرایه محاسبه کنیم. حاصل تقسیم sizeof(a)/sizeof(float) درون متغیر size قرار می‌گیرد تا از این مقدار در حلقهٔ for برای پیمایش و چاپ عناصر آرایه a استفاده شود.

هنگام استفاده از فهرست مقداردهی برای اعلان آرایه، می‌توانیم تعداد عناصر آرایه را هم به طور صریح ذکر کنیم. در این صورت اگر تعداد عناصر ذکر شده از تعداد عناصر موجود در فهرست مقداردهی بیشتر باشد، خانه‌های بعدی با مقدار صفر پر می‌شوند:

	a
0	55.5
1	66.6
2	77.7
3	0.0
4	0.0
5	0.0
6	0.0

```
float a[7] = { 55.5, 66.6, 77.7 };
```

اعلان بالا آرایهٔ a را با هفت عنصر از نوع float تعریف می‌کند. سه عنصر اول این آرایه با استفاده از فهرست مذکور مقداردهی می‌شوند و در چهار عنصر باقی‌مانده مقدار صفر قرار می‌گیرد.

× مثال 4-6 مقداردهی یک آرایه با صفرهای متوالی

برنامهٔ زیر، آرایه‌ای به نام a را اعلان و مقداردهی کرده و سپس مقدار عناصر آن را چاپ می‌کند:

```
int main()
{  float a[6] = { 22.2, 44.4, 66.6 };
   int size = sizeof(a)/sizeof(float);
   for (int i=0; i<size; i++)
      cout << "\ta[" << i << "] = " << a[i] << endl;
}
```

```
 a[0] = 22.2
 a[1] = 44.4
 a[2] = 66.6
 a[3] = 0
 a[4] = 0
 a[5] = 0
```

دقت کنید که تعداد مقادیر موجود در فهرست مقداردهی نباید از تعداد عناصر آرایه بیشتر باشد:

```
float a[3] = { 22.2, 44.4, 66.6, 88.8 };  // ERROR: too many values!
```

یک آرایه را می‌توانیم به طور کامل با صفر مقداردهی اولیه کنیم. برای مثال سه اعلان زیر با هم برابرند:

```
float a[ ] = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };
float a[9] = { 0, 0 };
float a[9] = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };
```

اما مطلب فوق اصلا به این معنی نیست که از فهرست مقداردهی استفاده نشود. درست مثل یک متغیر معمولی، اگر یک آرایه مقداردهی اولیه نشود، عناصر آن حاوی مقادیر زباله خواهد بود.

× مثال 6-5 یک آرایهٔ مقداردهی نشده

برنامهٔ زیر، آرایهٔ a را اعلان می‌کند ولی مقداردهی نمی‌کند. با وجود این، مقادیر موجود در آن را چاپ می‌کند:

```
int main()
{  const int SIZE=4; // defines the size N for 4 elements
   float a[SIZE];    // declares the array's elements as float
   for (int i=0; i<SIZE; i++)
      cout << "\ta[" << i << "] = " << a[i] << endl;
}
```

```
 a[0] = 6.01838e-39
 a[1] = 9.36651e-39
 a[2] = 6.00363e-39
 a[3] = 0
```

توجه کنید که مقادیر درون یک آرایهٔ مقداردهی نشده ممکن است صفر باشد یا نباشد، بسته به این که در آن قسمت از حافظه قبلا چه بوده است.

آرایه‌ها را می‌توان با استفاده از عملگر جایگزینی مقداردهی کرد اما نمی‌توان مقدار آن‌ها را به یکدیگر تخصیص داد:

```
float a[7] = { 22.2, 44.4, 66.6};
float b[7] = { 33.3, 55.5, 77.7};
b = a;    // ERROR: arrays cannot be assigned!
```

همچنین نمی‌توانیم یک آرایه را به طور مستقیم برای مقداردهی به آرایهٔ دیگر استفاده کنیم:

```
float a[7] = { 22.2, 44.4, 66.6};
float b[7] = a;   // ERROR: arrays cannot be used as
initializers!
```

6-4 ایندکس بیرون از حدود آرایه

در بعضی از زبان‌های برنامه‌نویسی، ایندکس آرایه نمی‌تواند از محدودهٔ تعریف شده برای آن بیشتر باشد. برای مثال در پاسکال اگر آرایهٔ a با تعداد پنج عنصر تعریف شده باشد و آنگاه a[7] دستیابی شود، برنامه از کار می‌افتد. این سیستم حفاظتی در

C++ وجود ندارد. مثال بعدی نشان می‌دهد که ایندکس یک آرایه هنگام دستیابی می‌تواند بیشتر از عناصر تعریف شده برای آن باشد و باز هم بدون این که خطایی گرفته شود، برنامه ادامه یابد.

× مثال 6-6 تجاوز ایندکس آرایه از محدودهٔ تعریف شده برای آن

برنامهٔ زیر یک خطای زمان اجرا دارد؛ به بخشی از حافظه دستیابی می‌کند که از محدودهٔ آرایه بیرون است:

```
int main()
{  const int SIZE=4;
   float a[SIZE} = { 33.3, 44.4, 55.5, 66.6};
   for (int i=0; i<7; i++)    // ERROR: index is out of bounds!
      cout << "\ta[" << i << "] = " << a[i] << endl;
}
```

```
 a[0] = 33.3
 a[1] = 44.4
 a[2] = 55.5
 a[3] = 66.6
 a[4] = 5.60519e-45
 a[5] = 6.01888e-39
 a[6] = 6.01889e-39
```

آرایه‌ای که در این برنامه تعریف شده، چهار عنصر دارد ولی تلاش می‌شود به هفت عنصر دستیابی شود. سه مقدار آخر واقعا جزو آرایه نیستند و فقط سلول‌هایی از حافظه‌اند که دقیقا بعد از عنصر چهارم آرایه قرار گرفته‌اند. این سلول‌ها دارای مقدار زباله هستند.

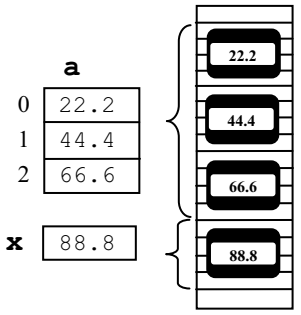
اگر ایندکس آرایه از محدودهٔ تعریف شده برای آن تجاوز کند، ممکن است ناخواسته مقدار سایر متغیرها دست‌کاری شوند و این باعث بروز فاجعه شود. به مثال زیر نگاه کنید.

× مثال 6-7 اثر همسایگی

برنامهٔ زیر از ایندکس خارج از محدوده استفاده می‌کند و این باعث می‌شود که مقدار یک متغیر به طور ناخواسته تغییر کند:

```
int main()
{  const int SIZE=4;
   float a[] = { 22.2, 44.4, 66.6 };
   float x=11.1;
   cout << "x = " << x << endl;
   a[3] = 88.8;   // ERROR: index is out of bounds!
   cout << "x = " << x << endl;
}
```

```
x = 88.8
```



متغیر x بعد از آرایهٔ a اعلان شده، پس یک سلول چهاربایتی بلافاصله بعد از دوازده بایت آرایه به آن تخصیص می‌یابد. بنابراین وقتی برنامه تلاش می‌کند مقدار 88.8 را در a[3] قرار دهد (که جزو آرایه نیست) این مقدار به شکل ناخواسته در x قرار می‌گیرد. شکل مقابل نشان می‌دهد چطور این اتفاق در حافظه رخ می‌دهد.

این خطا یکی از وحشت‌ناک‌ترین خطاهای زمان اجراست زیرا ممکن است اصلا نتوانیم منبع خطا را کشف کنیم. حتی ممکن است به این روش داده‌های برنامه‌های دیگری که در حال کارند را خراب کنیم و این باعث ایجاد اختلال در کل سیستم شود. به این خطا «اثر همسایگی» می‌گویند. این وظیفهٔ برنامه‌نویس است که تضمین کند ایندکس آرایه هیچ‌گاه از محدودهٔ آن خارج نشود.

مثال بعدی نوع دیگری از خطای زمان اجرا را نشان می‌دهد: وقتی ایندکس آرایه بیش از حد بزرگ باشد.

× مثال 8-6 ایجاد استثنای مدیریت نشده[1]

برنامهٔ زیر از کار می‌افتد زیرا ایندکس آرایه خیلی بزرگ است:

```
int main()
{  const int SIZE=4;
```

1 – Unhandled exception

```
    float a[] = { 22.2, 44.4, 66.6};
    float x=11.1;
    cout << "x = " << x << endl;
    a[3333] = 88.8;     // ERROR: index is out of bounds!
    cout << "x = " << x << endl;
}
```

وقتی این برنامه روی رایانه‌ای با سیستم عامل ویندوز اجرا شود، یک صفحهٔ هشدار که در شکل نشان داده شده روی صفحه ظاهر می‌شود. این پنجره بیان می‌کند که برنامه تلاش دارد به نشانی 0040108e از حافظه دستیابی کند. این مکان خارج از حافظهٔ تخصیصی است که برای این برنامه منظور شده، بنابراین سیستم عامل برنامه را متوقف می‌کند.

خطایی که در مثال 8-6 بیان شده یک «استثنای مدیریت نشده» نامیده می‌شود زیرا کدی وجود ندارد که به این استثنا پاسخ دهد. در C++ می‌توانیم کدهایی به برنامه اضافه کنیم که هنگام رخ دادن حالت‌های استثنا، از توقف برنامه جلوگیری کند. به این کدها «پردازش‌گر استثنا[1]» می‌گویند.

برخلاف بعضی از زبان‌های برنامه‌نویسی دیگر (مثل پاسکال و جاوا) آرایه‌ها را نمی‌توان به طور مستقیم به یکدیگر تخصیص داد و ایندکس آرایه‌ها نیز می‌تواند از محدودهٔ آرایه فراتر رود. این‌ها باعث ایجاد خطاهای زمان کامپایل و خطاهای زمان اجرا می‌شود. برای جلوگیری از بروز این خطاها، برنامه‌نویس باید دقت مضاعفی را در برنامه به کار بگیرد تا بتواند کد سریع‌تر و مطمئن‌تری تولید کند.

6-5 ارسال آرایه به تابع

کد `float a[];` که آرایه a را اعلان می‌کند دو چیز را به کامپایلر می‌گوید: این که نام آرایه a است و این که عناصر آرایه از نوع float هستند. سمبل a نشانی

1 – Exception handler

حافظهٔ آرایه را ذخیره می‌کند. لازم نیست تعداد عناصر آرایه به کامپایلر گفته شود زیرا از روی نشانی موجود در a می‌توان عناصر را بازیابی نمود. به همین طریق می‌توان یک آرایه را به تابع ارسال کرد. یعنی فقط نوع آرایه و نشانی حافظهٔ آن به عنوان پارامتر به تابع فرستاده می‌شود.

× مثال 9-6 ارسال آرایه به تابعی که مجموع عناصر آرایه را برمی‌گرداند

```
int sum(int[],int);
int main()
{  int a[] = { 11, 33, 55, 77 };
   int size = sizeof(a)/sizeof(int);
   cout << "sum(a,size) = " << sum(a,size) << endl;
}
int sum(int a[], int n)
{  int sum=0;
   for (int i=0; i<n; i++)
      sum += a[i];
   return sum;
}
```

```
sum(a,size) = 176
```

فهرست پارامتر تابع فوق به شکل (int a[], int n) است به این معنا که این تابع یک آرایه از نوع int و یک متغیر از نوع int دریافت می‌کند. به اعلان این تابع در بالای تابع main() نگاه کنید. نام پارامترها حذف شده است. هنگام فراخوانی تابع نیز از عبارت sum(a,size) استفاده شده که فقط نام آرایه به تابع ارسال شده. نام آرایه در حقیقت نشانی اولین عنصر آرایه است (یعنی a[0]) . تابع از این نشانی برای دستیابی به عناصر آرایه استفاده می‌کند. همچنین تابع می‌تواند با استفاده از این نشانی، محتویات عناصر آرایه را دست‌کاری کند. پس ارسال آرایه به تابع شبیه ارسال متغیر به طریق ارجاع است. به مثال بعدی دقت کنید.

× مثال 10-6 توابع ورودی و خروجی برای یک آرایه

در این برنامه از تابع read() استفاده می‌شود تا مقادیری به داخل آرایه وارد شود. سپس با استفاده از تابع print() مقادیر داخل آرایه چاپ می‌شوند:

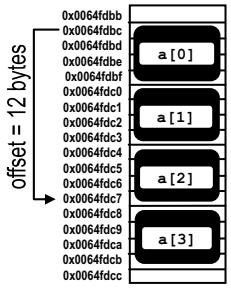
```
void read(int[],int&;)
void print(int[],int);
int main()
{  const int MAXSIZE=100;
   int a[MAXSIZE]={0}, size;
   read(a,size);
   cout << "The array has " << size << " elements: ";
   print(a,size);
}
void read(int a[], int& n)
{  cout << "Enter integers. Terminate with 0:\n";
   n = 0;
   do
   {  cout << "a[" << n << "]: ";
      cin >> a[n];
   {  while (a[n++] !=0 && n < MAXSIZE);
      --n;   // don't count the 0
}
void print(int a[], int n)
{  for (int i=0; i<n; i++)
      cout << a[i] << " ";
}
```

```
Enter integers. Terminate with 0:
a[0]: 11
a[1]: 22
a[2]: 33
a[3]: 44
a[4]: 0
The array has 4 elements: 11 22 33 44
```

تابع read() مقادیر آرایهٔ a و همچنین مقدار پارامتر n که تعداد عناصر آرایه است را تغییر می‌دهد. چون n یک متغیر است، برای این که تابع read() بتواند مقدار آن را تغییر دهد این متغیر باید به شکل ارجاع ارسال شود. همچنین برای این که تابع مذکور بتواند مقادیر داخل آرایه a را تغییر دهد، آرایه نیز باید به طریق ارجاع ارسال شود، اما ارجاع آرایه‌ها کمی متفاوت است.

در C++ توابع قادر نیستند تعداد عناصر آرایهٔ ارسالی را تشخیص دهند. بنابراین به منظور ارسال آرایه‌ها به تابع از سه مشخصه استفاده می‌شود: 1 – آدرس اولین خانهٔ آرایه 2 – تعداد عناصر آرایه 3 – نوع عناصر آرایه. تابع با استفاده از این سه عنصر می‌تواند به تک تک اعضای آرایه دستیابی کند. شکل کار هم به این طریق است که با استفاده از آدرس اولین خانه می‌توان به محتویات آن دسترسی داشت. از طرفی وقتی نوع عناصر مشخص باشد، معلوم می‌شود که هر خانهٔ آرایه چند بایت از حافظه را اشغال می‌کند. پس اگر این مقدار به آدرس خانهٔ اول اضافه شود، آدرس خانهٔ دوم بدست می‌آید. اگر مقدار مذکور به آدرس خانهٔ دوم اضافه شود، خانهٔ سوم معلوم می‌شود و به همین ترتیب تابع می‌تواند به کل آرایه دستیابی داشته باشد. با استفاده از تعداد عناصر آرایه که با یک پارامتر مجزا ارسال می‌شود، می‌توانیم مراقب باشیم که ایندکس آرایه از حد مجاز فراتر نرود.

آدرس اولین خانهٔ آرایه، همان نام آرایه است. پس وقتی نام آرایه را به تابع بفرستیم آدرس اولین خانه را به تابع فرستاده‌ایم. نوع آرایه نیز در تعریف تابع اعلان می‌شود. بنابراین با این دو مقدار، تابع می‌تواند به آرایه دسترسی داشته باشد.



به برنامهٔ مثال 10-6 نگاه کنید. آرایه a با 100 عنصر از نوع int تعریف شده است. با اجرای کد read(a,size) آدرس اولین خانهٔ آرایه که در a قرار دارد به تابع read() فرستاده می‌شود. هر بار که درون تابع کد cin >> a[n]; اجرا شود، آدرس خانهٔ nام به شیوهٔ بالا محاسبه می‌شود و مقدار ورودی در آن قرار می‌گیرد. مثلا اگر n=3 باشد، a[3] سه پله از خانهٔ a[0] فاصله دارد. چون آرایهٔ a از نوع int است و نوع int چهار بایت از حافظه را اشغال می‌کند، پس آدرس a[3] به اندازهٔ 3*4=12 بایت از خانهٔ a[0] فاصله دارد. لذا به اندازه دوازده بایت به آدرس اولین خانه (یعنی a[0]) افزوده می‌شود تا به خانهٔ a[3] برسیم. به مقدار 12 «***آفست***[1]» عنصر a[3] می‌گویند. آفست یک عنصر از آرایه،

1 – Offset

عددی است که باید به نشانی خانهٔ اول افزوده شود تا به خانهٔ آن عنصر برسیم.

از همهٔ گفته‌های فوق نتیجه می‌شود که برای ارسال یک آرایه به تابع فقط کافی است نام آرایه و اندازهٔ آرایه به تابع فرستاده شود. باز هم تاکید می‌کنیم که نام آرایه، آدرس اولین عنصر آرایه را در خود دارد. مثلا اگر آرایهٔ a در آدرس 0x0064fdbc واقع شده باشد، آنگاه درون a مقدار 0x0064fdbc قرار دارد. همچنین نام آرایه شبیه یک ثابت عمل می‌کند. یعنی آدرس یک آرایه همیشه ثابت است و آرایه نمی‌تواند به مکان دیگری از حافظه تغییر مکان دهد.

× مثال 11-6 آدرس اولین خانهٔ آرایه و مقدار درون آن

برنامهٔ زیر، آدرس ذخیره شده در نام آرایه و مقدار موجود در آن خانه را چاپ می‌کند:

```
int main()
{  int a[] = { 22, 44, 66, 88 };
   cout << "a = " << a << endl;    // the address of a[0]
   cout << "a[0] = " << a[0];      // the value of a[0]
}
```

```
a = 0x0064fdec
a[0] = 22
```

این برنامه تلاش می‌کند که به طور مستقیم مقدار a را چاپ کند. نتیجهٔ چاپ a این است که یک آدرس به شکل شانزده دهی چاپ می‌شود. این همان آدرس اولین خانهٔ آرایه است. یعنی درون نام a آدرس اولین عنصر آرایه قرار گرفته. خروجی نیز نشان می‌دهد که a آدرس اولین عنصر را دارد و a[0] مقدار اولین عنصر را.

6-6 الگوریتم جستجوی خطی[1]

آرایه‌ها بیشتر برای پردازش یک زنجیره از داده‌ها به کار می‌روند. اغلب لازم است که بررسی شود آیا یک مقدار خاص درون یک آرایه موجود است یا خیر. ساده‌ترین راه این است که از اولین عنصر آرایه شروع کنیم و یکی یکی همهٔ عناصر

1 – Linear searching

آرایه را جستجو نماییم تا بفهمیم که مقدار مورد نظر در کدام عنصر قرار گرفته. به این روش «جستجوی خطی» می‌گویند.

× مثال 12-6 جستجوی خطی

برنامهٔ زیر تابعی را آزمایش می‌کند که در این تابع از روش جستجوی خطی برای یافتن یک مقدار خاص استفاده شده:

```
int index(int,int[],int);
int main()
{  int a[] = { 22, 44, 66, 88, 44, 66, 55};
   cout << "index(44,a,7) = " << index(44,a,7) << endl;
   cout << "index(50,a,7) = " << index(50,a,7) << endl;
}
int index(int x, int a[], int n)
{  for (int i=0; i<n; i++)
      if (a[i] == x) return i;
   return n;   // x not found
}
```

```
index(44,a,7) = 1
index(40,a,7) = 7
```

تابع index() سه پارامتر دارد: پارامتر x مقداری است که قرار است جستجو شود، پارامتر a آرایه‌ای است که باید در آن جستجو صورت گیرد و پارامتر n هم ایندکس عنصری است که مقدار مورد نظر در آن پیدا شده است. در این تابع با استفاده از حلقهٔ for عناصر آرایه a پیمایش شده و مقدار هر عنصر با x مقایسه می‌شود. اگر این مقدار با x برابر باشد، ایندکس آن عنصر بازگردانده شده و تابع خاتمه می‌یابد. اگر مقدار x در هیچ یک از عناصر آرایه موجود نباشد، مقداری خارج از ایندکس آرایه بازگردانده می‌شود که به این معناست که مقدار x در آرایهٔ a موجود نیست. در اولین اجرای آزمایشی، مشخص شده که مقدار 44 در a[1] واقع است و در اجرای آزمایشی دوم مشخص شده که مقدار 40 در آرایهٔ a موجود نیست (یعنی مقدار 44 در a[7] واقع است و از آن‌جا که آرایهٔ a فقط تا a[6] عنصر دارد، مقدار 7 نشان می‌دهد که 40 در آرایه موجود نیست).

6-7 مرتب‌سازی حبابی[1]

جستجوی دودویی خیلی کارآمد نیست. هیچ کس برای یافتن معنی یک کلمه در واژه‌نامه، همهٔ کلمات را از ابتدا جستجو نمی‌کند زیرا کلمات در واژه‌نامه به ترتیب حروف الفبا مرتب شده است و برای یافتن معنی یک کلمه کافی است به طور مستقیم به بخشی برویم که حرف اول کلمهٔ ما در آن بخش فهرست شده است. به این صورت جستجو بسیار سریع‌تر صورت می‌گیرد و پاسخ در زمان کوتاه‌تری حاصل می‌شود. اما شرط این جستجو آن است که همهٔ عناصر مرتب باشند.

روش‌های زیادی برای مرتب کردن یک آرایه وجود دارد. «مرتب‌سازی حبابی» یکی از ساده‌ترین الگوریتم‌های مرتب‌سازی است. در این روش، آرایه چندین مرتبه پویش می‌شود و در هر مرتبه بزرگ‌ترین عنصر موجود به سمت بالا هدایت می‌شود و سپس محدودهٔ مرتب‌سازی برای مرتبهٔ بعدی یکی کاسته می‌شود. در پایان همهٔ پویش‌ها، آرایه مرتب شده است. طریقهٔ یافتن بزرگ‌ترین عنصر و انتقال آن به بالای عناصر دیگر به این شکل است که اولین عنصر آرایه با عنصر دوم مقایسه می‌شود. اگر عنصر اول بزرگ‌تر بود، جای این دو با هم عوض می‌شود. سپس عنصر دوم با عنصر سوم مقایسه می‌شود. اگر عنصر دوم بزرگ‌تر بود، جای این دو با هم عوض می‌شود و به همین ترتیب مقایسه و جابجایی زوج‌های همسایه ادامه می‌یابد تا وقتی به انتهای آرایه رسیدیم، بزرگ‌ترین عضو آرایه در خانهٔ انتهایی قرار خواهد گرفت. سپس محدودهٔ جستجو یکی کاسته می‌شود و دوباره زوج‌های کناری یکی یکی مقایسه می‌شوند تا عدد بزرگ‌تر بعدی به مکان بالای محدوده منتقل شود. این پویش ادامه می‌یابد تا این که وقتی محدوده جستجو به عنصر اول محدود شد، آرایه مرتب شده است.

× مثال 6-13 مرتب‌سازی حبابی

برنامهٔ زیر تابعی را آزمایش می‌کند که این تابع با استفاده از مرتب‌سازی حبابی یک آرایه را مرتب می‌نماید:

---

1 – Bobble sorting

```
void print(float[],int);
void sort(float[],int);
int main()
{  float a[] = {55.5, 22.2, 99.9, 66.6, 44.4, 88.8, 33.3, 77.7};
   print(a,8);
   sort(a,8);
   print(a,8);
}
void sort(float a[], int n)
{  // bubble sort:
   for (int i=1; i<n; i++)
      // bubble up max{a[0..n-i]}:
      for (int j=0; j<n-i; j++)
         if (a[j] > a[j+1]) swap (a[j],a[j+1]);
      //INVARIANT: a[n-1-i..n-1] is sorted
}
```

```
55.5, 22.2, 99.9, 66.6, 44.4, 88.8, 33.3, 77.7
22.2, 33.3, 44.4, 55.5, 66.6, 77.7, 88.8, 99.9
```

تابع sort() از دو حلقهٔ تودرتو استفاده می‌کند. حلقه for داخلی زوج‌های همسایه را با هم مقایسه می‌کند و اگر آن‌ها خارج از ترتیب باشند، جای آن دو را با هم عوض می‌کند. وقتی for داخلی به پایان رسید، بزرگ‌ترین عنصر موجود در محدودهٔ فعلی به انتهای آن هدایت شده است. سپس حلقهٔ for بیرونی محدودهٔ جستجو را یکی کم می‌کند و دوباره for داخلی را راه می‌اندازد تا بزرگ‌ترین عنصر بعدی به سمت بالای آرایه هدایت شود.

6-8 الگوریتم جستجوی دودویی[1]

در روش جستجوی دودویی به یک آرایهٔ مرتب نیاز است. هنگام جستجو آرایه از وسط به دو بخش بالایی و پایینی تقسیم می‌شود. مقدار مورد جستجو با آخرین عنصر بخش پایینی مقایسه می‌شود. اگر این عنصر کوچک‌تر از مقدار جستجو بود، مورد جستجو در بخش پایینی وجود ندارد و باید در بخش بالایی به دنبال آن گشت.

1 - Binary searching

دوباره بخش بالایی به دو بخش تقسیم می‌گردد و گام‌های بالا تکرار می‌شود. سرانجام محدودهٔ جستجو به یک عنصر محدود می‌شود که یا آن عنصر با مورد جستجو برابر است و عنصر مذکور یافت شده و یا این که آن عنصر با مورد جستجو برابر نیست و لذا مورد جستجو در آرایه وجود ندارد. این روش پیچیده‌تر از روش جستجوی خطی است اما در عوض بسیار سریع‌تر به جواب می‌رسیم. البته به شرطی به جواب می‌رسیم که آرایه مرتب شده باشد.

× مثال 14-6 جستجوی دودویی

برنامهٔ آزمون زیر با برنامهٔ آزمون مثال 12-6 یکی است اما تابعی که در زیر آمده از روش جستجوی دودویی برای یافتن مقدار درون آرایه استفاده می‌کند:

```
int index(int, int[],int);
int main()
{  int a[] = { 22, 33, 44, 55, 66, 77, 88 };
   cout << "index(44,a,7) = " << index(44,a,7) << endl;
   cout << "index(60,a,7) = " << index(60,a,7) << endl;
}

int index(int x, int a[], int n)
{  // PRECONDITION: a[0] <= a[1] <= ... <= a[n-1];
   // binary search:
   int lo=0, hi=n-1, i;
   while (lo <= hi)
   {  i = (lo + hi)/2;          // the average of lo and hi
      if (a[i] == x) return i;
      if (a[i] < x) lo = i+1;   // continue search in a[i+1..hi]
      else hi = i-1;            // continue search in a[0..i-1]
   }
 return n;                    // x was not found in a[0..n-1]
}
```

```
index(44,a,7) = 2
index(60,a,7) = 7
```

دقت کنید که آرایه، قبل از به کارگیری جستجوی دودویی باید مرتب باشد. این پیش‌شرط به شکل توضیح در برنامهٔ اصلی قید شده است.

برای این که بفهمیم تابع چطور کار می‌کند، فراخوانی index(44,a,7) را دنبال می‌کنیم. وقتی حلقه شروع می‌شود، x=44 و n=7 و lo=0 و hi=6 است. ابتدا i مقدار 3 = 2/(0+6) را می‌گیرد.پس عنصر a[i] عنصر وسط آرایهٔ a[0..6] است. مقدار a[3] برابر با 55 است که از مقدار x بزرگ‌تر است. پس x در نیمهٔ بالایی نیست و جستجو در نیمهٔ پایینی ادامه می‌یابد. لذا hi با i-1 یعنی 2 مقداردهی می‌شود و حلقه تکرار می‌گردد. حالا hi=2 و lo=0 است و دوباره عنصر وسط آرایهٔ a[0..2] یعنی a[1] با x مقایسه می‌شود. a[1] برابر با 33 است که کوچک‌تر از x می‌باشد. پس این دفعه lo برابر با i+1 یعنی 2 می‌شود. در سومین دور حلقه، hi=2 و lo=2 است. پس عنصر وسط آرایهٔ a[2..2] که همان a[2] است با x مقایسه می‌شود. a[2] برابر با 44 است که با x برابر است. پس مقدار 2 بازگشت داده می‌شود؛ یعنی x مورد نظر در a[2] وجود دارد.

lo	hi	i	a[i]	??	x
0	6	3	55	>	44
	2	1	33	<	44
2		2	44	==	44

حال فراخوانی index(60,a,7) را دنبال می‌کنیم. وقتی حلقه شروع می‌شود، x=60 و n=7 و lo=0 و hi=6 است. عنصر وسط آرایهٔ a[0..6] عنصر a[3]=55 است که از x کوچک‌تر است. پس lo برابر با i+1=4 می‌شود و حلقه دوباره تکرار می‌شود. این دفعه hi=6 و lo=4 است . عنصر وسط آرایهٔ a[4..6] عنصر a[5]=77 است که بزرگ‌تر از x می‌باشد. پس hi به i-1=4 تغییر می‌یابد و دوباره حلقه تکرار می‌شود. این بار hi=4 و lo=4 است و عنصر وسط آرایهٔ a[4..4] عنصر a[4]=66 است که بزرگ‌تر از x می‌باشد. لذا hi به i-1=3 کاهش می‌یابد. اکنون شرط حلقه غلط می‌شود زیرا hi<lo است. بنابراین تابع مقدار 7 را برمی‌گرداند یعنی عنصر مورد نظر در آرایه موجود نیست.

lo	hi	i	a[i]	??	x
0	6	3	55	<	60
4		5	77	>	60
	4	4	66	>	60

در تابع فوق هر بار که حلقه تکرار می‌شود، محدودهٔ جستجو 50٪ کوچک‌تر می‌شود. در آرایهٔ n عنصری، روش جستجوی دودویی حداکثر به $\log_2 n+1$ مقایسه نیاز دارد تا به پاسخ برسد. حال آن که در روش جستجوی خطی به n مقایسه نیاز است. برای مثال در یک آرایه 100 عنصری برای مشخص شدن این که مقدار مورد نظر در آرایه هست یا خیر در روش جستجوی دودویی به $\log_2 100+1=7.64$ مقایسه نیاز است. یعنی حداکثر با 8 مقایسه به پاسخ می‌رسیم ولی در روش جستجوی خطی روی همین آرایه به حداکثر 100 مقایسه نیاز داریم. پس جستجوی دودویی سریع‌تر از جستجوی خطی است. دومین تفاوت در این است که اگر چند عنصر دارای مقادیر یکسانی باشند، آنگاه جستجوی خطی همیشه کوچک‌ترین ایندکس را برمی‌گرداند ولی در مورد جستجوی دودویی نمی‌توان گفت که کدام ایندکس بازگردانده می‌شود. سومین فرق در این است که جستجوی دودویی فقط روی آرایه‌های مرتب کارایی دارد و اگر آرایه‌ای مرتب نباشد، جستجوی دودویی پاسخ غلط می‌دهد ولی جستجوی خطی همیشه پاسخ صحیح خواهد داد.

× مثال 15-6 مشخص کردن این که آیا آرایه مرتب است یا خیر

برنامهٔ زیر یک تابع بولی را آزمایش می‌کند. این تابع مشخص می‌نماید که آیا آرایهٔ داده شده غیر نزولی است یا خیر:

```
bool isNondecreasing(int a[], int n);
int main()
{  int a[] = { 22, 44, 66, 88, 44, 66, 55};
   cout << "isNondecreasing(a,4) = " << isNondecreasing(a,4)
        << endl;
   cout << "isNondecreasing(a,7) = " << isNondecreasing(a,7)
        << endl;
}
bool isNondecreasing(int a[], int n)
{  // returns true iff a[0] <= a[1] <= ... <= a[n-1]:
   for (int i=1; i<n; i++)
      if (a[i]<a[i-1]) return false;
   return true;
}
```

```
isNondecreasing(a,4) = 1
isNondecreasing(a,7) = 0
```

این تابع یک بار کل آرایه را پیمایش کرده و زوج‌های a[i-1] و a[i] را مقایسه می‌کند. اگر زوجی یافت شود که در آن a[i]<a[i-1] باشد، مقدار false را بر می‌گرداند به این معنی که آرایه مرتب نیست. ببینید که مقادیر true و false به شکل اعداد 1 و 0 در خروجی چاپ می‌شوند زیرا مقادیر بولی در حقیقت به شکل اعداد صحیح در حافظه ذخیره می‌شوند.

اگر ***پیش‌شرط***[1] مثال 14-6 یعنی مرتب بودن آرایه رعایت نشود، جستجوی دودویی پاسخ درستی نمی‌دهد. به این منظور ابتدا باید این پیش‌شرط بررسی شود. با استفاده از تابع assert() می‌توان اجرای یک برنامه را به یک شرط وابسته کرد. این تابع یک آرگومان بولی می‌پذیرد. اگر مقدار آرگومان false باشد، برنامه را خاتمه داده و موضوع را به سیستم عامل گزارش می‌کند. اگر مقدار آرگومان true باشد، برنامه بدون تغییر ادامه می‌یابد. تابع asset() در سرفایل <cassert> تعریف شده است.

× مثال 16-6 استفاده از تابع assert() برای رعایت کردن یک پیش‌شرط

برنامهٔ زیر نسخهٔ بهبودیافته‌ای از تابع search() مثال 14-6 را آزمایش می‌کند. در این نسخه، از تابع isNonDecreasing() مثال 15-6 استفاده شده تا مشخص شود آرایه مرتب است یا خیر. نتیجه این تابع به تابع assert() ارسال می‌گردد تا اگر آرایه مرتب نباشد برنامه به بیراهه نرود:

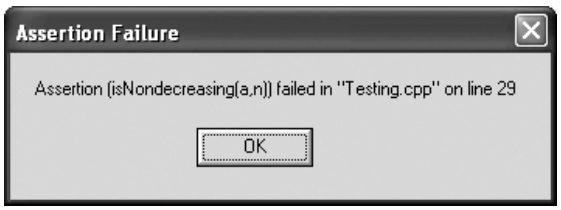
```
#include <cassert>    // defines the assert() function
#include <iostream>   // defines the cout object
using namespace std;
int index(int x, int a[], int n);
int main()
{  int a[] = { 22, 33, 44, 55, 66, 77, 88, 60 };
   cout << "index(44,a,7) = " << index(44,a,7) << endl;
   cout << "index(44,a,8) = " << index(44,a,8) << endl;
```

1 - Precondition

```
   cout << "index(60,a,8) = " << index(60,a,8) << endl;
}
bool isNondecreasing(int a[], int n);
int index(int x, int a[], int n)
{  // PRECONDITION: a[0] <= a[1] <= ... <= a[n-1];
   // binary search:
   assert(isNondecreasing(a,n));
   int lo=0, hi=n-1, i;
   while (lo <= hi)
   {  i = (lo + hi)/2;
      if (a[i] == x) return i;
      if (a[i] < x) lo = i+1;   // continue search in a[i+1..hi]
      else hi = i-1;            // continue search in a[0..n-1]
   }
 return n;    // x was not found in a[0..n-1]
}
```

```
index(44,a,7) = 2
```

آرایهٔ a[] که در این برنامه استفاده شده کاملا مرتب نیست اما هفت عنصر اول آن مرتب است. بنابراین در فراخوانی index(44,a,7) تابع بولی مقدار true را به assert() ارسال می‌کند و برنامه ادمه می‌یابد. اما در دومین فراخوانی index(44,a,8) باعث می‌شود که تابع isNondecreasing() مقدار false را به تابع assert() ارسال کند که در این صورت برنامه متوقف می‌شود و ویندوز پنجرهٔ هشدار مقابل را نمایش می‌دهد.



6-9 استفاده از انواع شمارشی در آرایه

انواع شمارشی در فصل دوم توضیح داده شده‌اند. با استفاده از انواع شمارشی نیز می‌توان آرایه‌ها را پردازش نمود.

× مثال 17-6 شمارش با استفاده از روزهای هفته

این برنامه یک آرایه به نام high[] با هفت عنصر از نوع float تعریف می‌کند که هر عنصر حداکثر دما در یک روز هفته را نشان می‌دهد:

```
int main()
{  enum Day { SUN, MON, TUE, WED, THU, FRI, SAT };
   float high[SAT+1] = {28.6, 29.1, 29.9, 31.3, 30.4, 32.0, 30.7};
   for (int day = SUN; day <= SAT; day++)
   cout << "The high temperature for day " << day << " was "
        << high[day] << endl;
}
```

```
The high temperature for day 0 was 28.6
The high temperature for day 1 was 29.1
The high temperature for day 2 was 29.9
The high temperature for day 3 was 31.3
The high temperature for day 4 was 30.4
The high temperature for day 5 was 32.0
The high temperature for day 6 was 30.7
```

به خاطر بیاورید که انواع شمارشی به شکل مقادیر عددی ذخیره می‌شوند. اندازهٔ آرایه، SAT+1 است زیرا SAT مقدار صحیح 6 را دارد و آرایه به هفت عنصر نیازمند است. متغیر day از نوع int است پس می‌توان مقادیر Day را به آن تخصیص داد. استفاده از انواع شمارشی در برخی از برنامه‌ها باعث می‌شود که کد برنامه «خود استناد» شود. مثلا در مثال 17-6 کنترل حلقه به شکل

```
for (int day = SUN; day <= SAT; day++)
```

باعث می‌شود که هر بیننده‌ای حلقهٔ for بالا را به خوبی درک کند.

10-6 تعریف انواع

انواع شمارشی یکی از راه‌هایی است که کاربر می‌تواند نوع ساخت خودش را تعریف کند. برای مثال دستور زیر :

```
enum Color { RED, ORANGE, YELLOW, GREEN, BLUE, VIOLET };
```

یک نوع جدید به نام Color تعریف می‌کند که متغیرهایی از این نوع می‌توانند مقادیر RED یا ORANGE یا YELLOW یا GREEN یا BLUE یا VIOLET را داشته باشند. پس با استفاده از این نوع می‌توان متغیرهایی به شکل زیر تعریف نمود:

```
Color shirt = BLUE;
Color car[] = { GREEN, RED, BLUE, RED };
float wavelength[VIOLET+1] = {420, 480, 530, 570, 600, 620};
```

در این‌جا shirt متغیری از نوع Color است و با مقدار BLUE مقداردهی شده. car یک آرایهٔ چهار عنصری است و مقدار عناصر آن به ترتیب GREEN و RED و BLUE و RED می‌باشد. همچنین wavelength آرایه‌ای از نوع float است که دارای VIOLET+1 عنصر یعنی 5+1=6 عنصر است.

در C++ می‌توان نام انواع استاندارد را تغییر داد. کلمهٔ کلیدی typedef یک نام مستعار برای یک نوع استاندارد موجود تعریف می‌کند. نحو استفاده از آن به شکل زیر است:

```
typedef type alias;
```

که *type* یک نوع استاندارد و alias نام مستعار برای آن است. برای مثال کسانی که با پاسکال برنامه می‌نویسند به جای نوع long از عبارت Integer استفاده می‌کنند و به جای نوع double از عبارت Real استفاده می‌نمایند. این افراد می‌توانند به شکل زیر از نام مستعار استفاده کنند:

```
typedef long Integer;
typedef double Real;
```

و پس از آن کدهای زیر معتبر خواهند بود:

```
Integer n = 22;
const Real PI = 3.141592653589793;
Integer frequency[64];
```

اگر دستور typedef را به شکل زیر بکار ببریم می‌توانیم آرایه‌ها را بدون علامت براکت تعریف کنیم:

```
typedef element-type alias[];
```

مثل تعریف زیر :

```
typedef float sequence[];
```

سپس می‌توانیم آرایهٔ a را به شکل زیر اعلان کنیم:

```
sequence a = {55.5, 22.2, 99.9};
```

دستور typedef نوع جدیدی را اعلان نمی‌کند، بلکه فقط به یک نوع موجود نام مستعاری را نسبت می‌دهد. مثال بعدی نحوهٔ به کارگیری typedef را نشان می‌دهد.

× مثال 18-6 دوباره مرتب‌سازی حبابی

برنامهٔ زیر همان برنامهٔ مثال 13-6 است با این فرق که از typedef استفاده شده تا بتوان از نام مستعار sequrnce به عنوان یک نوع استفاده کرد. سپس این نوع در فهرست پارامترها و اعلان a در تابع main() به کار رفته است:

```
typedef float Sequence[];
void sort(Sequence,int);
void print(Sequence,int);
int main()
{  Sequence a = {55.5, 22.2, 99.9, 66.6, 44.4, 88.8, 33.3, 77.7};
   print(a,8);
   sort(a,8);
   print(a,8);
}
void sort(Sequence a, int n)
{  for (int i=n-1; i>0; i--)
      for (int j=0; j<i; j++)
      if (a[j] > a[j+1]) swap(a[j],a[j+1]);
}
```

دوباره به دستور typedef نگاه کنید:

```
typedef float Seguence[];
```

علامت براکت‌ها [] نشان می‌دهند که هر چیزی که از نوع Sequence تعریف شود، یک آرایه است و عبارت float نیز بیان می‌کند که این آرایه از نوع float است.

6-11 آرایه‌های چند بعدی

همهٔ آرایه‌هایی که تاکنون تعریف کردیم، یک بعدی هستند، خطی هستند، رشته‌ای هستند. می‌توانیم آرایه‌ای تعریف کنیم که از نوع آرایه باشد، یعنی هر خانه از آن آرایه، خود یک آرایه باشد. به این قبیل آرایه‌ها، ***آرایه‌های چندبعدی***[1] می‌گوییم. یک آرایهٔ دو بعدی آرایه‌ای است که هر خانه از آن، خود یک آرایهٔ یک بعدی باشد. یک آرایهٔ سه بعدی آرایه‌ای است که هر خانه از آن یک آرایهٔ دو بعدی باشد.

دستور int a[5]; آرایه‌ای با پنج عنصر از نوع int تعریف می‌کند. این یک آرایهٔ یک بعدی است. دستور int a[3][5]; آرایه‌ای با سه عنصر تعریف می‌کند که هر عنصر، خود یک آرایهٔ پنج عنصری از نوع int است. این یک آرایهٔ دو بعدی است که در مجموع پانزده عضو دارد. دستور int a[2][3][5]; آرایه‌ای با دو عنصر تعریف می‌کند که هر عنصر، سه آرایه است که هر آرایه پنج عضو از نوع int دارد. این یک آرایهٔ سه بعدی است که در مجموع سی عضو دارد. به همین ترتیب می‌توان آرایه‌های چند بعدی تعریف نمود.

شکل دستیابی به عناصر در آرایه‌های چند بعدی مانند آرایه‌های یک بعدی است. مثلا دستور

```
a[1][2][3] = 99;
```

مقدار 99 را در عنصری قرار می‌دهد که ایندکس آن عنصر (1,2,3) است.

آرایه‌های چند بعدی مثل آرایه‌های یک بعدی به توابع فرستاده می‌شوند با این تفاوت که هنگام اعلان و تعریف تابع مربوطه، باید تعداد عناصر بعد دوم تا بعد آخر حتما ذکر شود.

---

1 – Multi dimensional arrays

× مثال 19-6 نوشتن و خواندن یک آرایهٔ دو بعدی

برنامهٔ زیر نشان می‌دهد که یک آرایهٔ دوبعدی چگونه پردازش می‌شود:

```
void read(int a[][5]);
void print(int a[][5]);
int main()
{  int a[3][5];
   read(a);
   print(a);
}
void read(int a[][5])
{  cout << "Enter 15 integers, 5 per row:\n";
   for (int i=0; i<3; i++)
   {  cout << "ROW " << i << ": ";
      for (int j=0; j<5; j++)
      cin >> a[i][j];
   }
}
void print(const int a[][5])
{  for (int i=0; i<3; i++)
   {  for (int j=0; j<5; j++)
         cout << " " << a[i][j];
      cout << endl;
   }
}
```

```
Enter 15 integers, 5 per row:
row 0: 44 77 33 11 44
row 1: 60 50 30 90 70
row 2: 65 25 45 45 55
 44 77 33 11 44
 60 50 30 90 70
 65 25 45 45 55
```

دقت کنید که در فهرست پارامترهای توابع بالا، بعد اول نامشخص است اما بعد دوم مشخص شده. علت هم این است که آرایهٔ دو بعدی a[][] در حقیقت آرایه‌ای یک‌بعدی از سه آرایهٔ پنج عنصری است. کامپایلر نیاز ندارد بداند که چه تعداد از این آرایه‌های پنج عنصری موجود است، اما باید بداند که آن‌ها پنج عنصری هستند.

وقتی یک آرایهٔ چند بعدی به تابع ارسال می‌شود، بعد اول مشخص نیست اما همهٔ ابعاد دیگر باید مشخص باشند.

× مثال 20-6 پردازش یک آرایهٔ دوبعدی از نمرات امتحانی

```
const NUM_STUDENTS = 3;
const NUM_QUIZZES = 5;
typedef int Score[NUM_STUDENTS][NUM_QUIZZES];
void read(Score);
void printQuizAverages(Score);
void printClassAverages(Score);
int main()
{  Score score;
   cout << "Enter " << NUM_QUIZZES
        << " quiz scores for each student:\n";
   read(score);
   cout << "The quiz averages are:\n";
   printQuizAverages(score);
   cout << "The class averages are:\n";
   printClassAverages(score);
}
void read(Score score)
{  for (int s=0; s<NUM_STUDENTS; s++)
   {  cout << "Student " << s << ": ";
      for (int q=0; q<NUM_QUIZZES; q++)
         cin >> score[s][q];
  }
}
void printQuizAverages(Score score)
{  for (int s=0; s<NUM_STUDENTS; s++)
   {  float sum = 0.0;
      for (int q=0; q<NUM_QUIZZES; q++)
         sum += score[s][q];
      cout << "\tStudent " << s << ": " << sum/NUM_QUIZZES
           << endl;
   }
```

```
}
void printClassAverages(Score score)
{  for (int q=0; q<NUM_QUIZZES; q++)
   {  float sum = 0.0;
      for (int s=0; s<NUM_STUDENTS; s++)
         sum += score[s][q];
      cout << "\tQuiz " << q << ": " << sum/NUM_STUDENTS
           << endl;
   }
}
```

```
Enter 5 quiz scores for each student:
student 0: 8 7 9 8 9
student 1: 9 9 9 9 8
student 2: 5 6 7 8 9
The quize averages are:
  student 0: 8.2
  student 1: 8.8
  student 2: 7
The class averages are:
  Quiz 0: 7.33333
  Quiz 1: 7.33333
  Quiz 2: 8.33333
  Quiz 3: 8.33333
  Quiz 4: 8.66667
```

در برنامهٔ فوق با استفاده از دستور typedef برای آرایه‌های دوبعدی 5*3 نام مستعار Score انتخاب شده. این باعث می‌شود که توابع خواناتر باشند. هر تابع از دو حلقهٔ for تودرتو استفاده کرده که حلقهٔ بیرونی، بعد اول را پیمایش می‌کند و حلقهٔ درونی بعد دوم را پیمایش می نماید.

تابع printQuizAverages() میانگین هر سطر از نمرات را محاسبه و چاپ می‌نماید و تابع printClassAverages() میانگین هر ستون از نمره‌ها را چاپ می‌کند.

× مثال 21-6 پردازش یک آرایهٔ سه بعدی

این برنامه تعداد صفرها را در یک آرایهٔ سه بعدی می‌شمارد:

```
int numZeros(int a[][4][3], int n1, int n2, int n3);
int main()
{  int a[2][4][3] = { { {5,0,2}, {0,0,9}, {4,1,0}, {7,7,7} },
                     { {3,0,0}, {8,5,0}, {0,0,0}, {2,0,9} } };
   cout << "This array has " << numZeros(a,2,4,3)
        << " zeros:\n";
}
int numZeros(int a[][4][3], int n1, int n2, int n3)
{  int count = 0;
   for (int i = 0; i < n1; i++)
      for (int j = 0; j < n2; j++)
         for (int k = 0; k < n3; k++)
            if (a[i][j][k] == 0) ++count;
   return count;
}
```

```
This array has 11 zeros:
```

توجه کنید که آرایه چگونه مقداردهی شده است. این قالب مقداردهی به خوبی نمایان می‌کند که آرایهٔ مذکور یک آرایه دو عنصری است که هر عنصر، خود یک آرایهٔ چهار عضوی است که هر عضو شامل آرایه‌ای سه عنصری می‌باشد. پس این آرایه در مجموع 24 عنصر دارد. آرایهٔ مذکور را به شکل زیر نیز می‌توانیم مقداردهی کنیم:

```
int a[2][4][3]={5,0,2,0,0,9,4,1,0,7,7,7,3,0,0,8,5,0,0,0,0,2,0,9};
```

و یا مانند این:

```
int a[2][4][3] =
{{5,0,2,0,0,9,4,1,0,7,7,7},{3,0,0,8,5,0,0,0,0,2,0,9}};
```

هر سهٔ این قالب‌ها برای کامپایلر یک مفهوم را دارند اما با نگاه کردن به دو قالب اخیر به سختی می‌توان فهمید که کدام عنصر از آرایه، کدام مقدار را خواهد داشت.

به سه حلقهٔ for تودرتو دقت کنید. به طور کلی برای پیمایش یک آرایهٔ n بعدی به n حلقهٔ تودرتو نیاز است.

پرسش‌های گزینه‌ای

1 – کدام گزینه اشتباه است؟

الف – یک آرایه مجموعه‌ای از متغیرهاست که همگی یک نوع دارند

ب – می‌توان آرایه را از قسمتی از حافظه به قسمت دیگر منتقل نمود

ج – محل قرارگیری عناصر آرایه در حافظه، پشت سر هم و پیوسته است

د – آرایه را می‌توان برای دست‌کاری به تابع ارسال کرد

2 – در مورد دستور `int a[5];` کدام گزینه اشتباه است؟

الف – این دستور یک آرایهٔ یک بعدی را اعلان می‌کند

ب – اعضای این آرایه از نوع عددی صحیح هستند

ج – این آرایه پنج عضو دارد

د – هر عضو دارای مقدار پیش‌فرض صفر است

3 – کد `int a[] = {0, 0, 0};` معادل کدام گزینهٔ زیر است؟

الف – `int a[] = {0};` ب – `int a[0];`

ج – `int a[0,0,0];` د – `int a[3] = {0};`

4 – در مورد دستور `int a[5] = {1, 2, 3};` کدام گزینه صحیح است؟

الف – عناصر چهارم و پنجم آرایه a دارای مقدار صفر هستند.

ب – آرایهٔ a سه عنصر دارد

ج – دو عنصر اول آرایه a دارای مقدار زباله هستند

د – در این دستور پنج آرایهٔ سه عنصری اعلان شده است

5 – در مورد دستور `float a[2] = {1.11, 2.22, 3.33};` کدام گزینه صحیح است؟

الف – آرایهٔ a سه عنصر از نوع `float` دارد

ب – هر عضو آرایهٔ a باید دو رقم اعشار داشته باشد

ج – عنصر سوم آرایهٔ a مقدار پیش‌فرض صفر دارد

د – این دستور اشتباه است زیرا فهرست مقداردهی بیش از عناصر آرایه عضو دارد

6 – اگر ایندکس آرایه از تعداد اعضای آن بیشتر شود آنگاه:

الف – ممکن است برنامه متوقف شود زیرا اثر همسایگی رخ می‌دهد

ب – کامپایلر خطا می‌گیرد و برنامه اصلا اجرا نمی‌شود

ج – سیستم عامل خطا می‌گیرد و برنامه متوقف می‌شود

د – در زمان اجرا به تعداد اعضای آرایه اضافه می‌شود تا به اندازه ایندکس برسد

7 – اگر آرایهٔ a از نوع int و با پنج عنصر تعریف شده باشد، آنگاه کد float b[] = a; چه اثری دارد؟

الف – این کد اشتباه است زیرا آرایه‌ها را نمی‌توان به یکدیگر تخصیص داد

ب – مقادیر آرایهٔ a به نوع float ارتقا یافته و سپس درون آرایهٔ b قرار می‌گیرد

ج – آرایهٔ b با پنج عنصر ایجاد می‌شود بدون این که اعضای آرایهٔ a در آن کپی شود

د – آرایهٔ b با پنج عنصر ایجاد می‌شود و اعضای آرایهٔ a درون آن کپی می‌شود

8 – دستور int a[2][4]; چه عملی انجام می‌دهد؟

الف – یک آرایهٔ دو بعدی تعریف می‌کند که این آرایه در کل 8 عنصر دارد

ب – یک آرایهٔ دو بعدی تعریف می‌کند که این آرایه در کل 6 عنصر دارد

ج – یک آرایهٔ دو بعدی تعریف می‌کند که مقدار پیش‌فرض عناصر بعد اول، مقدار 2 و مقدار پیش‌فرض عناصر بعد دوم ، مقدار 4 است

د – یک آرایهٔ یک بعدی با دو عضو تعریف می‌کند که مقدار عضو اول 2 و مقدار عضو دوم 4 است

9 – اگر a یک آرایه باشد، آنگاه با اجرای کد cout << a; چه رخ می‌دهد؟

الف – مقدار اعضای آرایهٔ a در خروجی چاپ می‌شود

ب – تعداد اعضای آرایهٔ a در خروجی چاپ می‌شود

ج – آدرس اولین خانهٔ آرایهٔ a در خروجی چاپ می‌شود

د – سیستم عامل خطا می‌گیرد و پیغام خطا در خروجی چاپ می‌شود

10 – فرض کنید تابع print() فقط یک پارامتر دارد و آن هم از نوع آرایه است. اگر آرایه a یک آرایهٔ سه عنصری باشد، آنگاه کدام یک از دستورهای زیر آرایهٔ a را به تابع print() می‌فرستد؟

الف – print(a[3]); ب – print(a[]);

ج – `print(a);` د – `a.print();`

11 – آرایهٔ `c` به شکل `int c[2][3][4];` تعریف شده است. تابع `print()` از نوع `void` بوده و قرار است که آرایهٔ `c` به آن ارسال شود. با این توضیحات، تابع `print()` باید چگونه اعلان شود؟

الف - `void print(int [][][]);`

ب – `void print(int [2][3][4]);`

ج - `void print(int[][3][4]);`

د – `void print(int, int, int);`

12 – کدام دستور مقدار اولین عنصر آرایهٔ `a` را چاپ می‌کند؟

الف - `cout << a[0];` ب – `cout << a[1];`

ج – `cout << a;` د – `cout << [a0];`

13 – اگر آرایهٔ `a` دارای پنج عنصر از نوع `int` باشد آنگاه کدام دستور مقدار 10 را درون آخرین عضو این آرایه قرار می‌دهد؟

الف - `a[5] = 10;` ب – `a[4] = 10;`

ج - `a(5) = 10;` د – `a(4) = 10;`

14 – اگر آرایهٔ `a` به شکل `int a[5][5][5];` تعریف شده باشد آنگاه کد `a[5][5][5] = 0;` چه اثری دارد؟

الف – آخرین عضو آرایهٔ a را صفر می‌کند

ب – عضوی که در محل (5, 5, 5) از آرایهٔ a قرار گرفته را صفر می‌کند

ج – همهٔ اعضای آرایهٔ a را صفر می‌کند

د – اثری روی اعضای آرایهٔ a ندارد

15 – دستور `typedef` چه کاری انجام می‌دهد؟

الف – یک نوع جدید تعریف می‌کند

ب – یک متغیر جدید تعریف می‌کند

ج – برای یک نوع موجود، نام مستعار تعریف می‌کند

د – برای یک متغیر موجود، نام مستعار تعریف می‌کند

16 – برای پیمایش آرایه‌ای که n بعد و در هر بعد k عضو دارد به چند حلقه نیازمندیم؟

الف – n حلقه ب – k حلقه ج – n-k حلقه د – n+k حلقه

17 – در مورد ارسال آرایه‌ها به تابع، کدام گزینه صحیح نیست؟

الف – آدرس اولین عنصر آرایه به تابع فرستاده می‌شود

ب – تابع می‌تواند با توجه به نوع عناصر آرایه و آدرس اولین عنصر آن، به تک تک عناصر آرایه دسترسی داشته باشد

ج – در حقیقت آرایه‌ها به طریق ارجاع به تابع ارسال می‌شوند

د – لازم نیست تابع چیزی راجع به ابعاد آرایه بداند

18 – به مقداری که باید به آدرس عنصر اول آرایه اضافه شود تا به یک عنصر مفروض برسیم چه می‌گویند؟

الف – آفست ب – ایندکس ج – بعد د – فاصله

پرسش‌های تشریحی

1- اعضای یک آرایه چند نوع متفاوت می‌توانند داشته باشند؟
2- ایندکس آرایه چه نوع و محدوده‌ای باید داشته باشد؟
3- اگر یک آرایه اعلان شده باشد ولی مقداردهی نشده باشد، عناصر آن آرایه چه مقادیری خواهند داشت؟
4- اگر یک آرایه اعلان شده باشد ولی فهرست مقداردهی آن نسبت به اعضای آرایه تعداد کم‌تری داشته باشد آنگاه عناصر آن آرایه چه مقادیری خواهند داشت؟
5- اگر در فهرست مقداردهی یک آرایه، عناصر بیشتری نسبت به اندازهٔ آرایه وجود داشته باشد چه اتفاقی می‌افتد؟
6- بین دستور `enum` و دستور `typedef` چه تفاوتی وجود دارد؟
7- اگر بخواهیم یک آرایه را به یک تابع ارسال کنیم، چرا باید اندازهٔ همه ابعاد به غیر از بعد اول در فهرست پارامترهای تابع ذکر شود؟

تمرین‌های برنامه‌نویسی

1- برنامهٔ مثال 1-6 را طوری تغییر دهید که برای هر ورودی، یک خط درخواست و برای هر خروجی، یک خط اعلان چاپ شود. مانند تصویر زیر:

```
Enter 5 numbers
  a[0]: 11.11
  a[1]: 33.33
  a[2]: 55.55
  a[3]: 77.77
  a[4]: 99.99
In reverse order, they are:
  a[4] = 99.99
  a[3] = 77.77
  a[2] = 55.55
  a[1] = 33.33
  a[0] = 11.11
```

2- برنامهٔ مثال 1-6 را طوری تغییر دهید که آرایه را به طور معکوس پر کند و سپس اعضای آرایه را به همان ترتیبی که ذخیره شده‌اند چاپ کند. مانند تصویر زیر:

```
Enter 5 numbers:
  a[4]: 55.55
  a[3]: 66.66
  a[2]: 77.77
  a[1]: 88.88
  a[0]: 99.99
In reverse order, they are:
  a[0] = 99.99
  a[1] = 88.88
  a[2] = 77.77
  a[3] = 66.66
  a[4] = 55.55
```

3- برنامهٔ مثال 9-6 را طوری تغییر دهید که با استفاده از تابع زیر، میانگین n عنصر اول آرایه را برگرداند:

```
float ave(int[] a, int n);
   // returns the average of the first n elements of a[]
```

4- برنامهٔ مثال 10-6 را طوری تغییر دهید که خود آرایه، مجموع آن و میانگین آن را چاپ کند. (به مثال 9-6 و مسالهٔ 3-6 نگاه کنید)

5- برنامهٔ مثال 11-6 را طوری تغییر دهید که برای هر عنصر آرایه، آدرس حافظهٔ آن و محتویات آن را چاپ کند. برای آرایه‌ای به نام a، از عبارت‌های a و a+1 و a+2 و... استفاده کنید تا آدرس‌های a[0] و a[1] و... را بدست آورید و از عبارت‌های a* و (a+1)* و (a+2)* و... استفاده کنید تا محتویات این مکان‌ها را بدست آورید. آرایه را به صورت unsigned int a[] اعلان کنید تا مقادیر آرایه وقتی داخل جریان cout درج می‌شوند، به شکل اعداد صحیح چاپ شوند.

6- برنامهٔ مثال 12-6 را طوری تغییر دهید که به جای اولین محل قرار گرفتن یک عنصر مفروض، آخرین محل قرار گرفتن آن را در آرایه برگرداند.

7- برنامهٔ مثال 15-6 را طوری تغییر دهید که مقدار true را برگرداند اگر و فقط اگر آرایه غیر صعودی باشد.

8- تابع زیر را نوشته و آزمایش کنید. این تابع در بین n عنصر اول آرایه، مقدار کمینه(مینیمم) را برمی‌گرداند.

```
float min(float a[], int n);
```

9- تابع زیر را نوشته و آزمایش کنید. این تابع، ایندکس اولین مقدار کمینه(مینیمم) را از میان n عنصر اول آرایهٔ مفروض برمی‌گرداند.

```
int minIndex(float a[], int n);
```

10- تابع زیر را نوشته و آزمایش کنید. این تابع مقدار کمینه و بیشینه را در بین n عنصر اول آرایهٔ مفروض با استفاده از پارامترهای ارجاعی‌اش برمی‌گرداند.

```
void getEXtremes(float& min, float& max, float a[], int n);
```

11- تابع زیر را نوشته و آزمایش کنید. این تابع بزرگ‌ترین مقدار و دومین بزرگ‌ترین مقدار (این دو می‌توانند مساوی باشند) را از بین n عنصر اول آرایهٔ مفروض با استفاده از پارامترهای ارجاعی‌اش برمی‌گرداند.

```
void largest(float& max1, float& max2, float a[], int n);
```

12-تابع زیر که یک مقدار را از آرایه حذف می‌کند، نوشته و آزمایش کنید:

```
void remove(float a[], int& n, int i);
```

تابع بالا به این روش عنصر a[i] را حذف می‌کند که تمام عناصر بعد از آن را یک پله به عقب می‌کشد و n را کاهش می‌دهد.

13- تابع زیر را نوشته و آزمایش کنید. این تابع سعی می‌کند یک عنصر را از آرایه حذف کند:

```
bool removeFirst(float a[], int& n, float x);
```

این تابع در بین n عنصر اول آرایهٔ a[] به دنبال x می‌گردد. اگر x پیدا شود آنگاه اولین محلی که x در آن واقع شده، حذف می‌شود و تمام عناصر بعدی یک پله به عقب کشیده می‌شوند و n نیز یک واحد کاهش می‌یابد و مقدار true نیز بازگشت داده می‌شود تا مشخص کند که حذف موفقیت‌آمیز بوده است. اگر x پیدا نشود، آرایه بدون تغییر می‌ماند و مقدار false برگردانده می‌شود. (به مسالهٔ 12 نگاه کنید)

14- تابع زیر را نوشته و آزمایش کنید. این تابع عناصری را از آرایه حذف می‌کند:

```
void removeAll(float a[], int& n, float x);
```

تابع مذکور همهٔ عناصری که با x برابرند را از n عنصر اول آرایه حذف می‌کند و n را به تعداد عناصر حذف شده، کاهش می‌دهد. (به مسالهٔ 13 نگاه کنید)

15- تابع زیر را نوشته و آزمایش کنید:

```
void rotate(int a[], int n, int k);
```

تابع n عنصر اول آرایه a را ،k موقعیت به راست (یا اگر k منفی باشد k موقعیت به چپ) منتقل می‌کند. kعنصر آخر به شروع آرایه منتقل می‌شوند. برای مثال، فراخوانی rotate(a,8,3) آرایهٔ {22,33,44,55,66,77,88,99} را به آرایهٔ {77,88,99,22,33,44,55,66} تبدیل می‌کند. بدیهی است که فراخوانی rotate(a,8,-5) تاثیر مشابهی خواهد داشت.

16- تابع زیر را نوشته و آزمایش کنید:

```
void append(int a[], int m, int b[], int n);
```

تابع بالا n عنصر اول آرایهٔ b را به انتهای m عنصر اول آرایهٔ a الحاق می‌کند. فرض بر این است که آرایهٔ a حداقل به اندازهٔ m+n عنصر جا دارد. برای مثال اگر آرایهٔ a برابر با {22,33,44,55,66,77,88,99} و آرایهٔ b نیز برابر با {20,30,40,50,60,70,80} باشد، آنگاه فراخوانی append(a,5,b,3) باعث می‌شود که آرای ۀ a به شکل {22,33,44,55,66,20,30,40} تغییر کند. توجه داشته باشید که آرایهٔ b تغییر نمی‌کند.

17- تابع زیر را نوشته و آزمایش کنید:

```
void insert(float a[], int& n, float x)
```

این تابع درون آرایهٔ مرتب a که n عنصری است مقدار x را درج می‌کند و n را افزایش می‌دهد. عنصر جدید در مکانی درج می‌شود که ترتیب آرایه حفظ شود. به این منظور، عناصر باید به جلو منتقل شوند تا برای عضو جدید، جا باز شود. (به همین دلیل آرایه باید حداقل n+1 عنصر داشته باشد)

18- تابع زیر را نوشته و آزمایش کنید:

```
int frequency(float a[], int n, int x);
```

این تابع دفعاتی را که عنصر x در میان n عنصر اول آرایه ظاهر می‌شود را شمرده و نتیجه را به عنوان تعداد تکرار x در a برمی‌گرداند.

19- تابع زیر را نوشته و آزمایش کنید:

```
void reverse(int a[] , int n);
```

تابع فوق n عنصر اول آرایه را معکوس می‌کند. برای مثال فراخوانی reverse(a,5) آرایهٔ {22,33,44,55,66} را به {66,55,44,33,22} تبدیل می‌کند.

20- تابع زیر را نوشته و آزمایش کنید:

```
void add(float a[], int n, float b[]);
```

تابع مذکور n عنصر اول b را به n عنصر متناظر در a اضافه می‌کند. برای مثال اگر a برابر با {2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9} باشد و b نیز برابر با {6.0,5.0,4.0,3.0,2.0,1.0} باشد، آنگاه فراخوانی add(a,5,b); باعث می‌شود که آرای ۀ a به {8.2,8.3,8.4,8.5,8.6,7.7,8.8,9.9} تبدیل شود.

21- تابع زیر را نوشته و آزمایش کنید:

```
float outerProduct(float p[][3], float a[], float b[]);
```

تابع بالا، حاصل ضرب بیرونی سه عنصر اول a با سه عنصر اول b را برمی‌گرداند. برای مثال اگر a برابر با {2.2,3.3,4.4} و b برابر با {2.0,-1.0,0.0} باشد، فراخوانی ;(outerProduct(p,a,b باعث می‌شود آرای ۀ دو بعدی p به صورت زیر تبدیل شود:

```
4.4  -2.2  0.0
6.6  -3.3  0.0
8.8  -4.4  0.0
```

عنصر p[i][j] حاصل ضرب a[i] با b[j] است.

22- تابعی را نوشته و آزمایش کنید که عناصر یک آرای ۀ دوبعدی مربعی را 90 درجه در جهت عقربه‌های ساعت بچرخاند. برای مثال این تابع باید آرای ۀ:

```
11  22  33
44  55  66
77  88  99
```

را به آرای ۀ

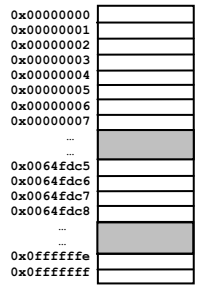
```
77  44  11
88  55  22
99  66  33
```

تبدیل می‌کند.

فصل هفتم

«اشاره‌گرها[1] و ارجاع‌ها[2]»

7-1 مقدمه



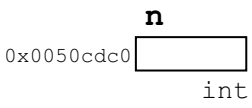
حافظهٔ رایانه را می‌توان به صورت یک آرایهٔ بزرگ در نظر گرفت. برای مثال رایانه‌ای با 256 مگابایت RAM در حقیقت حاوی آرایه‌ای به اندازهٔ 268،435،456 ($=2^{28}$) خانه است که اندازهٔ هر خانه یک بایت است. این خانه‌ها دارای ایندکس صفر تا 268،435،455 هستند. به ایندکس هر بایت، آدرس حافظهٔ آن می‌گویند. آدرس‌های حافظه را با اعداد شانزده‌دهی نشان می‌دهند. پس رایانهٔ مذکور دارای محدوده آدرس 0x00000000 تا 0x0fffffff می‌باشد. هر وقت که متغیری را اعلان می‌کنیم، سه ویژگی اساسی به آن متغیر نسبت داده می‌شود: «نوع متغیر» و «نام متغیر» و «آدرس حافظه» آن. مثلا اعلان int n; نوع int و نام n و آدرس چند خانه از حافظه که مقدار n در آن قرار می‌گیرد را به یکدیگر مرتبط می‌سازد. فرض کنید آدرس این متغیر 0x0050cdc0 است. بنابراین می‌توانیم n را

---

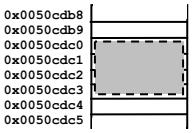
1 – Pointers

2 - References

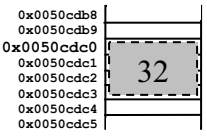
مانند شکل زیر مجسم کنیم:

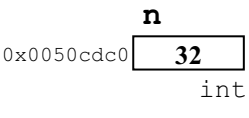


خود متغیر به شکل جعبه نمایش داده شده. نام متغیر، n، در بالای جعبه است و آدرس متغیر در سمت چپ جعبه و نوع متغیر، int، در زیر جعبه نشان داده شده. در بیشتر رایانه‌ها نوع int چهار بایت از حافظه را اشغال می‌نماید. بنابراین همان طور که در شکل مقابل نشان داده شده است، متغیر n یک بلوک چهاربایتی از حافظه را اشغال می‌کند که شامل بایت‌های 0x0050cdc0 تا 0x0050cdc3 است.



توجه کنید که آدرس شی، آدرس اولین بایت از بلوکی است که شی در آن جا ذخیره شده. اگر متغیر فوق به شکل int n=32; مقداردهی اولیه شود، آنگاه بلوک حافظه به شکل زیر خواهد بود. مقدار 32 در چهار بایتی که برای آن متغیر منظور شده ذخیره می‌شود.





در این فصل به بررسی و نحوهٔ استفاده از آدرس‌ها خواهیم پرداخت.

7-1 عملگر ارجاع

در C++ برای بدست آوردن آدرس یک متغیر می‌توان از ***عملگر ارجاع***[1] **&** استفاده نمود. به این عملگر «علمگر آدرس» نیز می‌گویند. عبارت &n آدرس متغیر n را به دست می‌دهد.

× مثال 7-1 چاپ آدرس یک متغیر

```
int main()
{  int n=44;
   cout << " n = " << n << endl;   // prints the value of n
   cout << "&n = " << &n << endl;  // prints the address of n
}
```

```
n = 44
&n = 0x00c9fdc3
```

1 – Reference operator

خروجی نشان می‌دهد که آدرس n در این اجرا برابر با 0x00c9fdc3 است. می‌توان فهمید که این مقدار باید یک آدرس باشد زیرا به شکل شانزده‌دهی نمایش داده شده. اعداد شانزده‌دهی را از روی علامت 0x می‌توان تشخیص داد. معادل دهدهی عدد بالا مقدار 13,237,699 می‌باشد.

نمایش دادن آدرس یک متغیر به این شیوه خیلی مفید نیست. عملگر ارجاع & استفاده‌های مهم‌تری دارد. یک کاربرد آن را در فصل پنجم گفته‌ایم: ساختن پارامترهای ارجاع در اعلان تابع. اکنون کاربرد دیگری معرفی می‌کنیم که خیلی به کاربرد قبلی شبیه است؛ اعلان متغیرهای ارجاع.

7-2 ارجاع‌ها

یک «ارجاع» یک اسم مستعار یا واژهٔ مترادف برای متغیر دیگر است. نحو اعلان یک ارجاع به شکل زیر است:

```
type& ref_name = var_name;
```

type نوع متغیر است، ref_name نام مستعار است و *var_name* نام متغیری است که می‌خواهیم برای آن نام مستعار بسازیم. برای مثال در اعلان :

```
int& rn=n;   // r is a synonym for n
```

rn یک ارجاع یا نام مستعار برای n است. البته n باید قبلا اعلان شده باشد.

× مثال 7-2 استفاده از ارجاع‌ها

در برنامهٔ زیر rn به عنوان یک ارجاع به n اعلان می‌شود:

```
int main()
{  int n=44;
   int& rn=n;      // rn is a synonym for n
   cout << "n = " << n << ", rn = " << rn << endl;
   --n;
   cout << "n = " << n << ", rn = " << rn << endl;
   rn *= 2;
   cout << "n = " << n << ", rn = " << rn << endl;
}
```

```
n = 44, rn = 44
n = 43, rn = 43
n = 86, rn = 86
```

n و rn نام‌های متفاوتی برای یک متغیر است. این دو همیشه مقدار یکسانی دارند. اگر n کاسته شود، rn نیز کاسته شده و اگر rn افزایش یابد، n نیز افزایش یافته است.

همانند ثابت‌ها، ارجاع‌ها باید هنگام اعلان مقداردهی اولیه شوند با این تفاوت که مقدار اولیهٔ یک ارجاع، یک متغیر است نه یک لیترال. بنابراین کد زیر اشتباه است:

```
int& rn=44;     // ERROR: 44 is not a variable
```

گرچه برخی از کامپایلرها ممکن است دستور بالا را مجاز بدانند ولی با نشان دادن یک هشدار اعلام می‌کنند که یک متغیر موقتی ایجاد شده تا rn به حافظهٔ آن متغیر، ارجاع داشته باشد.

درست است که ارجاع با یک متغیر مقداردهی می‌شود، اما ارجاع به خودی خود یک متغیر نیست. یک متغیر، فضای ذخیره‌سازی و نشانی مستقل دارد، حال آن که ارجاع از فضای ذخیره‌سازی و نشانی متغیر دیگری بهره می‌برد.

× مثال 3-7 ارجاع‌ها متغیرهای مستقل نیستند

```
int main()
{  int n=44;
   int& rn=n;          // rn is a synonym for n
   cout << "  &n = " << &n << ",  &rn = " << &rn << endl;
   int& rn2=n;        // rn2 is another synonym for n
   int& rn3=rn;       // rn3 is another synonym for n
   cout << "&rn2 = " << &rn2 << ", &rn3 = " << &rn3 << endl;
}
```

```
&n = 0x0064fde4,  &rn = 0x0064fde4
&rn2 = 0x0064fde4, &rn3 = 0x0064fde4
```

در برنامهٔ فوق فقط یک شی وجود دارد و آن هم n است. rn و rn2 و rn3 ارجاع‌هایی به n هستند. خروجی نیز تایید می‌کند که آدرس rn و rn2 و rn3 با آدرس n یکی است. یک شی می‌تواند چند ارجاع داشته باشد.

ارجاع‌ها بیشتر برای ساختن پارامترهای ارجاع در توابع به کار می‌روند. تابع می‌تواند مقدار یک آرگومان را که به طریق ارجاع ارسال شده تغییر دهد زیرا آرگومان اصلی و پارامتر ارجاع هر دو یک شی هستند. تنها فرق این است که دامنهٔ پارامتر ارجاع به همان تابع محدود شده است.

7-3 اشاره‌گرها

می‌دانیم که اعداد صحیح را باید در متغیری از نوع int نگهداری کنیم و اعداد اعشاری را در متغیرهایی از نوع float. به همین ترتیب کاراکترها را باید در متغیرهایی از نوع char نگهداریم و مقدارهای منطقی را در متغیرهایی از نوع bool. اما آدرس حافظه را در چه نوع متغیری باید قرار دهیم؟

عملگر ارجاع & آدرس حافظهٔ یک متغیر موجود را به دست می‌دهد. می‌توان این آدرس را در متغیر دیگری ذخیره نمود. متغیری که یک آدرس در آن ذخیره می‌شود ***اشاره‌گر*** نامیده می‌شود. برای این که یک اشاره‌گر اعلان کنیم، ابتدا باید مشخص کنیم که آدرس چه نوع داده‌ای قرار است در آن ذخیره شود. سپس از عملگر اشاره * استفاده می‌کنیم تا اشاره‌گر را اعلان کنیم. برای مثال دستور :

```
float* px;
```

اشاره‌گری به نام px اعلان می‌کند که این اشاره‌گر، آدرس متغیرهایی از نوع float را نگهداری می‌نماید. به طور کلی برای اعلان یک اشاره‌گر از نحو زیر استفاده می‌کنیم:

```
type* pointername;
```

که *type* نوع متغیرهایی است که این اشاره‌گر آدرس آن‌ها را نگهداری می‌کند و pointername نام اشاره‌گر است.

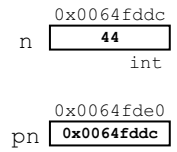
آدرس یک شی از نوع int را فقط می‌توان در اشاره‌گری از نوع int* ذخیره کرد و آدرس یک شی از نوع float را فقط می‌توان در اشاره‌گری از نوع float* ذخیره نمود. دقت کنید که یک اشاره‌گر، یک متغیر مستقل است.

× مثال 4-7 به کارگیری اشاره‌گرها

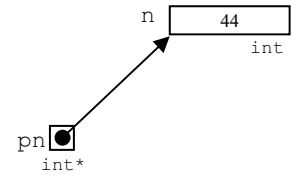
برنامهٔ زیر یک متغیر از نوع int به نام n و یک اشاره‌گر از نوع int* به نام pn را اعلان می‌کند:

```
int main()
{  int n=44;
   cout << "n = " << n << ", &n = " << &n << endl;
   int* pn=&n;   // pn holds the address of n
   cout << "       pn = " << pn << endl;
   cout << "&pn = " << &pn << endl;
}
```

```
n = 44, &n = 0x0064fddc
  pn = 0x0064fddc
&pn = 0x0064fde0
```



متغیر n با مقدار 44 مقداردهی شده و آدرس آن 0x0064fddc می‌باشد. اشاره‌گر pn با مقدار &n یعنی آدرس n مقداردهی شده. پس مقدار درون pn برابر با 0x0064fddc است (خط دوم خروجی این موضوع را تایید می‌کند) . اما pn یک متغیر مستقل است و آدرس مستقلی دارد. &pn آدرس pn را به دست می‌دهد. خط سوم خروجی ثابت می‌کند که متغیر pn مستقل از متغیر n است. تصویر زیر به درک بهتر این موضوع کمک می‌کند. در این تصویر ویژگی‌های مهم n و pn نشان داده شده. pn یک اشاره‌گر به n است و n مقدار 44 دارد.



وقتی می‌گوییم «pn به n اشاره می‌کند» یعنی درون pn آدرس n قرار دارد.

4-7 مقداریابی

فرض کنید n دارای مقدار 22 باشد و pn اشاره‌گری به n باشد. با این حساب باید بتوان از طریق pn به مقدار 22 رسید. با استفاده از * می‌توان مقداری که اشاره‌گر به آن اشاره دارد را به دست آورد. به این کار ***مقداریابی اشاره‌گر*** می‌گوییم.

× مثال 7-5 مقداریابی یک اشاره‌گر

این برنامه همان برنامهٔ مثال 7-4 است. فقط یک خط کد بیشتر دارد:

```
int main()
{  int n=44;
   cout << "n = " << n << ", &n = " << &n << endl;
   int* pn=&n;    // pn holds the address of n
   cout << "        pn = " << pn << endl;
   cout << "&pn = " << &pn << endl;
   cout << "*pn = " << *pn << endl;
}
```

```
n = 44, &n = 0x0064fdcc
  pn = 0x0064fdcc
&pn = 0x0064fdd0
xpn = 44
```

ظاهرا pn* یک اسم مستعار برای n است زیرا هر دو یک مقدار دارند.

یک اشاره‌گر به هر چیزی می‌تواند اشاره کند، حتی به یک اشاره‌گر دیگر. به مثال زیر دقت کنید.

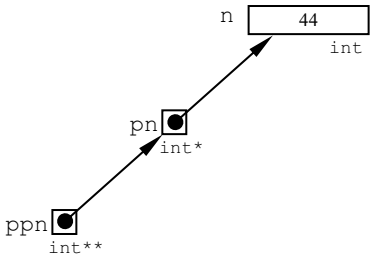
× مثال 7-6 اشاره‌گری به اشاره‌گرها

این کد ادامهٔ ساختار برنامهٔ مثال 7-4 است:

```
int main()
{  int n=44;
   cout << "    n = " << n << endl;
   cout << "   &n = " << &n << endl;
   int* pn=&n;      // pn holds the address of n
   cout << "   pn = " << pn << endl;
   cout << "  &pn = " << &pn << endl;
   cout << "  *pn = " << *pn << endl;
   int** ppn=&pn;   // ppn holds the address of pn
   cout << "  ppn = " << ppn << endl;
   cout << " &ppn = " << &ppn << endl;
   cout << " *ppn = " << *ppn << endl;
   cout << "**ppn = " << **ppn << endl;
```

```
}
```

```
    n = 44
   &n = 0x0064fd78
   pn = 0x0064fd78
  &pn = 0x0064fd7c
  *pn = 44
  ppn = 0x0064fd7c
 &ppn = 0x0064fd80
 *ppn = 0x0064fd78
**ppn = 44
```



در برنامهٔ بالا متغیر n از نوع int تعریف شده. pn اشاره‌گری است که به n اشاره دارد. پس نوع pn باید *int باشد. ppn اشاره‌گری است که به pn اشاره می‌کند. پس نوع ppn باید **int باشد. همچنین چون ppn به pn اشاره دارد، پس *ppn مقدار pn را نشان می‌دهد و چون pn به n اشاره دارد، پس *pn مقدار n را می‌دهد. اگر این دو دستور را کنار هم بچینیم نتیجه این می‌شود که **ppn مقدار n را بر می‌گرداند. خروجی حاصل از اجرای برنامهٔ بالا این گفته‌ها را تصدیق می‌نماید.

به طور کلی اگر متغیری از نوع T باشد، آنگاه اشاره‌گر به آن از نوع *T خواهد بود. به *T یک نوع «مشتق‌شده» می‌گویند زیرا از روی نوع دیگری ساخته شده است. *T خود یک نوع جدید است. حال اگر بخواهیم اشاره‌گری به نوع *T داشته باشیم (یعنی اشاره‌گری به اشاره‌گر دیگر) طبق قاعدهٔ فوق این اشاره‌گر جدید را باید از نوع **T تعریف نمود. نمودار فوق نحوهٔ کار اشاره‌گرها در مثال بالا را نشان می‌دهد.

گرچه pn و ppn هر دو اشاره‌گر هستند اما از یک نوع نیستند. pn از نوع *int است و ppn از نوع **int است.

عملگر مقداریابی * و عملگر ارجاع & معکوس یکدیگر رفتار می‌کنند. اگر این دو را با هم ترکیب کنیم، یکدیگر را خنثی می‌نمایند. اگر n یک متغیر باشد، &n آدرس آن متغیر است. از طرفی با استفاده از عملگر * می‌توان مقداری که در آدرس &n قرار گرفته را به دست آورد. بنابراین *&n برابر با خود n خواهد بود. همچنین اگر p یک اشاره‌گر باشد، *p مقداری که p به آن اشاره دارد را می‌دهد. از طرفی با استفاده از عملگر & می‌توانیم آدرس چیزی که در *p قرار گرفته را بدست آوریم. پس &*p برابر

با خود p خواهد بود. ترتیب قرارگرفتن این عملگرها مهم است. یعنی &n* با &*n برابر نیست. علت این امر را توضیح دهید.

عملگر * دو کاربرد دارد. اگر پسوند یک نوع باشد (مثل int*) یک اشاره‌گر به آن نوع را تعریف می‌کند و اگر پیشوند یک اشاره‌گر باشد (مثل p*) آنگاه مقداری که p به آن اشاره می‌کند را برمی‌گرداند. عملگر & نیز دو کاربرد دارد. اگر پسوند یک نوع باشد (مثل int&) یک نام مستعار تعریف می‌کند و اگر پیشوند یک متغیر باشد (مثل &n) آدرس آن متغیر را می‌دهد.

7-6 چپ مقدارها، راست مقدارها

یک دستور جایگزینی دو بخش دارد: بخشی که در سمت چپ علامت جایگزینی قرار می‌گیرد و بخشی که در سمت راست علامت جایگزینی قرار می‌گیرد. مثلا دستور n = 55; متغیر n در سمت چپ قرار گرفته و مقدار 55 در سمت راست. این دستور را نمی‌توان به شکل 55 = n; نوشت زیرا مقدار 55 یک ثابت است و نمی‌تواند مقدار بگیرد. پس هنگام استفاده از عملگر جایگزینی باید دقت کنیم که چه چیزی را در سمت چپ قرار بدهیم و چه چیزی را در سمت راست.

چیزهایی که می‌توانند در سمت چپ جایگزینی قرار بگیرند «چپ‌مقدار[1]» خوانده می‌شوند و چیزهایی که می‌توانند در سمت راست جایگزینی قرار بگیرند «راست‌مقدار[2]» نامیده می‌شوند. متغیرها (و به طور کلی اشیا) چپ‌مقدار هستند و لیترال‌ها (مثل 15 و "ABC") راست مقدار هستند.

یک ثابت در ابتدا به شکل یک چپ‌مقدار نمایان می‌شود:

```
const int MAX = 65535;      // MAX is an lvalue
```

اما از آن پس دیگر نمی‌توان به عنوان چپ مقدار از آن‌ها استفاده کرد:

```
MAX = 21024;     // ERROR: MAX is constant
```

به این گونه چپ‌مقدارها، چپ‌مقدارهای «تغییر ناپذیر» گفته می‌شود. مثل آرایه‌ها:

```
int a[] = {1,2,3};    // O.K
a[] = {1,2,3};        // ERROR
```

---

1 – L_values 2- R_values

مابقی چپ‌مقدارها که می‌توان آن‌ها را تغییر داد، چپ‌مقدارهای «تغییر پذیر» نامیده می‌شوند. هنگام اعلان یک ارجاع به یک چپ‌مقدار نیاز داریم:

```
int& r = n;             // O.K. n is an lvalue
```

اما اعلان‌های زیر غیرمعتبرند زیرا هیچ کدام چپ‌مقدار نیستند:

```
int& r = 44;            // ERROR: 44 is not an lvalue
int& r = n++;           // ERROR: n++ is not an lvalue
int& r = cube(n);       // ERROR: cube(n) is not an lvalue
```

یک تابع، چپ‌مقدار نیست اما اگر نوع بازگشتی آن یک ارجاع باشد، می‌توان تابع را به یک چپ‌مقدار تبدیل کرد.

7-7 بازگشت از نوع ارجاع

در بحث توابع، ارسال از طریق مقدار و ارسال از طریق ارجاع را دیدیم. این دو شیوهٔ تبادل در مورد بازگشت از تابع نیز صدق می‌کند: بازگشت از طریق مقدار و بازگشت از طریق ارجاع. توابعی که تاکنون دیدیم بازگشت به طریق مقدار داشتند. یعنی همیشه یک مقدار به فراخواننده برمی‌گشت. می‌توانیم تابع را طوری تعریف کنیم که به جای مقدار، یک ارجاع را بازگشت دهد. مثلا به جای این که مقدار m را بازگشت دهد، یک ارجاع به m را بازگشت دهد.

وقتی بازگشت به طریق مقدار باشد، تابع یک راست‌مقدار خواهد بود زیرا مقدارها لیترال هستند و لیترال‌ها راست‌مقدارند. به این ترتیب تابع را فقط در سمت راست یک جایگزینی می‌توان به کار برد مثل: `m = f();`

وقتی بازگشت به طریق ارجاع باشد، تابع یک چپ‌مقدار خواهد بود زیرا ارجاع‌ها چپ‌مقدار هستند. در این حالت تابع را می‌توان در سمت چپ یک جایگزینی قرار داد مثل : `f() = m;`

برای این که نوع بازگشتی تابع را به ارجاع تبدیل کنیم کافی است عملگر ارجاع را به عنوان پسوند نوع بازگشتی درج کنیم.

× مثال 8-7 بازگشت از نوع ارجاع

```
int& max(int& m, int& n)       // return type is reference to int
{  return (m > n ? m : n);     // m and n are non-local references
}
int main()
{  int m = 44, n = 22;
   cout << m << ", " << n << ", " << max(m,n) << endl;
   max(m,n) = 55;           // changes the vale of m from 44 to 55
   cout << m << ", " << n << ", " << max(m,n) << endl;
}
```

```
44, 22, 44
55, 22, 55
```

تابع max() از بین m و n مقدار بزرگ‌تر را پیدا کرده و سپس ارجاعی به آن را باز می‌گرداند. بنابراین اگر m از n بزرگ‌تر باشد، تابع max(m,n) آدرس m را برمی‌گرداند. پس وقتی می‌نویسیم max(m,n) = 55; مقدار 55 در حقیقت درون متغیر m قرار می‌گیرد (اگر m>n باشد). به بیانی ساده، فراخوانی max(m,n) خود m را بر می‌گرداند نه مقدار آن را.

اخطار: وقتی یک تابع پایان می‌یابد، متغیرهای محلی آن نابود می‌شوند. پس هیچ وقت ارجاعی به یک متغیر محلی بازگشت ندهید زیرا وقتی کار تابع تمام شد، آدرس متغیرهای محلی‌اش غیر معتبر می‌شود و ارجاع بازگشت داده شده ممکن است به یک مقدار غیر معتبر اشاره داشته باشد. تابع max() در مثال بالا یک ارجاع به m یا n را بر می‌گرداند. چون m و n خودشان به طریق ارجاع ارسال شده‌اند، پس محلی نیستند و بازگرداندن ارجاعی به آن‌ها خللی در برنامه وارد نمی‌کند.

به اعلان تابع max() دقت کنید:

```
int& max(int& m, int& n)
```

نوع بازگشتی آن با استفاده از عملگر ارجاع & به شکل یک ارجاع درآمده است.

× مثال 9-7 به کارگیری یک تابع به عنوان عملگر زیرنویس آرایه

```
float& component(float× v, int k)
```

```
{  return v[k-1];
}

int main()
{  float v[4];
   for (int k = 1; k <= 4; k++)
      component(v,k) = 1.0/k;
   for (int i = 0; i < 4; i++)
      cout << "v[" << i << "] = " << v[i] << endl;
}
```

```
v[0] = 1
v[1] = 0.5
v[2] = 0.333333
v[3] = 0.25
```

تابع component() باعث می‌شود که ایندکس آرایه v از «شماره‌گذاری از صفر» به «شماره‌گذاری از یک» تغییر کند. بنابراین component(v,3) معادل v[2] است. این کار از طریق بازگشت از طریق ارجاع ممکن شده است.

7-8 آرایه‌ها و اشاره‌گرها

گرچه اشاره‌گرها از انواع عددی صحیح نیستند اما بعضی از اعمال حسابی را می‌توان روی اشاره‌گرها انجام داد. حاصل این می‌شود که اشاره‌گر به خانهٔ دیگری از حافظه اشاره می‌کند. اشاره‌گرها را می‌توان مثل اعداد صحیح افزایش و یا کاهش داد و می‌توان یک عدد صحیح را به آن‌ها اضافه نمود یا از آن کم کرد. البته میزان افزایش یا کاهش اشاره‌گر بستگی به نوع داده‌ای دارد که اشاره‌گر به آن اشاره دارد. این موضوع را با استفاده از مثال زیر بررسی می‌کنیم.

× مثال 7-10 پیمایش آرایه با استفاده از اشاره‌گر

این مثال نشان می‌دهد که چگونه می‌توان از اشاره‌گر برای پیمایش یک آرایه استفاده نمود:

```
int main()
{  const int SIZE = 3;
```

```
   short a[SIZE] = {22, 33, 44};
   cout << "a = " << a << endl;
   cout << "sizeof(short) = " << sizeof(short) << endl;
   short* end = a + SIZE;       // converts SIZE to offset 6
   short sum = 0;
   for (short* p = a; p < end; p++)
   {  sum += *p;
      cout << "\t p = " << p;
      cout << "\t *p = " << *p;
      cout << "\t sum = " << sum << endl;
   }
   cout << "end = " << end << endl;
}
```

```
a = 0x3fffd1a
sizeof(short) = 2
  p = 0x3fffd1a      *p = 22      sum = 22
  p = 0x3fffd1c      *p = 33      sum = 55
  p = 0x3fffd1e      *p = 44      sum = 99
end = 0x3fffd20
```

تابع sizeof() تعداد بایت‌هایی که یک نوع بنیادی اشغال می‌نماید را نشان می‌دهد. خط دوم خروجی نشان می‌دهد که نوع short در این رایانه 2 بایت از حافظه را اشغال می‌کند. در برنامهٔ بالا آرایه‌ای از نوع short با سه عنصر اعلان شده. همچنین اشاره‌گرهایی به نام p و end از نوع اشاره‌گر به short اعلان شده‌اند. چون p اشاره‌گر به short است، هر زمان که p یک واحد افزایش یابد، دو بایت به آدرس درون p اضافه می‌کند و p در حقیقت به عدد short بعدی پیشروی می‌نماید (نه به خانهٔ بعدی حافظه). در حلقهٔ for ابتدا آدرس آرایهٔ a درون اشاره‌گر p قرار می‌گیرد. پس p به اولین عنصر آرایه اشاره می‌کند. هر بار که حلقه تکرار شود، یک واحد به p اضافه می‌شود و p به عنصر بعدی آرایه اشاره می‌نماید. عبارت sum += *p; نیز مقدار آن عنصر را دریافت کرده و به مقدار sum اضافه می‌کند.

این مثال نشان می‌دهد که هر گاه یک اشاره‌گر افزایش یابد، مقدار آن به اندازهٔ تعداد بایت‌های شیئی که به آن اشاره می‌کند، افزایش می‌یابد. مثلا اگر p اشاره‌گری به

double باشد و sizeof(double) برابر با هشت بایت باشد، هر گاه که p یک واحد افزایش یابد، اشاره‌گر p هشت بایت به پیش می‌رود. مثلا کد زیر :

```
float a[8];
float* p = a;     // p points to a[0]
++p;              // increases the value of p by sizeof(float)
```

اگر floatها 4 بایت را اشغال کنند آنگاه p++ مقدار درون p را 4 بایت افزایش می‌دهد و p += 5; مقدار درون p را 20 بایت افزایش می‌دهد. با استفاده از خاصیت مذکور می‌توان آرایه را پیمایش نمود: یک اشاره‌گر را با آدرس اولین عنصر آرایه مقداردهی کنید، سپس اشاره‌گر را پی در پی افزایش دهید. هر افزایش سبب می‌شود که اشاره‌گر به عنصر بعدی آرایه اشاره کند. یعنی اشاره‌گری که به این نحو به کار گرفته شود مثل ایندکس آرایه عمل می‌کند. همچنین با استفاده از اشاره‌گر می‌توانیم مستقیما به عنصر مورد نظر در آرایه دستیابی کنیم:

```
float* p = a;        // p points to a[0]
p += 5;              // now p points to a[5]
```

یک نکتهٔ ظریف در ارتباط با آرایه‌ها و اشاره‌گرها وجود دارد: اگر اشاره‌گر را بیش از ایندکس آرایه افزایش دهیم، ممکن است به بخش‌هایی از حافظه برویم که هنوز تخصیص داده نشده‌اند یا برای کارهای دیگر تخصیص یافته‌اند. تغییر دادن مقدار این بخش‌ها باعث بروز خطا در برنامه و کل سیستم می‌شود. همیشه باید مراقب این خطر باشید. کد زیر نشان می‌دهد که چطور این اتفاق رخ می‌دهد.

```
float a[8];
float* p = a[7];      // points to last element in the array
++p;                  // now p points to memory past last element!
*p = 22.2;            // TROUBLE!
```

مثال بعدی نشان می‌دهد که ارتباط تنگاتنگی بین آرایه‌ها و اشاره‌گرها وجود دارد. نام آرایه در حقیقت یک اشاره‌گر ثابت (const) به اولین عنصر آرایه است. همچنین خواهیم دید که اشاره‌گرها را مانند هر متغیر دیگری می‌توان با هم مقایسه نمود.

× مثال 11-7 پیمایش عناصر آرایه از طریق آدرس

```
int main()
{  short a[] = {22, 33, 44, 55, 66};
   cout << "a = " << a << ", *a = " << *a << endl;
   for (short* p = a; p < a +5; p++)
      cout << "p = " << p << ", *p = " << *p << endl;
}
```

```
 a = 0x3fffd08, *a = 22
 p = 0x3fffd08, *p = 22
 p = 0x3fffd0a, *p = 33
 p = 0x3fffd0c, *p = 44
 p = 0x3fffd0e, *p = 55
 p = 0x3fffd10, *p = 66
 p = 0x3fffd12, *p = 77
```

در نگاه اول، a و p مانند هم هستند: هر دو به نوع short اشاره می‌کنند و هر دو دارای مقدار 0x3fffd08 هستند. اما a یک اشاره‌گر ثابت است و نمی‌تواند افزایش یابد تا آرایه پیمایش شود. پس به جای آن p را افزایش می‌دهیم تا آرایه را پیمایش کنیم. شرط (p < a+5) حلقه را خاتمه می‌دهد. a+5 به شکل زیر ارزیابی می‌شود:

```
0x3fffd08 + 5*sizeof(short) = 0x3fffd08 + 5*2 = 0x3fffd08 + 0xa = 0x3fffd12
```

پس حلقه تا زمانی که p < 0x3fffd12 باشد ادامه می‌یابد.

عملگر زیرنویس [] مثل عملگر مقداریابی * رفتار می‌کند. هر دوی این‌ها می‌توانند به عناصر آرایه دسترسی مستقیم داشته باشند.

```
a[0] == *a
a[1] == *(a + 1)
a[2] == *(a + 2)
    ...
    ...
```

پس با استفاده از کد زیر نیز می‌توان آرایه را پیمایش نمود:

```
for (int i = 0; i < 8; i++)
   cout << *(a + i) << endl;
```

× مثال 12-7 مقایسهٔ الگو

در این مثال، تابع loc() در میان n1 عنصر اول آرایهٔ a1 به دنبال n2 عنصر اول آرایهٔ a2 می‌گردد. اگر پیدا شد، یک اشاره‌گر به درون a1 برمی‌گرداند که a2 از آن‌جا شروع می‌شود وگرنه اشاره‌گر NULL را برمی‌گرداند.

```
short* loc(short* a1, short* a2, int n1, int n2)
{  short* end1 = a1 + n1;
   for (short* p1 = a1; p1 <end1; p1++)
      if (*p1 == *a2)
      {  for (int j = 0; j < n2; j++)
         if (p1[j] != a2[j]) break;
         if (j == n2) return p1;
      }
   return 0;
}

int main()
{  short a1[9] = {11, 11, 11, 11, 11, 22, 33, 44, 55};
   short a2[5] = {11, 11, 11, 22, 33};
   cout << "Array a1 begins at location\t" << a1 << endl;
   cout << "Array a2 begins at location\t" << a2 << endl;
   short* p = loc(a1, a2, 9, 5);
   if (p)
   {  cout << "Array a2 found at location\t" << p << endl;
      for (int i = 0; i < 5; i++)
         cout << "\t" << &p[i] << ": " << p[i] << "\t"
              << &a2[i] << ": " << a2[i] << endl;
   }
   else cout << "Not found.\n";
}
```

```
Array a1 begins at location      0x3fffd12
Array a2 begins at location      0x3fffd08
Array a2 found at location        0x3fffd16
      0x3fffd16: 11      0x3fffd08: 11
      0x3fffd18: 11      0x3fffd0a: 11
      0x3fffd1a: 11      0x3fffd0c: 11
      0x3fffd1c: 22      0x3fffd0e: 22
      0x3fffd1e: 33      0x3fffd10: 33
```

الگوریتم مقایسهٔ الگو از دو حلقه استفاده می‌کند. حلقهٔ بیرونی اشاره‌گر p1 را در آرایهٔ a1 جلو می‌برد تا جایی که عنصری که p1 به آن اشاره می‌کند با اولین عنصر آرایهٔ a2 برابر باشد. آنگاه حلقهٔ درونی شروع می‌شود. در این حلقه عناصر بعد از p1 یکی یکی با عناصر متناظرشان در a2 مقایسه می‌شوند. اگر نابرابری پیدا شود، حلقهٔ درونی فورا خاتمه یافته و حلقهٔ بیرونی دور جدیدش را آغاز می‌کند. یعنی p1 دوباره در آرایهٔ a1 به پیش می‌رود تا به عنصر بعدی برسد که این عنصر با اولین عنصر آرایهٔ a2 برابر باشد. ولی اگر حلقهٔ داخلی بدون توقف به پایان رسید، به این معناست که عناصر بعد از p1 با عناصر متناظرشان در a2 برابرند. پس a2 در محل p1 یافت شده است و p1 به عنوان مکان مورد نظر بازگردانده می‌شود. دقت کنید که اگر چه p1 آرایه نیست ولی به شکل p1[j] استفاده شده. همان طور که قبلا گفتیم این عبارت با عبارت p1+j یکی است. برنامهٔ آزمون وارسی می‌کند که واقعا آدرس‌ها بررسی شوند و مقادیر درون آدرس‌ها یکسان باشد.

7-13 عملگر new

وقتی یک اشاره‌گر شبیه این اعلان شود:

```
float* p;    // p is a pointer to a float
```

یک فضای چهاربایتی به p تخصیص داده می‌شود (معمولا sizeof(float) چهار بایت است). حالا p ایجاد شده است اما به هیچ جایی اشاره نمی‌کند زیرا هنوز آدرسی درون آن قرار نگرفته. به چنین اشاره‌گری ***اشاره‌گر سرگردان*** می‌گویند. اگر سعی کنیم یک اشاره‌گر سرگردان را مقداریابی یا ارجاع کنیم با خطا مواجه می‌شویم. مثلا دستور:

```
xp = 3.14159;      // ERROR: no storage has been allocated for *P
```

خطاست. زیرا p به هیچ آدرسی اشاره نمی‌کند و سیستم عامل نمی‌داند که مقدار 3.14159 را کجا ذخیره کند. برای رفع این مشکل می‌توان اشاره‌گرها را هنگام اعلان، مقداردهی کرد:

```
float x = 0;     // x cintains the value 0
float* p = &x    // now p points to x
*p = 3.14159;    // O.K. assigns this value to address that p points to
```

در این حالت می‌توان به p* دستیابی داشت زیرا حالا p به x اشاره می‌کند و آدرس آن را دارد. راه حل دیگر این است که یک آدرس اختصاصی ایجاد شود و درون p قرار بگیرد. بدین ترتیب p از سرگردانی خارج می‌شود. این کار با استفاده از عملگر new صورت می‌پذیرد:

```
float* p;
p = new float;          // allocates storage for 1 float
xp = 3.14159;           // O.K. assigns this value to that
storage
```

دقت کنید که عملگر new فقط خود p را مقداردهی می‌کند نه آدرسی که p به آن اشاره می‌کند. می‌توانیم سه خط فوق را با هم ترکیب کرده و به شکل یک دستور بنویسیم:

```
float* p = new float(3.141459);
```

با این دستور، اشاره‌گر p از نوع float* تعریف می‌شود و سپس یک بلوک خالی از نوع float منظور شده و آدرس آن به p تخصیص می‌یابد و همچنین مقدار 3.14159 در آن آدرس قرار می‌گیرد. اگر عملگر new نتواند خانهٔ خالی در حافظه پیدا کند، مقدار صفر را برمی‌گرداند. اشاره‌گری که این چنین باشد، «اشاره‌گر تهی» یا NULL می‌نامند. با استفاده از کد هوشمند زیر می‌توانیم مراقب باشیم که اشاره‌گر تهی ایجاد نشود:

```
double* p = new double;
if (p == 0) abort();     // allocator failed: insufficent memory
else *p = 3.141592658979324;
```

در این قطعه کد، هرگاه اشاره‌گری تهی ایجاد شد، تابع abort() فراخوانی شده و این دستور لغو می‌شود.

تاکنون دانستیم که به دو طریق می‌توان یک متغیر را ایجاد و مقداردهی کرد. روش اول:

```
float x = 3.14159;              // allocates named memory
```

و روش دوم:

```
float* p = new float(3.14159);   // allocates unnamed memory
```

در حالت اول، حافظهٔ مورد نیاز برای x هنگام کامپایل تخصیص می‌یابد. در حالت دوم حافظهٔ مورد نیاز در زمان اجرا و به یک شیء بی‌نام تخصیص می‌یابد که با استفاده از p* قابل دستیابی است.

7-14 عملگر delete

عملگر delete عملی برخلاف عملگر new دارد. کارش این است که حافظهٔ اشغال شده را آزاد کند. وقتی حافظه‌ای آزاد شود، سیستم عامل می‌تواند از آن برای کارهای دیگر یا حتی تخصیص‌های جدید استفاده کند. عملگر delete را تنها روی اشاره‌گرهایی می‌توان به کار برد که با دستور new ایجاد شده‌اند. وقتی حافظهٔ یک اشاره‌گر آزاد شد، دیگر نمی‌توان به آن دستیابی نمود مگر این که دوباره این حافظه تخصیص یابد:

```
float* p = new float(3.14159);
delete p;            // deallocates q
xp = 2.71828;        // ERROR: q has been deallocated
```

وقتی اشاره گر p در کد بالا آزاد شود، حافظه‌ای که توسط new به آن تخصیص یافته بود، آزاد شده و به میزان sizeof(float) به حافظهٔ آزاد اضافه می‌شود. وقتی اشاره‌گری آزاد شد، به هیچ چیزی اشاره نمی‌کند؛ مثل متغیری که مقداردهی نشده. به این اشاره‌گر، اشاره‌گر سرگردان می‌گویند.

اشاره‌گر به یک شیء ثابت را نمی‌توان آزاد کرد:

```
const int* p = new int;
delete p;          // ERROR: cannot delete pointer to const objects
```

علت این است که «ثابت‌ها نمی‌توانند تغییر کنند».

اگر متغیری را صریحا اعلان کرده‌اید و سپس اشاره‌گری به آن نسبت داده‌اید، از عملگر delete استفاده نکنید. این کار باعث اشتباه غیر عمدی زیر می‌شود:

```
float x = 3.14159;      // x contains the value 3.14159
float* p = &x;          // p contains the address of x
delete p;               // WARNING: this will make x free
```

کد بالا باعث می‌شود که حافظهٔ تخصیص‌یافته برای x آزاد شود. این اشتباه را به سختی می‌توان تشخیص داد و اشکال‌زدایی کرد.

9-7 آرایه‌های پویا

نام آرایه در حقیقت یک اشاره‌گر ثابت است که در زمان کامپایل، ایجاد و تخصیص داده می‌شود:

```
float a[20];   // a is a const pointer to a block of 20 floats
float* const p = new float[20];    // so is p
```

هم a و هم p اشاره‌گرهای ثابتی هستند که به بلوکی حاوی 20 متغیر float اشاره دارند. به اعلان a ***بسته‌بندی ایستا***[2] می‌گویند زیرا این کد باعث می‌شود که حافظهٔ مورد نیاز برای a در زمان کامپایل تخصیص داده شود. وقتی برنامه اجرا شود، به هر حال حافظهٔ مربوطه تخصیص خواهد یافت حتی اگر از آن هیچ استفاده‌ای نشود. می‌توانیم با استفاده از اشاره‌گر، آرایهٔ فوق را طوری تعریف کنیم که حافظه مورد نیاز آن فقط در زمان اجرا تخصیص یابد:

```
float* p = new float[20];
```

دستور بالا، 20 خانهٔ خالی حافظه از نوع float را در اختیار گذاشته و اشاره‌گر p را به خانهٔ اول آن نسبت می‌دهد. به این آرایه، «آرایهٔ پویا[1]» می‌گویند. به این طرز ایجاد اشیا ***بسته‌بندی پویا***[3] یا «بسته‌بندی زمان جرا» می‌گویند.

آرایهٔ ایستای a و آرایهٔ پویای p را با یکدیگر مقایسه کنید. آرایهٔ ایستای a در زمان کامپایل ایجاد می‌شود و تا پایان اجرای برنامه، حافظهٔ تخصیصی به آن مشغول می‌ماند. ولی آرایهٔ پویای p در زمان اجرا و هر جا که لازم شد ایجاد می‌شود و پس از اتمام کار نیز می‌توان با عملگر delete حافظهٔ تخصیصی به آن را آزاد کرد:

```
delete [] p;
```

---

1 – Dynamic arrays 2 – Static binding 3 – Dynamic binding

برای آزاد کردن آرایهٔ پویای p براکت‌ها [] قبل از نام p باید حتما قید شوند زیرا p به یک آرایه اشاره دارد.

× مثال 15-7 استفاده از آرایه‌های پویا

تابع get() در برنامهٔ زیر یک آرایهٔ پویا ایجاد می‌کند:

```
void get(double*& a, int& n)
{  cout << "Enter number of items: ";  cin >> n;
   a = new double[n];
   cout << "Enter " << n << " items, one per line:\n";
   for (int i = 0; i < n; i++)
   {  cout << "\t" << i+1 << ": ";
      cin >> a[i];
   }
}
void print(double* a, int n)
{  for (int i = 0; i < n; i++)
   cout << a[i] << " " ;
   cout << endl;
}
int main()
{  double* a;     // a is simply an unallocated pointer
   int n;
   get(a,n);      // now a is an array of n doubles
   print(a,n);
   delete [] a;   // now a is simply an unallocated pointer again
   get(a,n);      // now a is an array of n doubles
   print(a,n);
}
```

```
Enter number of items: 4
Enter 4 items, one per line:
      1: 44.4
      2: 77.7
      3: 22.2
      4: 88.8
44.4 77.7 22.2 88.8
Enter number of items: 2
```

```
Enter 2 items, one per line:
      1: 3.33
      2: 9.99
3.33 9.99
```

وقتی برنامه اجرا می‌شود، ابتدا اشاره‌گر تهی a از نوع double* ایجاد می‌شود. سپس این اشاره‌گر به تابع get() فرستاده می‌شود. تابع get() یک آرایهٔ پویا ایجاد کرده و آدرس اولین خانهٔ آن را درون اشاره‌گر a می‌گذارد. نکته جالب این جاست که تعداد عناصر آرایه هنگام اجرا مشخص می‌شود. یعنی وقتی تابع get() فراخوانی شد، از کاربر پرسیده می‌شود که اندازهٔ آرایه چقدر باشد. سپس با استفاده از عملگر new آرایه‌ای به همان اندازه ساخته می‌شود. پس از آن با استفاده از حلقهٔ for مقادیر آرایه یکی یکی از ورودی دریافت شده و با استفاده از عبارت cin >> a[i]; درون عناصر آرایه قرار می‌گیرد (عبارت a[i] با عبارت a+i معادل است. مثال 12-7 را ببینید). در نهایت آرایهٔ a و تعداد عناصرش به برنامهٔ اصلی باز می‌گردد. تابع print() وظیفه دارد که با پیش‌بردن اشاره‌گر روی این آرایه، مقادیر موجود در آن را چاپ کند. پس از این کار، با استفاده از عملگر delete آرایهٔ مورد نظر آزاد می‌شود. دوباره با فراخوانی تابع get() می‌توان آرایهٔ جدیدی ساخت. این آرایهٔ جدید می‌تواند اندازهٔ متفاوتی داشته باشد. توجه کنید که عملگر زیرنویس [] حتما باید در دستور delete به کار رود تا کل آرایه آزاد شود. همچنین ببینید که پارامتر a در تابع get() به چه شکلی اعلان شده:

```
void get(double*& a, int& n)
```

تابع get() قرار است که به اشاره‌گر تهی a مقداری را نسبت دهد. به همین دلیل a باید به شکل ارجاع ارسال شود تا تابع بتواند مقدار آن را دست‌کاری کند. چون a از نوع double* است، پس شکل ارجاعی‌اش به صورت double*& خواهد بود. برهان بالا به این معناست که تابع می‌تواند اشاره‌گرهایی محلی داشته باشد و اگر اشاره‌گری به طریق ارجاع ارسال نشود، تابع یک نسخهٔ محلی از آن می‌سازد. حالا بگویید چرا در تعریف تابع print() اشاره‌گر a به شکل ارجاع ارسال نشده است ولی با وجود این می‌توان عناصر a را درون تابع به درستی چاپ کرد؟

7-10 اشاره‌گر ثابت

«اشاره‌گر به یک ثابت» با «اشاره‌گر ثابت» تفاوت دارد. این تفاوت در قالب مثال زیر نشان داده شده است.

× مثال 7-16 اشاره‌گرهای ثابت و اشاره‌گرهایی به ثابت‌ها

در این کد چهار اشاره‌گر اعلان شده. اشاره‌گر p، اشاره‌گر ثابت cp، اشاره به یک ثابت pc، اشاره‌گر ثابت به یک ثابت cpc :

```
int n = 44;                    // an int
int* p = &n;                   // a pointer to an int
++(*p);                        // OK: increments int *p
++p;                           // OK: increments pointer p
int* const cp = &n;            // a const pointer to an int
++(*cp);                       // OK: increments int *cp
++cp;                          // illegal: pointer cp is const
const int k = 88;              // a const int
const int * pc = &k;           // a pointer to a const int
++(*pc);                       // illegal: int *pc is const
++pc;                          // OK: increments pointer pc
const int* const cpc = &k;     // a const pointer to a const int
++(*cpc);                      // illegal: int *pc is const
++cpc;                         // illegal: pointer cpc is const
```

اشاره‌گر p اشاره‌گری به متغیر n است. هم خود p قابل افزایش است (++p) و هم مقداری که p به آن اشاره می‌کند قابل افزایش است ((*P) ++). اشاره گر cp یک اشاره‌گر ثابت است. یعنی آدرسی که در cp است قابل تغییر نیست ولی مقداری که در آن آدرس است را می‌توان دست‌کاری کرد. اشاره‌گر pc اشاره‌گری است که به آدرس یک ثابت اشاره دارد. خود pc را می‌توان تغییر داد ولی مقداری که pc به آن اشاره دارد قابل تغییر نیست. در آخر هم cpc یک اشاره‌گر ثابت به یک شیء ثابت است. نه مقدار cpc قابل تغییر است و نه مقداری که آدرس آن در cpc است.

7-11 آرایه‌ای از اشاره‌گرها

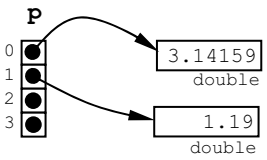
می‌توانیم آرایه‌ای تعریف کنیم که اعضای آن از نوع اشاره‌گر باشند. مثلا دستور:

```
float* p[4];
```

آرایهٔ p را با چهار عنصر از نوع float* (یعنی اشاره‌گری به float) اعلان می‌کند. عناصر این آرایه را مثل اشاره‌گرهای معمولی می‌توان مقداردهی کرد:

```
p[0] = new float(3.14159);
p[1] = new float(1.19);
```

این آرایه را می توانیم شبیه شکل مقابل مجسم کنیم:



مثال بعد نشان می‌دهد که آرایه‌ای از اشاره‌گرها به چه دردی می‌خورد. از این آرایه می‌توان برای مرتب‌کردن یک فهرست نامرتب به روش حبابی استفاده کرد. به جای این که خود عناصر جابجا شوند، اشاره‌گرهای آن‌ها جابجا می‌شوند.

×مثال 7-17 مرتب‌سازی حبابی غیرمستقیم

```
void sort(float* p[], int n)
{  float* temp;
   for (int i = 1; i < n; i++)
      for (int j = 0; j < n-i; j++)
         if (*p[j] > *p[j+1])
         {  temp = p[j];
            p[j] = p[j+1];
            p[j+1] = temp;
         }
}
```

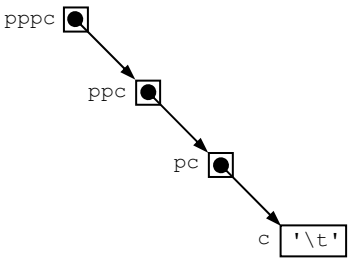
تابع sort() آرایه‌ای از اشاره‌گرها را می‌گیرد. سپس درون حلقه‌های تودرتوی for بررسی می‌کند که آیا مقادیری که اشاره‌گرهای مجاور به آن‌ها اشاره دارند، مرتب هستند یا نه. اگر مرتب نبودند، جای اشاره‌گرهای آن‌ها را با هم عوض می‌کند. در پایان به جای این که یک فهرست مرتب داشته باشیم، آرایه‌ای داریم که اشاره‌گرهای درون آن به ترتیب قرار گرفته اند.

7-12 اشاره‌گری به اشاره‌گر دیگر

یک اشاره‌گر می‌تواند به اشاره‌گر دیگری اشاره کند. مثلا:

```
char c = 't';
char* pc = &c;
char** ppc = &pc;
char*** pppc = &ppc;
***pppc = 'w';   // changes value of c to 'w'
```

حالا pc اشاره‌گری به متغیر کاراکتری c است. ppc اشاره‌گری به اشاره‌گر pc است و اشاره‌گر pppc هم به اشاره‌گر ppc اشاره دارد. مثل شکل مقابل:



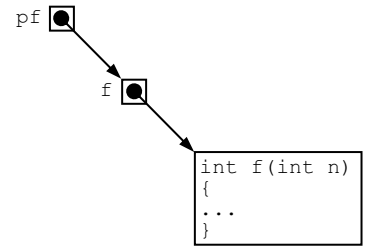
با این وجود می‌توان با اشاره‌گر pppc مستقیما به متغیر c رسید.

7-13 اشاره‌گر به توابع

این بخش ممکن است کمی عجیب به نظر برسد. حقیقت این است که نام یک تابع مثل نام یک آرایه، یک اشاره‌گر ثابت است. نام تابع، آدرسی از حافظه را نشان می‌دهد که کدهای درون تابع در آن قسمت جای گرفته‌اند. پس بنابر قسمت قبل اگر اشاره‌گری به تابع اعلان کنیم، در اصل اشاره‌گری به اشاره‌گر دیگر تعریف کرده‌ایم. اما این تعریف، نحو متفاوتی دارد:

```
int f(int);       // declares function f
int (*pf)(int);   // declares function pointer pf
pf = &f;          // assigns address of f to pf
```

اشاره‌گر pf همراه با * درون پرانتز قرار گرفته، یعنی این که pf اشاره‌گری به یک تابع است. بعد از آن یک int هم درون پرانتز آمده است، به این معنی که تابعی که pf به آن اشاره می‌نماید، پارامتری از نوع int دارد. اشاره‌گر pf را می‌توانیم به شکل زیر تصور کنیم:



فایدهٔ اشاره‌گر به توابع این است که به این طریق می‌توانیم توابع مرکب بسازیم. یعنی می‌توانیم یک تابع را به عنوان آرگومان به تابع دیگر ارسال کنیم! این کار با استفاده از اشاره‌گر به تابع امکان پذیر است.

× مثال 18-7 تابع مرکب جمع

تابع sum() در این مثال دو پارامتر دارد: اشاره‌گر تابع pf و عدد صحیح n:

```
int sum(int (*)(int), int);
int square(int);
int cube(int);

int main()
{  cout << sum(square,4) << endl;      // 1 + 4 + 9 + 16
   cout << sum(cube,4) << endl;        // 1 + 8 + 27 + 64
}
```

تابع sum() یک پارامتر غیر معمول دارد. نام تابع دیگری به عنوان آرگومان به آن ارسال شده. هنگامی که sum(square,4) فراخوانی شود، مقدار square(1)+square(2)+square(3)+square(4) بازگشت داده می‌شود. چون square(k) مقدار k*k را برمی‌گرداند، فراخوانی sum(square,4) مقدار 1+4+9+16=30 را محاسبه نموده و بازمی‌گرداند. تعریف توابع و خروجی آزمایشی به شکل زیر است:

```
int sum(int (*pf)(int k), int n)
{  // returns the sum f(0) + f(1) + f(2) + ... + f(n-1):
   int s = 0;
   for (int i = 1; i <= n; i++)
      s += (*pf)(i);
   return s;
}

int square(int k)
{  return k*k;
}
```

```
int cube(int k)
{  return k*k*k;
}
```

```
30
100
```

pf در فهرست پارامترهای تابع sum() یک اشاره‌گر به تابع است. اشاره‌گر به تابعی که آن تابع پارامتری از نوع int دارد و مقداری از نوع int را برمی‌گرداند. k در تابع sum اصلا استفاده نشده اما حتما باید قید شود تا کامپایلر بفهمد که pf به تابعی اشاره دارد که پارامتری از نوع int دارد. عبارت (*pf)(i) معادل با square(i) یا cube(i) خواهد بود، بسته به این که کدام یک از این دو تابع به عنوان آرگومان به sum() ارسال شوند.

نام تابع، آدرس شروع تابع را دارد. پس square آدرس شروع تابع square() را دارد. بنابراین وقتی تابع sum() به شکل sum(square,4) فراخوانی شود، آدرسی که درون square است به اشاره‌گر pf فرستاده می‌شود. با استفاده از عبارت (*pf)(i) مقدار i به آرگومان تابعی فرستاده می‌شود که pf به آن اشاره دارد.

7-14 NUL و NULL

ثابت صفر (0) از نوع int است اما این مقدار را به هر نوع بنیادی دیگر می‌توان تخصیص داد:

```
char c = 0;         // initializes c to the char '\0'
short d = 0;        // initializes d to the short int 0
int n = 0;          // initializes n to the int 0
unsigned u = 0;     // initializes u to the unsigned int 0
float x = 0;        // initializes x to the float 0.0
double z = 0;       // initializes z to the double 0.0
```

مقدار صفر معناهای گوناگونی دارد. وقتی برای اشیای عددی به کار رود، به معنای عدد صفر است. وقتی برای اشیای کاراکتری به کار رود، به معنای ***کاراکتر تهی*** یا NUL است. NUL معادل کاراکتر '\0' نیز هست. وقتی مقدار صفر برای اشاره‌گرها به کار رود، به معنای «هیچ چیز» یا NULL است. NULL یک کلمهٔ کلیدی است و

کامپایلر آن را می‌شناسد. هنگامی که مقدار NULL یا صفر در یک اشاره‌گر قرار می‌گیرد، آن اشاره‌گر به خانه 0x0 در حافظه اشاره دارد. این خانهٔ حافظه، یک خانهٔ استثنایی است که قابل پردازش نیست. نه می‌توان آن خانه را مقداریابی کرد و نه می‌توان مقداری را درون آن قرار داد. به همین دلیل به NULL «هیچ چیز» می‌گویند.

وقتی اشاره‌گری را بدون استفاده از new اعلان می‌کنیم، خوب است که ابتدا آن را NULL کنیم تا مقدار زبالهٔ آن پاک شود. اما همیشه باید به خاطر داشته باشیم که اشاره‌گر NULL را نباید مقداریابی نماییم:

```
int* p = 0;     // p points to NULL
xp = 22;        // ERROR: cannot dereference the NULL pointer
```

پس خوب است هنگام مقداریابی اشاره‌گرها، احتیاط کرده و بررسی کنیم که آن اشاره‌گر NULL نباشد:

```
if (p) *p = 22;   // O.K.
```

حالا دستور *p=22; وقتی اجرا می‌شود که p صفر نباشد. می‌دانید که شرط بالا معادل شرط زیر است:

```
if (p != NULL) *p = 22;
```

اشاره‌گرها را نمی‌توان نادیده گرفت. آن‌ها سرعت پردازش را زیاد می‌کنند و کدنویسی را کم. با استفاده از اشاره‌گرها می‌توان به بهترین شکل از حافظه استفاده کرد. با به کارگیری اشاره‌گرها می‌توان اشیایی پیچیده‌تر و کارآمدتر ساخت. این موضوع در قالب «ساختمان داده‌ها» بحث می‌شود و یکی از مهم‌ترین مباحث در برنامه‌نویسی است. همچنین اشاره‌گرها در برنامه‌نویسی سطح پایین و برای دست‌کاری منابع سخت‌افزاری نیز فراوان به کار می‌روند. اما اشاره‌گرها امنیت برنامه را به خطر می‌اندازند. می‌توانند به راحتی برنامهٔ حاضر یا برنامه‌هایی که هم‌زمان با برنامهٔ فعلی در حال کارند را خراب کنند. نمونه‌ای از این بحران را در مثال‌های این فصل دیدیم. اگر می‌خواهید از مزایای اشاره‌گرها بهره ببرید و کارآیی برنامه‌تان را بهبود ببخشید، همیشه باید محتاطانه با اشاره‌گرها برخورد کنید. این فقط وقتی ممکن است که اشاره‌گرها را خوب شناخته باشید و برای استفاده از آن‌ها تسلط کافی داشته باشید.

پرسش‌های گزینه‌ای

1 – برای به دست آوردن آدرس متغیر n از کدام گزینه استفاده می شود؟

الف) &n ب) *n ج) n د) (n)

2 – در مورد دستور int& k=n; کدام گزینه صحیح است؟

الف) k متغیری مستقل از n است که مقدار n را دارد

ب) k متغیری مستقل از n است که آدرس n را دارد

ج) k یک ارجاع به n است

د) k یک اشاره‌گر به n است

3 – حاصل اجرای کد int& k=37; چیست؟

الف) مقدار 37 در متغیر مستعار k قرار می گیرد

ب) مقدار 37 در متغیری که k نام مستعار آن است، قرار می‌گیرد

ج) مقدار 37 در متغیری که آدرس آن در k است، قرار می‌گیرد

د) کامپایلر خطا می‌گیرد زیرا ارجاع‌ها را نمی‌توان با مقدار صریح مقداردهی کرد

4 – در مورد دستور int& c=n; کدام گزینه صحیح است؟

الف) c و n آدرس‌های یکسان و مقدارهای یکسان دارند

ب) c و n آدرس‌های متفاوت و مقدارهای متفاوت ولی نوع یکسان دارند

ج) c و n آدرس‌های یکسان ولی مقدارهای متفاوت دارند

د) c و n آدرس‌های متفاوت ولی مقدارهای یکسان دارند

5 – در رابطه با عبارت float* p=k; کدام گزینه صحیح است؟

الف) اشاره‌گر p متغیر مستقلی است که مقدار k در آن ذخیره می‌شود

ب) اشاره‌گر p متغیر مستقلی است که آدرس k در آن ذخیره می‌شود

ج) اشاره‌گر p متغیر مستقلی است که نام k در آن ذخیره می‌شود

د) اشاره‌گر p متغیر مستقل نیست و فقط یک نام مستعار برای متغیر k است

6 – اگر nt از نوع صحیح با مقدار 50 و آدرس 0x000cc70 باشد و سپس اشاره‌گر p به شکل int* p=&nt; اعلان شود، آنگاه حاصل عبارت cout << p; چه خواهد بود؟

الف) مقدار 50 در خروجی چاپ می‌شود
ب) مقدار 0x000cc70 در خروجی چاپ می‌شود
ج) عبارت "&nt" در خروجی چاپ می‌شود
د) آدرس p در خروجی چاپ می‌شود

7 – اگر k اشاره‌گری به نوع float باشد و بخواهیم l را طوری تعریف کنیم که به k اشاره کند، آنگاه :

الف) l باید از نوع float* تعریف شود
ب) l باید از نوع float** تعریف شود
ج) l باید از نوع float تعریف شود
د) l باید از نوع float& تعریف شود

8 – اگر pt یک اشاره‌گر باشد، آنگاه حاصل عبارت زیر چیست؟

```
cout << pt << endl << *pt << endl << &pt ;
```

الف) روی سطر اول آدرس درون pt و روی سطر دوم مقداری که pt به آن اشاره دارد و روی سطر سوم آدرس خود pt چاپ می‌شود
ب) روی سطر اول مقداری که pt به آن اشاره می‌کند و روی سطر دوم آدرس درون pt و روی سطر سوم آدرس خود pt چاپ می‌شود
ج) روی سطر اول آدرس خود pt و روی سطر دوم مقداری که pt به آن اشاره می‌کند و روی سطر سوم آدرس درون pt چاپ می‌شود
د) روی سطر اول آدرس درون pt و روی سطر دوم آدرس خود pt و روی سطر سوم مقداری که pt به آن اشاره دارد چاپ می‌شود

9 – اگر n یک متغیر از نوع بنیادی باشد آنگاه :

الف) *(&n)=n
ب) &(*n)=n
ج) *n=n
د) &n=n

10 – کدام یک از اعلان‌های زیر، تابع f() را به شکل بازگشت از نوع ارجاع اعلان می‌کند؟

الف) int f(int& k)
ب) int* f(int k)
ج) int& f(int k)
د) void f(int& k)

11 – اگر p از نوع float* باشد و sizeof(float)=4 باشد، آنگاه حاصل عبارت ;p++ چیست؟

الف) آدرس درون p یک بایت افزایش می‌یابد

ب) آدرس درون p چهاربایت افزایش می‌یابد

ج) آدرس درون p دو بایت افزایش می‌یابد

د) کامپایلر خطا می‌گیرد زیرا اشاره‌گرها را نمی‌توان افزایش داد

12 – کدام گزینه صحیح نیست؟

الف) اشاره‌گر به یک شیء ثابت (const) را نمی‌توان آزاد کرد

ب) با استفاده از عملگر new می‌توان یک آدرس آزاد را به یک اشاره‌گر اختصاص داد

ج) با استفاده از عملگر delete می‌توان آدرس تخصیص‌یافته به یک اشاره‌گر را آزاد کرد

د) پس از آزاد کردن یک اشاره‌گر هنوز می‌توان به آدرس آن دستیابی نمود

13 – اگر عملگر new نتواند یک آدرس آزاد پیدا کند آنگاه

الف) مقدار NULL را برمی‌گرداند

ب) مقدار NUL را برمی‌گرداند

ج) خطای زمان اجرا رخ می‌دهد و برنامه متوقف می‌شود

د) دوباره تلاش می‌کند تا این که یک فضای آزاد بیابد

14 – کدام عبارت صحیح است؟

الف) آرایه پویا در زمان اجرا ایجاد می‌شود ولی آرایه ایستا در زمان کامپایل ایجاد می‌شود

ب) آرایه پویا را در هر زمانی می‌توان آزاد کرد ولی آرایه ایستا تا پایان برنامه باقی می‌ماند

ج) اندازه آرایه پویا متغیر است ولی اندازه آرایه ایستا ثابت است

د) همه موارد فوق صحیح است

15 – در رابطه با عبارت float* t[10]; کدام عبارت صحیح نیست؟

الف) آرایه t یک آرایه پویا است

ب) آرایه t یک آرایه ایستا است

ج) اندازه آرایه t در هنگام اجرا قابل تغییر است

د) آرایه t در زمان اجرا ساخته می‌شود

پرسش‌های تشریحی

1- چگونه به آدرس حافظهٔ یک متغیر دستیابی پیدا می‌کنید؟

2- چگونه به محتویات مکانی از حافظه که آدرس آن در یک متغیر اشاره‌گر ذخیره شده است دستیابی می‌کنید؟

3- تفاوت بین دو اعلان زیر را شرح دهید:

```
int n1=n;
int& n2=n;
```

4- تفاوت بین دو نحوهٔ استفاده از عملگر ارجاع & در کدهای زیر را شرح دهید:

```
int& r = n;
p = &n;
```

5- تفاوت بین دو نحوهٔ استفاده از عملگر اشاره * در کدهای زیر را شرح دهید:

```
int* q = p;
n = *p;
```

6- درست یا نادرست است؟ توضیح دهید:

الف) اگر (x == y) آنگاه (&x == &y)

ب) اگر (x == y) آنگاه (*x == *y)

7- الف) یک اشاره‌گر سرگردان چیست؟

ب) از مقداریابی یک اشاره‌گر سرگردان چه نتیجهٔ ناخوشایندی حاصل می‌شود؟

ج) چگونه می‌توان از این نتیجهٔ نامطلوب اجتناب کرد؟

8- چه خطایی در کد زیر است؟

```
int& r = 22;
```

9- چه خطایی در کد زیر است؟

```
int* p = &44;
```

10- چه خطایی در کد زیر است؟

```
char c = 'w';
char p = &c;
```

11- چرا نمی‌توان متغیر ppn مثال 6-7 را شبیه این اعلان کرد:

```
int** ppn = &&n;
```

12- چه تفاوتی بین «بسته‌بندی ایستا» و «بسته‌بندی پویا» است؟

13- چه اشتباهی در کد زیر است؟

```
char c = 'w';
char* p = c;
```

14- چه اشتباهی در کد زیر است؟

```
short a[32];
for (int i = 0; i < 32; i++)
*a++ = i*i;
```

15- مقدار هر یک از متغیرهای زیر را بعد از اجرا شدن کد زیر مشخص کنید. فرض کنید که هرعدد صحیح 4 بایت را اشغال می‌کند و m در محلی از حافظه ذخیره شده که از بایت 0x3fffd00 شروع می‌شود.

```
int m = 44;
int* p = &m;
int& r = m;
int n = (*p++);
int* q = p - 1;
r = *(--p) + 1;
++*q;
```

الف) m ب) n پ) &m ت) *p ث) r ج) *q

16- هر یک از موارد زیر را در دسته‌های چپ‌مقدار تغییرپذیر، چپ‌مقدار تغییرناپذیر یا غیرچپ‌مقدار طبقه بندی کنید:

الف) `double x = 1.23;`

ب) `4.56*x + 7.89`

پ) `const double y = 1.23;`

ت) `double a[8] = {0.0};`

ث) `a[5]`

ج) `double f() { return 1.23 };`

چ) `f(1.23)`

ح) `double& r = x;`

خ) `double* p = &x;`

د) `*p`

ذ) `const double* p = &x;`

ر) `double* const p = &x;`

17- چه اشتباهی در کد زیر است؟

```
float x = 3.14159;
float* p = &x;
short d = 44;
```

```
short* q = &d;
p = q;
```

18- چه اشتباهی در کد زیر است؟

```
int* p = new int;
int* q = new int;
cout << "p = " << p << ", p + q = " << p + q
     << endl;
```

19- تنها کاری که نباید هرگز با اشاره‌گر NULL انجام دهید چیست؟

20- در اعلان زیر توضیح دهید که نوع p چیست و بگویید آن چگونه می‌تواند استفاده شود:

```
double**** p;
```

21- اگر x آدرس 0x3fffd1c داشته باشد، آنگاه مقادیر p و q برای هر یک از موارد زیر چیست؟

```
double x = 1.01;
double* p = &x;
double* q = p + 5;
```

22- اگر p و q اشاره‌گرهایی به int باشند و n متغیری ار نوع int باشد، کدام یک از موارد زیر مجاز است؟

الف) p + q ب) p - q پ) p + n

ت) p - n ث) n + p ج) n - q

23- این گفته که یک آرایه در حقیقت یک اشاره گر ثابت است چه معنی می‌دهد؟

24- وقتی تنها آدرس اولین عنصر یک آرایه به تابع ارسال می‌شود، چگونه است که تابع می‌تواند هر عنصری از آرایه را دستیابی کند؟

25- توضیح دهید چرا سه شرط زیر برای آرایهٔ a و یک عدد صحیح i درست است؟

```
a[i] == *(a + i);
*(a + i) == i[a];
a[i] == i[a];
```

26- تفاوت بین دو اعلان زیر را توضیح دهید:

```
double * f();
double (*f());
```

27- برای هر یک از موارد زیر یک اعلان بنویسید:

الف) یک آرایه از هشت float

ب) یک آرایه از هشت اشاره‌گر به float

پ) یک اشاره‌گر به یک آرایه از هشت float
ت) یک اشاره‌گر به یک آرایه از هشت اشاره‌گر به float
ث) یک تابع که یک float را برمی‌گرداند
ج) یک تابع که یک اشاره‌گر به float را برمی‌گرداند
چ) یک اشاره‌گر به تابع که یک float را برمی‌گرداند
ح) یک اشاره‌گر به یک تابع که یک اشاره‌گر به float را برمی‌گرداند

تمرین‌های برنامه‌نویسی

1- تابعی بنویسید که از اشاره‌گرها برای جستجوی آدرس یک عدد صحیح مفروض در یک آرایه استفاده کند. اگر عدد مفروض پیدا شود تابع آدرس آن را برگرداند و در غیر این صورت null را برگرداند.

2- تابعی بنویسید که n اشاره گر به float دریافت کند و آرایه جدیدی را برگرداند که شامل مقادیر آن n عدد float باشد.

3- کد تابع زیر را بنویسید. این تابع مجموع floatهایی که با n اشاره گر در آرایه p به آن ها اشاره می شود را بر می گرداند.

```
float sum(float* p[], int n);
```

4- کد تابع زیر را بنویسید. این تابع علامت هر یک از flatهای منفی که به وسیله n اشاره گر در آرایه p به آن ها اشاره می شود را تغییر می دهد.

```
void abs(float* p[], int n);
```

5- کد تابع زیر را بنویسید. این تابع به صورت غیر مستقیم floatهای اشاره شده توسط n اشاره گر در آرایه p را با مرتب سازی اشاره گرها مرتب می کند.

```
void sort(float* p[], int n);
```

6- کد تابع زیر را بنویسید. این تابع تعداد بایت های درون s را می شمارد تا این که s به کاراکتر '\0' اشاره کند.

```
unsigned len(const char* s);
```

7- کد تابع زیر را بنویسید. این تابع n بایت اول حافظه با شروع از s2* را داخل بایت های شروع شده از s1* کپی می کند. n تعداد بایت هایی است که s2 می تواند افزایش یابد قبل از این که به کاراکتر '\0' اشاره کند.

```
void cpy(char* s1, const char* s2);
```

8- کد تابع زیر را بنویسید. این تابع حداکثر n بایت با شروع از s2* را با بایت های متناظر با شروع از s1* مقایسه می کند که n تعداد بایت هایی است که s2 می تواند افزایش یابد قبل از این که به کاراکتر '\0' اشاره کند. اگر همه n بایت معادل باشند تابع باید 0 را برگرداند. در غیر این صورت بسته به این که در اولین اختلاف بایت موجود در s1 کوچک تر یا بزرگ تر از بایت موجود در s2 باشد مقدار 1- یا 1 را برگرداند.

```
int cmp(char* s1, char* s2);
```

9- کد تابع زیر را بنویسید. این تابع n بایت با شروع از s را به دنبال کاراکتر c جستجو می کند که n تعداد بایت هایی است که s می تواند افزایش یابد قبل از این که به کاراکتر تهی '\0' اشاره کند. اگر کاراکتر c پیدا شود یک اشاره گر به آن برگردانده می شود و در غیر این صورت NULL برگردانده می شود.

```
char* chr(char* s, char c);
```

10- تابع زیر که حاصل ضرب n مقدار f(1) و f(2) و ... و f(n) را برمی گرداند بنویسید (به مثال 18-7 نگاه کنید).

```
int product(int (*pf)(int k), int n);
```

11- قانون ذوزنقه را برای انتگرال گیری یک تابع پیاده سازی کنید. از تابع زیر استفاده نمایید:

```
double trap(double (*pf)(double x), double a,
            double b, int n);
```

در این جا pf به تابع f که باید انتگرال گیری شود اشاره می کند. a و b فاصله ای است که f در آن فاصله انتگرال گیری می شود و n تعداد زیر فاصله های استفاده شده است. برای مثال فراخوانی trap(square, 1, 2, 100); مقدار 1.41421 را بر می گرداند. قانون زوذنقه مجموع مساحت های ذوزنقه که مساحت تقریبی زیر نمودار f است را برمی گرداند. به طور مثال اگر h = 5 باشد آنگاه تابع مذکور مقدار زیر را بر می گرداند که h = (b-a) /5 پهنای هر یک از ذوزنقه های زیر است:

$$\frac{h}{2}[f(a)+2f(a+h)+2f(a+2h)+2f(a+3h)+2f(a+4h)+f(b)]$$

فصل هشتم

« رشته‌های کاراکتری و فایل‌ها در C++ استاندارد»

8-1 مقدمه

داده‌هایی که در رایانه‌ها پردازش می‌شوند همیشه عدد نیستند. معمولا لازم است که اطلاعات کاراکتری مثل نام افراد – نشانی‌ها – متون – توضیحات – کلمات و ... نیز پردازش گردند، جستجو شوند، مقایسه شوند، به یکدیگر الصاق شوند یا از هم تفکیک گردند. در این فصل بررسی می‌کنیم که چطور اطلاعات کاراکتری را از ورودی دریافت کنیم و یا آن‌ها را به شکل دلخواه به خروجی بفرستیم. در همین راستا توابعی معرفی می‌کنیم که انجام این کارها را آسان می‌کنند.

یک ***رشتهٔ کاراکتری*** (که به آن رشتهٔ C نیز می‌گویند) یک سلسله از کاراکترهای کنار هم در حافظه است که با کاراکتر NUL یعنی `'\0'` پایان یافته است. یک رشتهٔ کاراکتری را می‌توان به وسیلهٔ متغیرهایی از نوع `char*` (یعنی اشاره‌گری به `char`) دستیابی نمود. چون آرایه‌ها بسیار شبیه اشاره‌گرها رفتار می‌کنند، از آرایه‌هایی با نوع `char` نیز می‌توان برای پردازش رشته‌های کاراکتری استفاده کرد.

سرفایل `<cstring>` توابع ویژهٔ زیادی دارد که این توابع ما را در دست‌کاری رشته‌های کاراکتری یاری می‌کنند. مثلا تابع `strlen(s)` تعداد کاراکترهای رشتهٔ

کاراکتری S بدون کاراکتر پایانی NUL را برمی‌گرداند. همهٔ توابع مذکور دارای پارامترهایی از نوع char* هستند. پس قبل از این که رشته‌های کاراکتری را مطالعه نماییم، به اختصار اشاره‌گرها را مرور می‌کنیم.

8-2 مروری بر اشاره‌گرها

یک اشاره‌گر متغیری است که حاوی یک آدرس از حافظه می‌باشد. نوع این متغیر از نوع مقداری است که در آن آدرس ذخیره شده. با استفاده از عملگر ارجاع & می‌توان آدرس یک شی را پیدا کرد. همچنین با استفاده از عملگر مقداریابی * می‌توانیم مقداری که در یک آدرس قرار دارد را مشخص کنیم. به تعاریف زیر نگاه کنید:

```
int n = 44;
int* p = &n;
```

p اشاره‌گری از نوع int* است. یعنی اشاره‌گری که به یک مقدار int اشاره دارد. عبارت &n آدرس متغیر n را به دست می‌آورد. این آدرس درون اشاره‌گر p قرار گرفته است. حالا اشاره‌گر p به n اشاره می‌کند. بنابراین دستور

```
*p = 55;
```

مقدار n را به 55 تغییر می‌دهد.

یک اشاره‌گر، یک متغیر مستقل است که فضای ذخیره‌سازی مجزایی دارد. مثلا در کد بالا اگر آدرس n هنگام اجرا 64fddc باشد، این مقدار درون اشاره‌گر p ذخیره می‌شود. خود p می‌تواند دارای آدرس 64ff19 باشد که مقدار 64fddc درون آن آدرس قرار گرفته.

یک شی می‌تواند چندین اشاره‌گر داشته باشد. مثلا دستور

```
float* q = &n;
```

اشاره‌گر دیگری به نام q اعلان می‌کند که این هم به n اشاره دارد. حالا هم p و هم q به n اشاره می‌کنند. البته p و q دو اشاره‌گر مجزا و مستقل از هم هستند. با دستور

```
cout << *p;
```

مقدار متغیری که p به آن اشاره دارد، چاپ می‌شود. همچنین با دستور

```
cout << p;
```

آدرس متغیری که p به آن اشاره دارد چاپ می‌شود. البته اکنون به یک استثنا برمی‌خوریم. اگر p از نوع char* باشد، حاصل cout << p; طور دیگری خواهد بود.

8-3 رشته‌های کاراکتری در C

در زبان C++ یک «رشتهٔ کاراکتری» آرایه‌ای از کاراکترهاست که این آرایه دارای ویژگی مهم زیر است:

1 - یک بخش اضافی در انتهای آرایه وجود دارد که مقدار آن، کاراکتر NUL یعنی '\0' است. پس تعداد کل کاراکترها در آرایه همیشه یکی بیشتر از طول رشته است.

2 – رشتهٔ کاراکتری را می‌توان با لیترال رشته‌ای به طور مستقیم مقدارگذاری کرد مثل: `char str[] = "string";`

توجه کنید که این آرایه هفت عنصر دارد: 's' و 't' و 'r' و 'i' و 'n' و 'g' و '\0'.

3 – کل یک رشتهٔ کاراکتری را می‌توان مثل یک متغیر معمولی چاپ کرد. مثل:

```
cout << str;
```

در این صورت، همهٔ کاراکترهای درون رشتهٔ کاراکتری str یکی یکی به خروجی می‌روند تا وقتی که به کاراکتر انتهایی NUL برخورد شود.

4 – یک رشتهٔ کاراکتری را می‌توان مثل یک متغیر معمولی از ورودی دریافت کرد مثل: `cin >> str;`

در این صورت، همهٔ کاراکترهای وارد شده یکی یکی درون str جای می‌گیرند تا وقتی که به یک فضای خالی در کاراکترهای ورودی برخورد شود. برنامه‌نویس باید مطمئن باشد که آرایهٔ str برای دریافت همهٔ کاراکترهای وارد شده جا دارد.

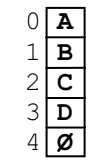
5 – توابع تعریف شده در سرفایل <cstring> را می‌توانیم برای دست‌کاری رشته‌های کاراکتری به کار بگیریم. این توابع عبارتند از: تابع طول رشته strlen()، توابع کپی رشته strcpy() و strncpy()، توابع الصاق رشته‌ها strcat() و strncat()، توابع مقایسهٔ رشته‌ها strcmp() و strncmp() و تابع استخراج نشانه strtok(). این توابع در بخش 8-8 توضیح داده می‌شوند.

× مثال 8-1 رشته‌های کاراکتری با کاراکتر NUL خاتمه می‌یابند

برنامهٔ کوچک زیر نشان می‌دهد که کاراکتر '\0' به رشته‌های کاراکتری الصاق می‌شود:

```
int main()
{  char s[] = "ABCD";
   for (int i = 0; i < 5; i++)
      cout << "s[" << i << "] = '" << s[i] << "'\n";
}
```

```
s[0] = 'A'
s[1] = 'B'
s[2] = 'C'
s[3] = 'D'
s[4] = ''
```



رشتهٔ کاراکتری s دارای پنج عضو است که عضو پنجم، کاراکتر '\0' می‌باشد. تصویر خروجی این مطلب را تایید می‌نماید. وقتی کاراکتر '\0' به cout فرستاده می‌شود، هیچ چیز چاپ نمی‌شود. حتی جای خالی هم چاپ نمی‌شود. خط آخر خروجی، عضو پنجم را نشان می دهد که میان دو علامت آپستروف هیچ چیزی چاپ نشده.

8-4 ورودی/خروجی رشته‌های کاراکتری

در C++ به چند روش می‌توان رشته‌های کاراکتری را دریافت کرده یا نمایش داد. یک راه استفاده از عملگرهای کلاس string است که در بخش‌های بعدی به آن

خواهیم پرداخت. روش دیگر، استفاده از توابع کمکی است که آن را در ادامه شرح می‌دهیم.

× مثال 2-8 روش سادهٔ دریافت و نمایش رشته‌های کاراکتری

در برنامهٔ زیر یک رشتهٔ کاراکتری به طول 79 کاراکتر اعلان شده و کلماتی که از ورودی خوانده می‌شود در آن رشته قرار می‌گیرد:

```
int main()
{  char word[80];
   do
   {  cin >> word;
      if (*word) cout << "\t\"" << word << "\"\n";
   } while (*word);
}
```

```
Today's date is April 1, 2005.
        "Today's"
        "date"
        "is"
        "April"
        "1,"
        "2005."
Tomorrow is Saturday.
        "Tomorrow"
        "is"
        "Saturday."
^z
```

وقتی اجرای برنامه به دستور cin برسد، همان جا متوقف می‌شود تا این که کاربر کاراکترهایی را تایپ کرده و سپس کلید Enter را فشار دهد. پس از فشردن کلید Enter همهٔ کاراکترهای تایپ شده به یک حافظهٔ میانی منتقل می‌شود و عملگر cin از آن حافظه کاراکترها را برداشت می‌کند تا این که به کاراکتر فضای خالی یا کاراکتر پایان خط برسد. سپس برنامه ادامه می‌یابد. وقتی دوباره برنامه به cin برخورد کند به همان حافظهٔ میانی مراجعه می‌نماید تا مجموعهٔ کاراکترها را تا کاراکتر فضای خالی بعدی برداشت کند. اگر cin به کاراکتر پایان خط برسد، حافظهٔ میانی خالی شده است و در اجرای بعدی cin دوباره منتظر می‌ماند تا کاربر کاراکترهایی را وارد نماید.

در نمونه اجرای بالا، حلقه while ده مرتبه تکرار می‌شود و هر بار یک کلمه را می‌خواند (خواندن Ctrl+Z نیز در یک تکرار انجام می‌گیرد). هر بار که کاربر عبارتی را تایپ کند و کلید Enter را فشار دهد، برنامه کلمات آن عبارت را یکی یکی خوانده و آن کلمات را میان دو علامت نقل قول "" چاپ می‌کند. وقتی همهٔ کلمات تمام شدند، برنامه منتظر می‌ماند تا کاربر عبارت جدیدی را تایپ کرده و کلید Enter را فشار دهد.

عبارت word* حلقه را کنترل می‌کند. چون word یک آرایه است، word* به اولین خانهٔ آرایه اشاره می‌کند. بنابراین مقدار عبارت word* تنها وقتی صفر (یعنی false) است که رشتهٔ کاراکتری word یک رشتهٔ خالی باشد که فقط شامل '\0' است. به یک رشتهٔ کاراکتری، ***خالی*** می‌گویند اگر اولین عنصر آن، کاراکتر NUL باشد.

بنابراین، برنامهٔ بالا هیچ وقت متوقف نمی‌شود مگر این که کاراکتر '\0' وارد شود. این کار ممکن نیست مگر این که کلید Ctrl+Z را فشار دهید. در این صورت کاراکتر تهی به ابتدای رشتهٔ کاراکتری word فرستاده می‌شود و به این ترتیب حلقه پایان می‌یابد. عمل حلقهٔ مثال 2-8 می‌تواند با کد زیر جایگزین شود:

```
cin >> word
while (*word)
{  cout << "\t\"" << word << "\"\n";
   cin >> word;
}
```

توجه کنید که کاراکترهای نقطه‌گذاری (آپستروف، نقطه، ویرگول ، کاما و ...) می‌توانند در رشتهٔ کاراکتری فوق وجود داشته باشند ولی کاراکترهای فضای سفید (جای خالی، پرش‌ها، خط جدید و ...) چنین نیستند.

می‌بینید که عملگر خروجی >> با اشاره‌گر نوع char* رفتار متفاوتی دارد. این عملگر وقتی به اشاره‌گر نوع char* برخورد کند کل رشته‌ای که اشاره‌گر به آن اشاره می‌کند را چاپ می‌نماید. در صورتی که با اشاره‌گری از نوع دیگر، فقط آدرس درون آن اشاره‌گر را چاپ می‌نماید.

8-5 چند تابع عضو cin و cout

به cin شیء فرآیند ورودی می‌گویند. این شی شامل توابع زیر است:

cin.getline() , cin.get() , cin.ignore() , cin.putback() , cin.peek()

همهٔ این توابع شامل پیشوند cin هستند زیرا آن‌ها عضوی از cin می‌باشند. به cout شیء فرآیند خروجی می‌گویند. این شی نیز شامل تابع cout.put() است. نحوهٔ کاربرد هر یک از این توابع عضو را در ادامه خواهیم دید.

فراخوانی cin.getline(str,n); باعث می‌شود که n کاراکتر به درون str خوانده شود و مابقی کاراکترهای وارد شده نادیده گرفته می‌شوند.

× مثال 8-3 تابع cin.getline() با دو پارامتر

این برنامه ورودی را خط به خط به خروجی می‌فرستد:

```
int main()
{  char line[80];
   do
   {  cin.getline(line,80);
      if (*line) cout << "\t[" << line << "]\n";
   } while (*line);
}
```

شرط کنترل حلقه به شکل (*line) است. این شرط فقط وقتی true است که رشتهٔ کاراکتری line یک رشتهٔ کاراکتری غیر تهی باشد. هر بار که حلقه اجرا می‌گردد، 80 کاراکتر به درون رشتهٔ کاراکتری line خوانده می‌شود.

تابع getline() نسخهٔ دیگری هم دارد. وقتی این تابع با سه پارامتر به شکل

```
cin.getline(str, n, ch);
```

فراخوانی شود، همهٔ کاراکترهای ورودی به درون رشتهٔ کاراکتری str منتقل می‌شوند تا این که تعدادشان به n کاراکتر برسد یا این که به کاراکتر خاص ch برخورد شود. به کاراکتر ch که باعث تفکیک کاراکترهای ورودی می‌شود، «کاراکتر مرزبندی» می‌گویند. وقتی کاراکتر پایان خط به عنوان کاراکتر مرزبندی در نظر گرفته شود، نیازی

نیست که در تابع getline() قید شود و به این ترتیب تابع مذکور فقط با دو پارامتر فراخوانی می‌شود.

در مثال بعدی کاراکتر کاما ',' به عنوان کاراکتر مرزبندی فرض شده است.

× مثال 4-8 تابع cin.getline() با سه پارامتر

برنامه زیر، متن ورودی را جمله به جمله تفکیک می‌نماید:

```
int main()
{  char clause[20];
   do
   {  cin.getline(clause, 20, ',');
      if (*clause) cout << "\t[" << clause << "]\n";
   } while (*clause);
}
```

```
Once upon a midnight dreary, while I pondered, weak and weary,
Over many a quaint and curious volume of forgotten lore,
^z  [Once upon a midnight dreary]
    [ while I pondered]
    [ weak and weary]
    [
Over many a quaint and curious volume of forgotten lore]
    [
]
```

در برنامهٔ بالا چون کاراکتر کاما ',' به عنوان کاراکتر مرزبندی منظور شده، هر کاراکتر دیگری به عنوان یک کاراکتر معمولی فرض می‌شود. مثلا بعد از عبارت "weary," کاراکتر پایان خط به صورت مخفی وجود دارد اما چون فقط کاراکتر کاما برای مرزبندی منظور شده، کاراکتر پایان خط به عنوان یک کاراکتر معمولی در ورودی بعدی استفاده و پردازش می‌شود. توجه داشته باشید هر دفعه که به کاراکتر مرزبندی برخورد شود، آن کاراکتر حذف می‌شود و ورودی بعدی از کاراکتر بعد از آن شروع می‌کند.

تابع cin.get() برای خواندن یک کاراکتر از ورودی به کار می‌رود. فراخوانی cin.get(ch) باعث می‌شود که کاراکتر بعدی از ورودی cin خوانده شده و به داخل متغیر ch کپی شود. اگر این کار موفقیت آمیز باشد، مقدار 1 بازگشت داده می‌شود. اگر به کاراکتر پایان خط برخورد شود، مقدار 0 بازگردانده می‌شود.

× مثال 8-5 تابع cin.get()

این برنامه تعداد حرف 'e' در جریان ورودی را شمارش می‌کند. تا وقتی cin.get(ch) کاراکترها را با موفقیت به درون ch می‌خواند، حلقه ادامه می‌یابد:

```
int main()
{  char ch;
   int count = 0;
   while (cin.get(ch))
      if (ch = = 'e') ++count;
   cout << count << " e's were counted.\n";
}
```

```
Once upon a midnight dreary, while I pondered, weak and weary,
Over many a quaint and curious volume of forgotten lore,
^z
11 e's were counted.
```

معکوس تابع ورودی get() تابع خروجی put() است. تابع cout.put() برای نوشتن یک کاراکتر در خروجی به کار می‌رود. به مثال بعدی نگاه کنید.

× مثال 8-6 تابع cout.put()

برنامهٔ زیر، اولین حرف از هر کلمهٔ ورودی را به حرف بزرگ تبدیل کرده و آن را مجددا در خروجی چاپ می‌کند:

```
int main()
{  char ch, pre = '\0';
   while (cin.get(ch))
   {  if (pre = = ' ' || pre = = '\n')
         cout.put(char(toupper(ch)));
      else cout.put(ch);
      pre = ch;
   }
}
```

```
Fourscore and seven years ago our fathers
Fourscore And Seven Years Ago Our Fathers
brought forth upon this continent a new nation.
Brought Forth Upon This Continent A New Nation.
^z
```

متغیر pre کاراکتر خوانده شدهٔ قبلی را نگه می‌دارد. منطق برنامه این گونه است که اگر pre یک کاراکتر خالی یا کاراکتر خط جدید باشد، آنگاه کاراکتری که در ch خوانده می‌شود اولین حرف از کلمهٔ بعدی است. در این حالت کاراکتر درون ch با حرف بزرگ جایگزین می‌شود.

تابع toupeer(ch) باعث می‌شود که اگر ch حرف کوچک باشد، آن را به حرف بزرگ معادلش تبدیل کند. این تابع در سرفایل <ctype.h> تعریف شده. البته به جای این تابع می‌توانیم عبارت زیر را به کار ببریم:

```
ch += 'A' - 'a';
```

تابع cin.putback() آخرین کاراکتر خوانده شده توسط cin.get() را به جریان ورودی باز پس می‌فرستد و به این ترتیب دستور cin بعدی می‌تواند دوباره آن کاراکتر را بخواند.

تابع cin.ignore() یک یا چند کاراکتر را بدون آن که آن‌ها را پردازش کند، نادیده گرفته و از روی آن ها پرش می‌کند. به مثال بعدی نگاه کنید.

× مثال 7-8 توابع cin.ignore() و cin.putback()

با استفاده از برنامهٔ زیر، تابعی آزمایش می‌شود که این تابع اعداد صحیح را از ورودی استخراج می‌کند:

```
int nextInt();
int main()
{  int m = nextInt(), n = nextInt();
   cin.ignore(80,'\n');          // ignore rest of input line
   cout << m << " + " << n << " = " << m+n << endl;
}
int nextInt()
{  char ch;
   int n;
   while (cin.get(ch))
      if (ch >= '0' && ch <= '9')  // next character is a digit
      {  cin.putback(ch);          // put it back so it can be
         cin >> n;                 // read as a complite int
```

```
         break;
      }
   return n;
}
```

```
What is 305 plus 9416?
305 + 9416 = 9721
```

تابع nextInt() از تمام کاراکترهای موجود در cin گذر می‌کند تا این که به اولین عدد برخورد کند. در اجرای آزمایشی بالا، اولین عدد 3 است. چون این رقم بخشی از عدد صحیح 305 می‌باشد، مجددا به درون cin برگردانده می‌شود و حالا دستور cin بعدی دوباره از همان کاراکتر شروع به خواندن می‌کند و تا فاصلهٔ خالی بعدی، خواندن را ادامه داده و نتیجه را به درون n منتقل می‌نماید. به این ترتیب کل عدد 305 به درون n خوانده می‌شود.

تابع cin.peek() ترکیبی از دو تابع cin.get() و cin.putback() است. یعنی عبارت ch = cin.peek(); کاراکتر بعدی را به درون ch می‌خواند بدون این که آن کاراکتر را از جریان ورودی حذف نماید. مثال زیر، طریقهٔ استفاده از این تابع را نشان می‌دهد.

× مثال 8-8 تابع cin.peek()

این نسخه از تابع nextInt() معادل آن است که در مثال قبلی بود:

```
int nextInt()
{  char ch;
   int n;
   while (ch = cin.peek())
      if (ch >= '0' && ch <= '9')
      {  cin >> n;
         break;
      }
      else cin.get(ch);
   return n;
}
```

عبارت `ch = cin.peek()` کاراکتر بعدی را درون ch می‌خواند و در صورت موفقیت، مقدار 1 را برمی‌گرداند. در خط بعدی، اگر ch یک عدد باشد، عدد صحیح به طور کامل درون n خوانده می‌شود وگرنه آن کاراکتر از cin حذف می‌شود و حلقه ادامه می‌یابد. اگر در کاراکترهای ورودی به کاراکتر پایان فایل برخورد شود، عبارت `ch = cin.peek();` مقدار 0 را برمی‌گرداند و حلقه متوقف می‌شود.

8-6 توابع کاراکتری C استاندارد

در مثال 8-6 به تابع `toupper()` اشاره شد. این فقط یکی از توابعی است که برای دست‌کاری کاراکترها استفاده می‌شود. سایر توابعی که در سرفایل <ctype.h> یا <cctype> تعریف شده به شرح زیر است:

نام تابع	شرح
`isalnum()`	`int isalnum(int c);` اگر c کاراکتر الفبایی یا عددی باشد مقدار غیرصفر وگرنه صفر را برمی‌گرداند
`isalpha()`	`int isalpha(int c);` اگر c کاراکتر الفبایی باشد مقدار غیرصفر و در غیر آن، صفر را برمی‌گرداند
`iscntrl()`	`int iscntrl(int c);` اگر c کاراکتر کنترلی باشد مقدار غیرصفر و در غیر آن، صفر را برمی‌گرداند
`isdigit()`	`int isdigit(int c);` اگر c کاراکتر عددی باشد، مقدار غیرصفر و در غیر آن، صفر را برمی‌گرداند
`isgraph()`	`int isgraph(int c);` اگر c کاراکتر چاپی و غیرخالی باشد مقدار غیرصفر وگرنه صفر را برمی‌گرداند
`islower()`	`int islower(int c);` اگر c حرف کوچک باشد مقدار غیرصفر و در غیر آن، صفر را برمی‌گرداند
`isprint()`	`int isprint(int c);` اگر c کاراکتر قابل چاپ باشد مقدار غیرصفر و در غیر آن، صفر را برمی‌گرداند
`ispunct()`	`int ispunct(int c);` اگر c کاراکتر چاپی به غیر از حروف و اعداد و فضای خالی باشد، مقدار غیرصفر برمی‌گرداند وگرنه مقدار صفر را برمی‌گرداند

isspace()	int isspace(int c); اگر c کاراکتر فضای سفید شامل فضای خالی ' ' و عبور فرم '\f' و خط جدید '\n' و بازگشت نورد '\r' و پرش افقی '\t' و پرش عمودی '\v' باشد، مقدار غیرصفر را برمی‌گرداند وگرنه صفر را برمی‌گرداند
isupper()	int isupper(int c); اگر c حرف بزرگ باشد، مقدار غیرصفر برمی‌گرداند وگرنه صفر را برمی‌گرداند
isxdigit()	int isxdigit(int c); اگر c یکی از ده کاراکتر عددی یا یکی از دوازده حرف عدد شانزده‌دهی شامل 'a' و 'b' و 'c' و 'd' و 'e' و 'f' و 'A' و 'B' و 'C' و 'D' و 'E' و 'F' باشد، مقدار غیرصفر برمی‌گرداند وگرنه مقدار صفر را برمی‌گرداند
tolower()	int tolower(int c); اگر c حرف بزرگ باشد، کاراکتر کوچک معادل آن را برمی‌گرداند وگرنه خود c را برمی‌گرداند
toupper()	int toupper(int c); اگر c حرف کوچک باشد، کاراکتر بزرگ معادل آن را برمی‌گرداند وگرنه خود c را برمی‌گرداند

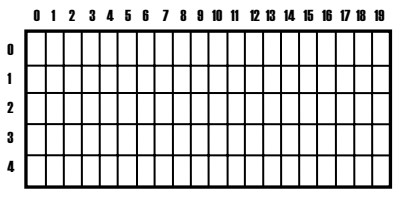
توجه کنید که همهٔ توابع فوق یک پارامتر از نوع int دریافت می‌کنند و یک مقدار int را برمی‌گردانند. علت این است که نوع char در اصل یک نوع صحیح است. در عمل وقتی توابع فوق را به کار می‌برند، یک مقدار char به تابع می‌فرستند و مقدار بازگشتی را نیز در یک char ذخیره می‌کنند. به همین خاطر این توابع را به عنوان «توابع کاراکتری» در نظر می‌گیریم.

7-8 آرایه‌ای از رشته‌ها

به خاطر دارید که گفتیم یک آرایهٔ دوبعدی در حقیقت آرایه‌ای یک بعدی است که هر کدام از اعضای آن یک آرایهٔ یک بعدی دیگر است. مثلا در آرایهٔ دو بعدی که به شکل مقابل اعلان شده باشد:

```
char name[5][20];
```

این آرایه در اصل پنج عضو دارد که هر عضو می‌تواند بیست کاراکتر داشته باشد. اگر آرایهٔ فوق را با تعریف رشته‌های کاراکتری مقایسه کنیم، نتیجه این می‌شود که آرایهٔ



بالا یک آرایهٔ پنج عنصری است که هر عنصر آن یک رشتهٔ کاراکتری بیست حرفی است. این آرایه را می‌توانیم به شکل مقابل تصور کنیم.

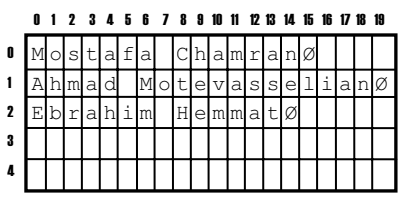
از طریق name[0] و name[1] و name[2] و name[3] و name[4] می‌توانیم به هر یک از رشته‌های کاراکتری در آرایهٔ بالا دسترسی داشته باشیم. یعنی آرایهٔ name گرچه به صورت یک آرایهٔ دوبعدی اعلان شده لیکن به صورت یک آرایهٔ یک بعدی با آن رفتار می‌شود. به مثال بعدی دقت کنید.

× مثال 9-8 آرایه‌ای از رشته‌های کاراکتری

برنامهٔ زیر چند رشتهٔ کاراکتری را از ورودی می‌خواند و آن‌ها را در یک آرایه ذخیره کرده و سپس مقادیر آن آرایه را چاپ می‌کند:

```
int main()
{  char name[5][20];
   int count=0;
   cout << "Enter at most 4 names with at most 19 characters:\n";
   while (cin.getline(name[count++], 20))
      ;
   --count;
   cout << "The names are:\n";
   for (int i=0; i<count; i++)
      cout << "\t" << i << ". [" << name[i] << "]" << endl;
}
```

```
Enter at most 4 names with at most 19 characters:
Mostafa Chamran
Ahmad Motevasselian
Ebrahim Hemmat
^z
The names are:
        0: [Mostafa Chamran]
        1: [Ahmad Motevasselian]
        2: [Ebrahim Hemmat]
```



آرایهٔ name را می‌توان به شکل مقابل تصور نمود. تمام فعالیت‌های حلقهٔ while در قسمت کنترلی آن انجام می‌شود:

```
cin.getline(name[count++],20)
```

وقتی این بخش اجرا می‌شود، تابع cin.getline() خط بعدی را از ورودی دریافت کرده و درون name[count] قرار می‌دهد و سپس count را افزایش می‌دهد. اگر این عمل موفقیت‌آمیز باشد، تابع getline() مقدار غیر صفر را برمی‌گرداند و در نتیجه حلقهٔ while دوباره تکرار می‌شود. اگر کاراکتر پایان فایل (Ctrl+Z) وارد شود، وقتی تابع getline() با آن برخورد کند، متوقف شده و مقدار صفر را برمی‌گرداند و در اثر این مقدار، حلقهٔ while خاتمه می‌یابد. بدنهٔ حلقهٔ while خالی است و هیچ دستوری ندارد. یک سمیکولن تنها در بدنهٔ while این را نشان می‌دهد.

برای ذخیره کردن رشته‌های کاراکتری در یک آرایه، راه بهتری هم هست: «آرایه‌ای از اشاره‌گرها». دستور

```
char* name[4];
```

یک آرایه را نشان می‌دهد که این آرایه چهار عضو از نوع اشاره‌گر به char دارد. یعنی هر name[i] یک رشتهٔ کاراکتری را نشان می‌دهد. مزیت آرایهٔ فوق این است که از ابتدا هیچ حافظه‌ای برای رشته‌های کاراکتری تخصیص نمی‌یابد ولذا طول رشته‌های کاراکتری می‌تواند متفاوت باشد. در عوض مجبوریم که هر رشته را در یک قسمت از حافظه ذخیره کنیم و سپس نشانی اولین خانهٔ آن قسمت را در آرایه قرار دهیم. این روش در مثال 10-8 نشان داده شده. روش مذکور بسیار موثرتر است زیرا برای نگهداری هر رشتهٔ کاراکتری فقط به اندازهٔ همان رشته حافظه تخصیص می‌یابد نه بیشتر. هزینه‌ای که برای این کارایی باید بپردازیم این است که روال ورودی کمی پیچیده می‌شود. در این روش به یک نگهبان نیاز داریم تا وقتی ورودی هر رشته پایان یافت، به آرایه علامت بدهد.

× مثال 10-8 یک آرایهٔ رشته‌ای پویا

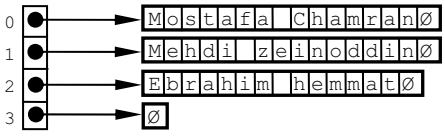
این برنامه نشان می‌دهد که چگونه می‌توان از کاراکتر '$' به عنوان کاراکتر نگهبان در تابع getline() استفاده کرد. مثال زیر تقریبا معادل مثال 9-8 است. برنامهٔ زیر مجموعه‌ای از اسامی را می‌خواند، طوری که هر اسم روی یک خط نوشته

می‌شود و هر اسم با کاراکتر '\n' پایان می‌یابد. این اسامی در آرایهٔ name ذخیره می‌شوند. سپس نام‌های ذخیره شده در آرایهٔ name چاپ می‌شوند:

```
int main()
{  char buffer[80];
   cin.getline(buffer,80,'$');
   char* name[4];
   name[0] = buffer;
   int count = 0;
   for (char* p=buffer; *p ! '\0'; p++)
      if (*p == '\n')
      {  *p = '\0';              // end name[count]
         name[++count] = p+1;    // begin next name
      }
   cout << "The names are:\n";
   for (int i=0; i<count; i++)
      cout << "\t" << i << . [" << name[i] << "]" << endl;
}
```

یک فضای ذخیره‌سازی به نام buffer با ظرفیت 80 کاراکتر به کار گرفته شده است. کل ورودی با استفاده از تابع getline() به درون این فضا خوانده می‌شود طوری که یا تعداد کل کاراکترها به 80 برسد و یا به کاراکتر '$' برخورد شود که در این صورت عمل دریافت کاراکترها خاتمه می‌یابد. سپس آرایهٔ name تعریف شده که این آرایه دارای چهار عنصر اشاره‌گر به char است. با استفاده از حلقهٔ for، آرایهٔ buffer پویش می‌شود. اشاره‌گر p این پویش را انجام می‌دهد. هر دفعه که p به کاراکتر '\n' برسد، به جای آن کاراکتر '\0' را می‌گذارد، سپس شمارندهٔ count را افزایش داده و آدرس کاراکتر بعدی یعنی p+1 را در name[count] ذخیره می‌نماید. به این ترتیب name[count] به نام بعدی اشاره دارد. دومین حلقهٔ for وظیفهٔ چاپ آرایهٔ رشته‌ای name را دارد.

هنگام وارد کردن اسامی، کاربر می‌تواند به هر تعداد که خواست کلید Enter را فشار دهد. پس از هر بار فشردن این کلید، همهٔ کاراکترهای روی آن خط به همراه یک کاراکتر '\n' به cin فرستاده می‌شود. همین که تعداد کاراکترها به 80 رسید یا کاراکتر نگهبان '$' تشخیص داده شد،



جریان ورودی قطع می‌شود و ادامهٔ برنامه دنبال می‌شود. تصویر مقابل نشان می‌دهد که اگر زنجیرهٔ

```
"Mostafa chamran\nMehdi Zeinoddin\n Ebrahim Hemmat\n"
```

درون buffer قرار گرفته باشد، آنگاه آرایهٔ رشته‌ای name چگونه مقداردهی خواهد شد. می‌بینید که بایت‌های اضافی که در مثال 9-8 وجود داشتند ، این جا وجود ندارند.

ممکن است رشته‌های کاراکتری مورد استفاده، در زمان کامپایل مشخص باشند. در این موارد به کارگیری و مدیریت یک آرایهٔ ایستا آسان‌تر است(مثل مثال 9-8). مثال بعدی نشان می‌دهد که مقداردهی به یک آرایهٔ رشته‌ای ایستا چقدر آسان‌تر است.

× مثال 8-11 مقداردهی یک آرایهٔ رشته‌ای

این برنامه هم آرایهٔ رشته‌ای name را مقداردهی کرده و سپس مقادیر آن را چاپ می‌نماید:

```
int main()
{  char* name[]
      = { "Mostafa Chamran", "Mehdi Zeinoddin", "Ebrahim Hemmat" };
      cout << "The names are:\n";
      for (int i = 0; i < 3; i++)
         cout << "\t" << i << ". [" << name[i] << "]"
              << endl;
}
```

```
The names are:
        0. [Mostafa Chamran]
        1. [Mehdi Zeinoddin]
        2. [Ebrahim Hemmat]
```

8-8 توابع استاندارد رشته‌های کاراکتری

سرفایل <cstring> که به آن «کتابخانهٔ رشته‌های کاراکتری» هم می‌گویند، شامل خانوادهٔ توابعی است که برای دست‌کاری رشته‌های کاراکتری خیلی مفیدند. مثال

بعدی ساده‌ترین آن‌ها یعنی تابع طول رشته را نشان می‌دهد. این تابع، طول یک رشتهٔ کاراکتری ارسال شده به آن (یعنی تعداد کاراکترهای آن رشته) را برمی‌گرداند.

× مثال 12-8 تابع strlen()

برنامهٔ زیر یک برنامهٔ آزمون ساده برای تابع strlen() است. وقتی strlen(s) فراخوانی می‌شود، تعداد کاراکترهای درون رشتهٔ s که قبل از کاراکتر NUL قرار گرفته‌اند، بازگشت داده می‌شود:

```
#include <cstring>
int main()
{  char s[] = "ABCDEFG";
   cout << "strlen(" << s << ") = " << strlen(s) << endl;
   cout << "strlen(\"\") = " << strlen("") << endl;
   char buffer[80];
   cout << "Enter string: ";   cin >> buffer;
   cout << "strlen(" << buffer << ") = " << strlen(buffer)
        << endl;
}
```

در مثال بعدی سه تابع دیگر را بررسی می‌کنیم. این توابع برای «مکان‌یابی» یک کاراکتر یا زیررشته در یک رشتهٔ کاراکتری مفروض استفاده می‌شوند.

× مثال 13-8 توابع strrchr(), strchr(), strstr()

برنامهٔ زیر، مکان‌یابی یک کاراکتر یا زیررشتهٔ خاص را در رشتهٔ کاراکتری s نشان می‌دهد:

```
#include <cstring>
int main()
{  char s[] = "The Mississippi is a long river.";
   cout << "s = \"" << s << "\"\n";
   char* p = strchr(s, ' ');
   cout << "strchr(s, ' ') points to s[" << p - s << "].\n";
```

```
    p = strchr(s, 'e');
    cout << "strchr(s, 'e') points to s[" << p - s << "].\n";
    p = strrchr(s, 'e');
    cout << "strrchr(s, 'e') points to s[" << p - s << "].\n";
    p = strstr(s, "is");
    cout << "strstr(s, \"is\") points to s[" << p - s
         << "].\n";
    p = strstr(s, "isi");
    cout << "strstr(s, \"is\") points to s[" << p - s
         << "].\n";
    if (p == NULL) cout << "strstr(s, \"isi\") returns
                           NULL\n";
}
```

```
s = "The SOFTWARE MOVEMENT is began."
strchr(s, ' ') points to s[3].
strchr(s, 'e') points to s[2].
strrchr(s, 'e') points to s[26].
strstr(s, "is") points to s[22].
strstr(s, "isi") returns NULL
```

وقتی strchr(s,' ') فراخوانی می‌شود، در رشتهٔ s کاراکتر فضای خالی ' ' جستجو می‌شود و در اولین خانه‌ای که یافت شد، یک اشاره‌گر به آن خانه بازگشت داده می‌شود. عبارت s-p ایندکس اولین کاراکتر یافته شده در رشتهٔ s را محاسبه می‌کند (به خاطر داشته باشید که آرایه‌ها با ایندکس 0 شروع می‌شوند). اولین فضای خالی در رشتهٔ s در ایندکس 3 قرار گرفته. به همین ترتیب وقتی strchr(s,'e') فراخوانی می‌شود، اشاره‌گری به اولین کاراکتر 'e' در رشتهٔ s بازگشت داده می‌شود. سپس عبارت s-p ایندکس آن خانه را محاسبه می‌نماید. اولین 'e' در ایندکس 2 قرار گرفته.

فراخوانی strrchr(s, 'e') یک اشاره‌گر به آخرین محل وقوع کاراکتر 'e' در رشتهٔ s را برمی‌گرداند. آخرین کاراکتر 'e' در رشتهٔ s در ایندکس 26 قرار گرفته است.

فراخوانی strstr(s, "is") باعث می‌شود که زیررشتهٔ "is" در رشتهٔ کاراکتری s جستجو شود و در اولین مکانی که یافت شد، اشاره‌گری به آن

مکان بازگشت داده می‌شود. زیر رشتهٔ مذکور در s[22] واقع شده. فراخوانی strstr(s, "isi") اشاره‌گر NULL را برمی‌گرداند زیرا عبارت "isi" در رشتهٔ کاراکتری s وجود ندارد.

رشته‌های کاراکتری را نمی‌توانیم با استفاده از عملگر جایگزینی (=) درون یکدیگر کپی کنیم (چرا؟) اما دو تابع وجود دارد که عمل جایگزینی را شبیه‌سازی می‌نمایند. تابع strcpy(s1, s2) باعث می‌شود که رشتهٔ کاراکتری s2 درون رشتهٔ کاراکتری s1 کپی شود. همچنین تابع strncpy(s1, s2, n) باعث می‌شود که n کاراکتر اول از رشتهٔ s2 روی n کاراکتر اول رشتهٔ s1 کپی شود. هر دو تابع فوق s1 را برمی‌گردانند و s2 را بدون تغییر می‌گذارند. مثال‌های بعدی طریقهٔ استفاده از این دو تابع را نشان می‌دهند.

× مثال 8-14 تابع strcpy()

برنامهٔ زیر نشان می‌دهد که فراخوانی strcpy(s1, s2) چه تاثیری دارد:

```
#include <iostream>
#include <cstring>
int main()
{  char s1[] = "ABCDEFG";
   char s2[] = "XYZ";
   cout << "Before strcpy(s1,s2):\n";
   cout << "\ts1 = [" << s1 << "], length = " << strlen(s1)
        << endl;
   cout << "\ts2 = [" << s2 << "], length = " << strlen(s2)
        << endl;
   strcpy(s1,s2);
   cout << "After strcpy(s1,s2):\n";
   cout << "\ts1 = [" << s1 << "], length = " << strlen(s1)
        << endl;
   cout << "\ts2 = [" << s2 << "], length = " << strlen(s2)
        << endl;
}
```

```
Before strcpy(s1,s2):
      s1 = [ABCDEFG], length = 7
```

```
      s2 = [XYZ], length = 3
After strcpy(s1,s2):
      s1 = [XYZ], length = 3
      s2 = [XYZ], length = 3
```

وقتی s2 به داخل s1 کپی شد، این دو دیگر فرقی با هم نمی‌کنند. هر دو شامل سه کاراکتر XYZ هستند. خروجی نشان می‌دهد که فراخوانی strcpy(s1, s2) چه تاثیری دارد. چون s2 دارای طول 3 است، فراخوانی strcpy(s1, s2) باعث می‌شود که چهار کاراکتر s2 (کاراکتر NUL هم جزئی از s2 است) روی چهار کاراکتر اول s1 رونویسی شود. حالا s1 و s2 دارای طول 3 هستند. کاراکترهای اضافی s1 بدون استفاده رها می‌شوند. درست است که s1 و S2 هم مقدار هستند ولی جدا از هم می‌باشند و اگر در ادامه یکی از آن دو تغییر کند، تاثیری بر دیگری نخواهد داشت.

× مثال 15-8 تابع strncpy()

برنامهٔ زیر بررسی می‌کند که فراخوانی strncpy(s1, s2, n) چه اثری دارد:

```
int main()
{  char s1[] = "ABCDEFG";
   char s2[] = "XYZ";
   cout << "Before strncpy(s1,s2,2):\n";
   cout << "\ts1 = [" << s1 << "], length = " << strlen(s1)
        << endl;
   cout << "\ts2 = [" << s2 << "], length = " << strlen(s2)
        << endl;
   strncpy(s1,s2,2);
   cout << "After strncpy(s1,s2,2):\n";
   cout << "\ts1 = [" << s1 << "], length = " << strlen(s1)
        << endl;
   cout << "\ts2 = [" << s2 << "], length = " << strlen(s2)
        << endl;
}
```

```
Before strncpy(s1,s2,2):
      s1 = [ABCDEFG], length = 7
      s2 = [XYZ], length = 3
After strncpy(s1,s2,2):
      s1 = [XYCDEFG], length = 7
      s2 = [XYZ], length = 3
```

فراخوانی strncpy(s1,s2,2) باعث می‌شود که دو کاراکتر اول رشتهٔ s2 روی دو کاراکتر اول رشتهٔ s1 کپی شود. طول s2 تاثیری بر طول s1 ندارد و اندازهٔ s1 تغییر نمی‌کند.

در فراخوانی strncpy(s1,s2,n) اگر strlen(s1)>n باشد، آنگاه n کاراکتر اول s2 روی n کاراکتر اول s1 کپی می‌شود و اگر strlen(s1)<=n باشد، آنگاه تاثیر این تابع با تابع strcpy() یکی خواهد بود.

دو تابع strcat() و strncat() همانند توابع strcpy() و strncpy() رفتار می‌کنند با این تفاوت که این توابع، کاراکترهای رشتهٔ s2 را به انتهای رشتهٔ s1 الصاق می‌کنند. عبارت "cat" از کلمهٔ "catenate" به معنای «الصاق نمودن» گرفته شده. البته دقت کنید که توابع مذکور s1 و s2 را به یک رشتهٔ واحد تبدیل نمی‌کنند بلکه یک کپی از کاراکترهای s2 را به انتهای s1 پیوند می‌زنند.

× مثال 8-16 تابع الصاق رشته strcat()

برنامهٔ زیر بررسی می‌کند که فراخوانی strcat(s1, s2) چه تاثیری دارد:

```
int main()
{  char s1[] = "ABCDEFG";
   char s2[] = "XYZ";
   cout << "Before strcat(s1,s2):\n";
   cout << "\ts1 = [" << s1 << "], length = " << strlen(s1)
        << endl;
   cout << "\ts2 = [" << s2 << "], length = " << strlen(s2)
        << endl;
   strcat(s1,s2);
   cout << "After strcat(s1,s2):\n";
   cout << "\ts1 = [" << s1 << "], length = " << strlen(s1)
        << endl;
   cout << "\ts2 = [" << s2 << "], length = " << strlen(s2)
        << endl;
}
```

```
Before strcat(s1,s2):
      s1 = [ABCDEFG], length = 7
      s2 = [XYZ], length = 3
```

```
After strcat(s1,s2):
      s1 = [ABCDEFGXYZ], length = 10
      s2 = [XYZ], length = 3
```

تصویر خروجی نشان می‌دهد که فراخوانی strcat(s1,s2) چگونه s1 را تحت تاثیر قرار می‌دهد. چون s2 طول 3 دارد، فراخوانی strcat(s1, s2) باعث می‌شود که سه بایت به انتهای s1 اضافه گردد و سپس کاراکترهای s2 به درون آن‌ها کپی شوند و در نهایت کاراکتر NUL به آن‌ها اضافه می‌شود تا s1 تکمیل شود. توجه داشته باشید که باز هم s1 و s2 مستقل از هم باقی می‌مانند.

× مثال 17-8 تابع الصاق رشته stancat()

برنامهٔ زیر تاثیر فراخوانی strncat(s1,s2,n) را نشان می‌دهد:

```
#include <iostream.h>
#include <cstring.h>
int main()
{  // test-driver for the strncat() function:
   char s1[] = "ABCDEFG";
   char s2[] = "XYZ";
   cout << "Before strncat(s1,s2,2):\n";
   cout << "\ts1 = [" << s1 << "], length = " << strlen(s1)
        << endl;
   cout << "\ts2 = [" << s2 << "], length = " << strlen(s2)
        << endl;
   strncat(s1,s2,2);
   cout << "After strncat(s1,s2,2):\n";
   cout << "\ts1 = [" << s1 << "], length = " << strlen(s1)
        << endl;
   cout << "\ts2 = [" << s2 << "], length = " << strlen(s2)
        << endl;
}
```

```
Before strncat(s1,s2,2):
      s1 = [ABCDEFG], length = 7
      s2 = [XYZ], length = 3
After strncat(s1,s2,2):
      s1 = [ABCDEFGXY], length = 9
      s2 = [XYZ], length = 3
```

فراخوانی strncat(s1,s2,2) دو کاراکتر ابتدای s2 را به انتهای s1 اضافه می‌کند و سپس کاراکتر NUL را هم به انتهای s1 می‌افزاید.

هر دو تابع strcat() و strncat() بایت‌هایی را به انتهای رشتهٔ s1 اضافه می‌کنند. اگر این بایت‌ها در حافظه قبلا اشغال شده باشند و رشتهٔ s1 نتواند در مکان فعلی گسترش یابد، آنگاه کل این رشته به بخشی از حافظه منتقل می‌شود که بایت‌های مورد نیاز در دسترس باشند و رشتهٔ مورد نظر بتواند آزادانه گسترش یابد.

جدول زیر برخی از توابع مفیدی که در سرفایل <cstring> تعریف شده‌اند را نشان می‌دهد. عبارت size_t که در توابع جدول استفاده شده یک نوع عدد صحیح ویژه است که در سرفایل <csting> تعریف شده است:

توابع تعریف شده در سرفایل <cstring>

نام تابع	شرح
memcpy()	void* memcpy(void* s1, const void* s2, size_t n); n بایت اول S1* را با n بایت اول S2* جایگزین می‌کند. s را بر می‌گرداند.
strcat()	char* strcat(char* s1, const char* s2); s2 را به s1 الحاق می‌کند. s1 را بر می گرداند.
strchr()	char* strchr(const char* s, int c); یک اشاره‌گر به اولین وقوع c در رشتهٔ s بر می‌گرداند. اگر c در s نباشد، NULL را برمی‌گرداند.
strcmp()	int strcmp(const char* s1, const char* s2); s1 را با زیررشتهٔ s2 مقایسه می‌کند. اگر s1 به ترتیب الفبایی، کوچک‌تر یا مساوی یا بزرگ‌تر از s2 باشد، مقدار منفی یا صفر یا مقدار مثبت را بر می‌گرداند.
strcpy()	char* strcpy(char* s1, const char* s2); s2 را در s1 کپی می‌کند. s1 را بر می‌گرداند.
strcspn()	size_t strcspn(char* s1, const char* s2); طول بزرگ‌ترین زیررشته ای از s1 را بر می‌گرداند که با s[0] شروع شده و شامل هیچ‌یک از

	کاراکترهای موجود در s2 نیست.
strlen()	size_t strlen(const char* s); طول s را بر می‌گرداند که تعداد کاراکترهایی است که با s[0] شروع می‌شود و با اولین کاراکتر NUL خاتمه می‌یابد.
strncat()	char* strncat(char* s1, const char* s2, size_t n); n کاراکتر اول s2 را به s1 الصاق می‌کند. اگر n>=strlen(s2) باشد، آنگاه strncat(s1,s2,n) تاثیری مشابه strcat(s1,s2) دارد.
strncmp()	int strncmp(const char* s1, const char* s2, size_t n); n کاراکتر اول s1 را با n کاراکتر اول s2 مقایسه می‌کند و اگر زیررشتهٔ اول به ترتیب الفبایی از زیررشتهٔ دوم بزرگ‌تر، مساوی یا کوچک‌تر باشد، مقدار مثبت، صفر یا مقدار منفی را بر می‌گرداند. اگر n>=strlen(s2) باشد، آنگاه strncmp(s1, s2, n) تاثیری مشابه با strcmp(s1, s2) خواهد داشت.
strncpy()	char* strncpy(char* s1, const char* s2, size_t n); n کاراکتر اول s1 را با n کاراکتر اول s2 جایگزین می‌کند و s1 را بر می‌گرداند. اگر n<=strlen(s1) باشد، طول s1 تغییر نمی‌کند. اگر n>=strlen(s2) باشد آنگاه strncpy(s1, s2, n) تاثیری مشابه strcpy(s1, s2) خواهد داشت.
strpbrk()	char* strpbrk(const char* s1, const char* s2); محل اولین رخداد هر یک از کاراکترهای s2 را در s1 بر می‌گرداند. اگر هیچ‌یک از کاراکترهای s2 در s1 یافت نشد، NULL را بر می‌گرداند.
strrchr()	char* strrchr(const char* s, int c); آخرین محل قرار گرفتن کاراکتر c در رشتهٔ s را بر می‌گرداند. اگر c در s نباشد، NULL را برمی‌گرداند.
strspn()	size_t strspn(char* s1, const char* s2); طول بزرگ‌ترین زیررشته ای از s1 را بر می‌گرداند که از s[0] شروع شده و فقط شامل کاراکترهای موجود در s2 است.

strstr()	char* strstr(const char* s1, const char* s2); آدرس اولین محل وقوع زیررشتهٔ s2 در رشتهٔ s1 را بر می‌گرداند. اگر s2 در s1 نباشد NULL را برمی‌گرداند.
strtok()	char* strtok(char* s1, char* s2); رشتهٔ s1 را با استفاده از کاراکترهای موجود در رشتهٔ s2 نشانه‌گذاری می‌کند. پس از فراخوانی آغازین strtok(s1, s2) هر فراخوانی موفقیت‌آمیز strtok(NULL,s2) اشاره‌گری به نشانهٔ یافت‌شدهٔ بعدی در s1 بر می‌گرداند. این فراخوانی ها رشتهٔ s1 را تغییر می‌دهد و هر کاراکتر نشانه را با کاراکتر NUL جایگزین می‌کند.

8-9 رشته‌های کاراکتری در C++ استاندارد

رشته‌های کاراکتری که تاکنون تشریح شد، در زبان C استفاده می‌شوند و البته بخش مهمی از C++ نیز محسوب می‌شوند زیرا وسیلهٔ مفیدی برای پردازش سریع داده‌ها هستند. اما این سرعت پردازش، هزینه‌ای هم دارد: خطر خطاهای زمان اجرا. این خطاها معمولا از این ناشی می‌شوند که فقط بر کاراکتر NUL به عنوان پایان رشته تکیه می‌شود. C++ رشته‌های کاراکتری خاصی نیز دارد که امن‌تر و مطمئن‌تر هستند. در این رشته‌ها، طول رشته نیز درون رشته ذخیره می‌شود و لذا فقط به کاراکتر NUL برای مشخص نمودن انتهای رشته اکتفا نمی‌شود.

8-10 نگاهی دقیق‌تر به تبادل داده‌ها

وقتی می‌خواهیم داده‌هایی را وارد کنیم، این داده‌ها را در قالب مجموعه‌ای از کاراکترها تایپ می‌کنیم. همچنین وقتی می‌خواهیم نتایجی را به خارج از برنامه

بفرستیم، این نتایج در قالب مجموعه‌ای از کاراکترها نمایش داده می‌شوند. لازم است که این کاراکترها به نحوی برای برنامه تفسیر شوند. مثلا وقتی قصد داریم یک عدد صحیح را وارد کنیم، چند کاراکتر عددی تایپ می‌کنیم. حالا ساز و کاری لازم است که از این کاراکترها یک مقدار صحیح بسازد و به برنامه تحویل دهد. همچنین وقتی قصد داریم یک عدد اعشاری را به خروجی بفرستیم، باید با استفاده از راه‌کاری، آن عدد اعشاری به کاراکترهایی تبدیل شود تا در خروجی نمایش یابد. ***جریان‌ها***[1] این وظایف را در C++ بر عهده دارند. جریان‌ها شبیه پالایه‌ای هستند که داده‌ها را به کاراکتر تبدیل می‌کنند و کاراکترها را به داده‌هایی از یک نوع بنیادی تبدیل می‌نمایند. به طور کلی، ورودی‌ها و خروجی‌ها را یک کلاس جریان به نام stream کنترل می‌کند. این کلاس خود به زیرکلاس‌هایی تقسیم می‌شود: شیء istream جریانی است که داده‌های مورد نیاز را از کاراکترهای وارد شده از صفحه کلید، فراهم می‌کند. شیء ostream جریانی است که داده‌های حاصل را به کاراکترهای خروجی قابل نمایش روی صفحهٔ نمایش‌گر تبدیل می‌نماید. شیء ifstream جریانی است که داده‌های مورد نیاز را از داده‌های داخل یک فایل، فراهم می‌کند. شیء ofstream جریانی است که داده‌های حاصل را درون یک فایل ذخیره می‌نماید. این جریان‌ها و طریقهٔ استفاده از آن‌ها را در ادامه خواهیم دید.

کلاس istream نحوهٔ رفتار با کاراکترهای ورودی را تعریف می‌نماید. مهم‌ترین چیزی که باید تعریف شود و اتفاقا بیشترین استفاده را هم دارد، نحوهٔ رفتار «عملگر برون کشی << » است (به آن عملگر ورودی نیز می‌گوییم). قبلا در برنامه‌ها از آن استفاده کرده‌ایم. این عملگر دو عملوند دارد: شیء istream که مشخص می‌کند کاراکترها از کجا باید بیرون کشیده شوند، و شیئی که مشخص می‌کند مقدار برون‌کشی‌شده باید از چه نوعی باشد و کجا باید ذخیره شود. به این پردازش که از کاراکترهای خام ورودی مقادیری با نوع مشخص تولید می‌کند، ***قالب‌بندی***[2] می‌گویند.

× مثال 18-8 عملگر برون کشی << ورودی را قالب‌بندی می‌کند

فرض کنید کد زیر اجرا شده و وردی به شکل "46 " وارد شده است:

```
int n;
cin >> n;
```

ورودی بالا در حقیقت شامل هفت کاراکتر است: ' ' و ' ' و ' ' و ' ' و '4' و '6' و '\n' یعنی چهار کاراکتر فضای خالی و سپس '4' و بعد از آن '6' و در

1 – Streams 2 - Formatting

نهایت کاراکتر پایان خط '\n' آمده است. این کاراکترها درون جریان ورودی قرار می‌گیرند. شیء جریان cin کاراکترها را یکی یکی پویش می‌کند. اگر اولین کاراکتری که به آن وارد می‌شود، کاراکتر فضای خالی یا هر کاراکتر فضای سفید دیگر (مثل tab یا خط جدید) باشد، آن را نادیده گرفته و از جریان ورودی حذف می‌کند. این کار همچنان ادامه می‌یابد تا این که به یک کارکتر غیرفاصله‌ای برخورد کند. چون دومین عملوند در عملگر برون کشی cin >> n از نوع int است، پس شیء cin به دنبال اعدادی می‌گردد تا بتواند مقداری از این نوع حاصل نماید. بنابراین تلاش می‌کند که پس از دور انداختن همهٔ فاصله‌ها، یکی از دوازده کاراکتر '+' یا '-' یا '0' یا '1' یا '2' یا '3' یا '4' یا '5' یا '6' یا '7' یا '8' یا '9' را بیابد. اگر به غیر از این کاراکترها به هر کدام از 244 کاراکتر دیگر برخورد کند، از کار می‌ایستد. در مثال بالا کاراکتر '4' پیدا شده. لذا شیء cin این کاراکتر را تفکیک کرده و کاراکتر بعدی را به امید یافتن عدد دیگر، پویش می‌کند. تا زمانی که کاراکترهای بعدی هم عدد باشند، cin آن‌ها را تفکیک می‌نماید و پویش را ادامه می‌دهد. به محض این که به یک کاراکتر غیرعددی برخورد شود، cin متوقف می‌شود و آن کاراکتر غیرعددی را همچنان در جریان ورودی نگه می‌دارد. در مثال بالا، شیء cin شش کاراکتر را از جریان ورودی برداشت می‌کند: چهار فضای خالی را حذف می‌کند و کاراکترهای '4' و '6' را هم با هم ترکیب می‌کند تا مقدار 46 بدست آید و سپس این مقدار را در n قرار می‌دهد. وقتی که برداشت به پایان رسید، هنوز کاراکتر خط جدید در جریان ورودی باقی مانده است. وقتی دستور cin بعدی اجرا شود، این کاراکتر اولین کاراکتری است که از جریان ورودی برداشت می‌شود و چون این کاراکتر از نوع کاراکترهای فضای سفید است، نادیده گرفته شده و حذف می‌شود.

از گفته‌های بالا به راحتی می‌توان استنتاج کرد که از عملگر برون کشی << نمی‌توانیم برای وارد کردن کاراکترهای فضای سفید استفاده کنیم. برای این کار باید از یک تابع ورودی که کاراکترها را قالب‌بندی نمی‌کند استفاده کنیم.

یک عبارت ورودی مثل cin >> x; به غیر از این که مقداری را برای x فراهم می‌کند، حاصل دیگری هم به شکل یک دادهٔ منطقی به دست می‌دهد. این حاصل، برحسب این که عمل برون‌کشی موفقیت‌آمیز بوده یا خیر، مقدار true یا false دارد. از این حاصل می‌توان به عنوان یک شرط منطقی در دستورات دیگر استفاده کرد. مثلا می‌توان حلقه‌ها را با استفاده از آن کنترل کرد.

× مثال 8-19 استفاده از عملگر بیرون‌کشی برای کنترل کردن یک حلقه

```
int main()
{  int n;
   while (cin >> n)
      cout << "n = " << n << endl;
}
```

```
46
n = 46
22        44        66        88
n = 22
n = 44
n = 66
n = 88
33, 55, 77, 99
n = 33
```

تا زمانی که اعداد ورودی فقط به وسیلهٔ کاراکترهای فضای سفید (مثل tab یا فاصلهٔ خالی یا کاراکتر خط جدید) از هم جدا شده باشند، حلقه ادامه می‌یابد. همین که به اولین کاراکتر غیرفاصله‌ای و غیرعددی برخورد شود (که در مثال بالا کاراکتر کاما ' , ' است)، عملگر برون‌کشی از کار می‌ایستد و به همین واسطه، حلقه نیز خاتمه می‌یابد.

8-11 ورودی قالب‌بندی نشده

سرفایل <iostream> توابع مختلفی برای ورودی دارد. این توابع برای وارد کردن کاراکترها و رشته‌های کاراکتری به کار می‌روند که کاراکترهای فضای سفید را

نادیده نمی‌گیرند. رایج‌ترین آن‌ها، تابع cin.get() برای دریافت یک کاراکتر تکی و تابع cin.getline() برای دریافت یک رشتهٔ کاراکتری است.

× مثال 8-20 دریافت کاراکترها با استفاده از تابع cin.get()

```
while (cin.get(c))
{  if (c >= 'a' && c <= 'z') c += 'A' - 'a';   // capitalize c
   cout.put(c);
   if (c = = '\n') break;
}
```

حلقهٔ بالا با استفاده از عبارت cin.get(c) کنترل می‌شود. اگر در جریان ورودی، کاراکتر پایان فایل تشخیص داده شود (کلیدهای Ctrl+Z)، تابع cin.get() متوقف شده و مقدار false را برمی‌گرداند و در نتیجه حلقه متوقف می‌شود. همچنین اگر کاراکتر وارد شده یک کارکتر خط جدید باشد، دستور break درون حلقه باعث می‌شود که حلقه متوقف شود. اولین دستور if کاراکتر وارد شده را به حرف بزرگ تبدیل می‌کند و سپس کاراکتر حاصل با تابع cout.put(c) به خروجی فرستاده می‌شود. شکل زیر نمونه‌ای از اجرای حلقهٔ بالاست:

```
I like C++ features!
I LIKE C++ FEATURES!
```

× مثال 8-21 وارد کردن یک رشتهٔ کاراکتری به وسیلهٔ تابع cin.getline()

برنامهٔ زیر نشان می‌دهد که چطور می‌توان داده‌های متنی را خط به خط از ورودی خوانده و درون یک آرایهٔ رشته‌ای قرار داد:

```
const int LEN=32;         // maximum word length
const int SIZE=10;        // array size
typedef char Name[LEN];   // defines Name to be a C_string type
int main()
{  Name martyr[SIZE]; // defines martyr to be an array of 10 names
   int n=0;
   while(cin.getline(martyr[n++], LEN) && n<SIZE)
      ;
   --n;
   for (int i=0; i<n; i++)
```

```
        cout << '\t' << i+1 << ". " << martyr[i] << endl;
}
```

شیء martyr آرایه‌ای از 10 شی با نوع Name است. قبلا با استفاده از typedef مشخص شده که نوع Name معادل یک آرایهٔ 32 عنصری از نوع char است (یعنی رشته‌ای به طول 32 کاراکتر +NUL). فراخوانی تابع cin.getline(martyr[n++], LEN) همهٔ کاراکترها را از جریان ورودی می‌خواند تا زمانی که تعداد کاراکترهای خوانده شده به LEN-1 برسد یا این که به کاراکتر پایان خط برخورد شود. سپس این کاراکترهای خوانده شده به درون رشتهٔ کاراکتری martyr[n] کپی می‌شوند. هرگاه به کاراکتر خط جدید برخورد شود، این کاراکتر از جریان ورودی حذف می‌شود. در نتیجه هیچ وقت کاراکتر خط جدید به درون رشتهٔ کاراکتری فرستاده نمی‌شود.

توجه داشته باشید که بدنهٔ حلقهٔ while خالی است. این حلقه پایان می‌یابد وقتی که در جریان ورودی به کاراکتر پایان فایل برخورد شود یا این که n==size شود. چون n از صفر شروع شده و پس از هر بار دریافت نام، یک واحد افزوده می‌شود، همیشه مقدار n یکی بیشتر از تعداد اسامی خوانده شده است. به همین دلیل پس از اتمام حلقه، مقدار n باید یک واحد کاسته شود تا تعداد واقعی اسامی خوانده شده را نشان دهد. به راحتی می‌توان با استفاده از یک حلقهٔ for مقدار رشته‌ها را چاپ کرد یا روی آن‌ها پردازش‌های دیگری انجام داد. اگر فرض کنیم که اسامی از فایل متنی مقابل خوانده شده باشند، آنگاه خروجی کد بالا به شکل زیر خواهد بود:

martyr.dat

```
Thamen-al-aemmeh (1360/7/5) - ABADAN
Tarigh-al-ghods (1360/9/8) - BOSTAN
Fath-al-mobin (1361/1/1) - DEZFUL
Beyt-al-moghaddas(1361/2/10) - KHORAMSHAHR
Ramazan (1361/4/23) - TACTICAL
Val-fajr 6 (1362/12/2) - CHAZABEH
Val-fajr 8 (1364/11/20) - MAJNOUN
Karbala 1 (1365/4/10) - MEHRAN
```

```
1. Thamen-al-aemmeh (1360/7/5) - ABADAN
2. Tarigh-al-ghods (1360/9/8) - BOSTAN
3. Fath-al-mobin (1361/1/1) - DEZFUL
4. Beyt-al-moghaddas(1361/2/10) - KHORAMSHAHR
5. Ramazan (1361/4/23) - TACTICAL
```

```
6. Val-fajr 6 (1362/12/2) - CHAZABEH
7. Val-fajr 8 (1364/11/20) - MAJNOUN
8. Karbala 1 (1365/4/10) - MEHRAN
```

8-12 نوع string در C++ استاندارد

در C++ استاندارد نوع داده‌ای خاصی به نام string وجود دارد که مشخصات این نوع در سرفایل <string> تعریف شده است. برای آشنایی با این نوع جدید، از طریقهٔ اعلان و مقداردهی آن شروع می‌کنیم. اشیایی که از نوع string هستند به چند طریق می‌توانند اعلان و مقداردهی شوند:

```
string s1;                              // s1 contains 0 characters
string s2 = "PNU University";          // s2 contains 14 characters
string s3(60, '*');                     // s3 contains 60 asterisks
string s4 = s3;                         // s4 contains 60 asterisks
string s5(s2, 4, 2);          // s5 is the 2-character string "Un"
```

stringها مثل رشته‌های کاراکتری برای ذخیره کردن مجموعه‌ای از کاراکترها به کار می‌روند. s1 در کد بالا از نوع string اعلان شده. اگر یک شی از نوع string اعلان شده ولی مقداردهی نشده باشد (مثل s1)، آنگاه درون آن یک رشتهٔ خالی صفر کاراکتری خواهد بود. stringها را می‌توان مثل رشته‌های کاراکتری به طور مستقیم مقداردهی کرد، مثل s2. اشیای string را می‌توانیم طوری مقداردهی کنیم که با تعداد مشخصی از یک کاراکتر دلخواه پر شود. مثل s3 که حاوی شصت کاراکتر '*' است. stringها را بر خلاف رشته‌های کاراکتری می‌توانیم به یکدیگر تخصیص دهیم و آن‌ها را از روی یک شیء string موجود مقداردهی کنیم. مثل s4 که همهٔ مقادیر s3 درون آن قرار خواهد گرفت. همچنین می‌توانیم اشیای string را با زیررشته‌ای از یک string موجود مقداردهی نماییم. مثل s5 که دو کاراکتر از s2 با شروع از ایندکس چهارم درون آن قرار خواهد گرفت. توجه کنید که سازندهٔ زیررشته سه قسمت دارد: 1 – رشتهٔ والد که زیررشته از درون آن استخراج می‌شود (در این جا s2 است) 2 – کاراکتر آغازین زیررشته (در این جا s2[4] است) 3 – طول زیررشته (در این جا 2 است).

ورودی قالب‌بندی شده با stringها مثل رشته‌های کاراکتری معمولی رفتار می‌کند. یعنی هنگام وارد کردن کاراکترهای دریافتی، کاراکترهای فضای سفید را نادیده گرفته و حذف می‌کند و همین که بعد از کلمهٔ جاری به یک کاراکتر فضای سفید برسد، دریافت کاراکترها را خاتمه می‌دهد. stringها تابعی به نام getline() مخصوص به خودشان دارند که بسیار شبیه تابع cin.getline() رفتار می‌کند:

```
string s = "ABCDEFG";
getline(cin, s);      // reads the entireline of characters into s
```

همچنین درون stringها می‌توانیم از عملگر زیرنویس مثل رشته‌های کاراکتری استفاده کنیم:

```
char c = s[2];          // assigns 'C' to c
s[4] = '*';             // changes s to "ABCD*FG"
```

دقت کنید که این جا هم ایندکس از صفر شروع می‌شود.

اشیای نوع string را می‌توانیم با استفاده از تابع زیر به نوع رشتهٔ کاراکتری تبدیل کنیم:

```
const char* cs = s.c_str();   // converts s into the C-string cs
```

کلاس string در C++ استاندارد، تابعی به نام length() دارد که با استفاده از آن می‌توانیم تعداد کاراکترهای موجود در یک شیء string را بیابیم. این تابع را می‌توانیم به شکل زیر به کار ببریم:

```
cout << s.length() << endl;
// prints 7 for the string s = = "ABCD*FG"
```

در C++ می‌توانیم stringها را با استفاده از عملگرهای رابطه‌ای با هم مقایسه کنیم. درست شبیه انواع بنیادی دیگر:

```
if (s2 < s5) cout << "s2 lexicographically precedes s5\n";
while (s4 = = s3)    // ...
```

به راحتی می‌توانیم با استفاده از عملگرهای + و =+ محتویات stringها را به یکدیگر پیوند بزنیم یا با هم ترکیب کنیم:

```
string s6 = s + "HIJK";     // changes s6 to "ABCD*FGHIJK"
s2 += s5;                   // changes s2 to "PNU UniversityUn"
```

می‌توانیم با استفاده از تابع substr() یک زیررشته را از درون یک string استخراج کنیم:

```
s4 = s6.substr(5,3);        // changes s4 to "FGH"
```

توابع erase() و replace() بخشی از محتویات درون یک string را حذف کرده یا رونویسی می‌کنند:

```
s6.erase(4, 2);             // changes s6 to "ABCDGHIJK"
s6.replace(5, 2, "xyz");    // changes s6 to "ABCDGxyzJK"
```

تابع erase() دو پارامتر دارد که پارامتر اول، نقطه شروع حذف را نشان می‌دهد و پارامتر دوم، تعداد کاراکترهایی که باید حذف شوند. تابع replace() سه پارامتر دارد: پارامتراول، نقطه شروع رونویسی را نشان می‌دهد، پارامتر دوم تعداد کاراکترهایی که باید حذف شوند و پارامتر سوم زیررشته ای است که باید به جای کاراکترهای حذف شده قرار بگیرد.

تابع find() ایندکس اولین وقوع یک زیررشته مفروض را در string فعلی نشان می‌دهد:

```
string s7 = "The SOFTWARE MOVEMENT bases";
cout << s7.find("EM") << endl;    // prints 17
cout << s7.find("EO") << endl;
// prints 27, the length of the string
```

اگر تابع find() زیررشتهٔ مورد نظر را پیدا نکند، طول رشتهٔ تحت جستجو را برمی‌گرداند.

می‌بینید که استفاده از نوع string بسیار آسان است. توابع کمکی که در بالا به اختصار معرفی شدند کاربرد stringها را سهل‌تر می‌نمایند. نکتهٔ قابل توجه، نحوهٔ فراخوانی این توابع است، در همهٔ این فراخوانی‌ها (به غیر از تابع getline()) ابتدا نام string مربوطه آمده و سپس یک نقطه . و بعد از آن تابع مورد نظر ذکر شده است. علت این است که نوع String از روی کلاس string ساخته می‌شود و در

استفاده از آن باید از قوانین کلاس‌ها پیروی کنیم. موضوع کلاس‌ها را در فصل‌های آتی به دقت بررسی می‌نماییم.

× مثال 22-8 استفاده از نوع string

کد زیر یک مجموعه کاراکتر را از ورودی می‌گیرد و سپس بعد از هر کاراکتر "E" یک علامت ویرگول ',' اضافه می‌نماید. مثلا اگر عبارت

```
"The SOFTWARE MOVEMENT is began"
```

وارد شود، برنامهٔ زیر، آن را به جملهٔ زیر تبدیل می‌کند:

```
The SOFTWARE, MOVE,ME,NT is began
```

متن برنامه این چنین است:

```
string word;
int k;
while (cin >> word)
{  k = word.find("E") + 1;
   if (k < word.length())
      word.relace(k, 0, ",");
   cout << word << ' ';
}
```

حلقهٔ while به وسیلهٔ ورودی کنترل می‌شود، وقتی پایان فایل شناسایی شود، پایان می‌پذیرد. در این حلقه هر دفعه یک کلمه خوانده میشود. اگر حرف E پیدا شود، یک ویرگول ',' بعد از آن درج می‌شود.

8-13 فایل‌ها

یکی از مزیت‌های رایانه، قدرت نگهداری اطلاعات حجیم است. ***فایل‌ها***[1] این قدرت را به رایانه می‌دهند. اگر چیزی به نام فایل وجود نمی‌داشت، شاید رایانه‌ها به شکل امروزی توسعه و کاربرد پیدا نمی‌کردند. چون اغلب برنامه‌های امروزی با فایل‌ها سر و کار دارند، یک برنامه‌نویس لازم است که با فایل آشنا باشد و بتواند با استفاده از این امکان ذخیره و بازیابی، کارایی برنامه‌هایش را ارتقا دهد. پردازش فایل در C++ بسیار شبیه تراکنش‌های معمولی ورودی و خروجی است زیرا این‌ها همه از اشیای

1 – Files

جریان مشابهی بهره می‌برند. جریان fstream برای تراکنش برنامه با فایل‌ها به کار می‌رود. fstream نیز به دو زیرشاخهٔ ifstream و ofstream تقسیم می‌شود. جریان ifstream برای خواندن اطلاعات از یک فایل به کار می‌رود و جریان ofstream برای نوشتن اطلاعات درون یک فایل استفاده می‌شود. فراموش نکنید که این جریان‌ها در سرفایل <fstream> تعریف شده‌اند. پس باید دستور پیش‌پردازندهٔ #include <fstream> را به ابتدای برنامه بیافزایید. سپس می‌توانید عناصری از نوع جریان فایل به شکل زیر تعریف کنید:

```
ifstream readfile("INPUT.TXT");
ofstream writefile("OUTPUT.TXT");
```

طبق کدهای فوق، readfile عنصری است که داده‌ها را از فایلی به نام INPUT.TXT می‌خواند و writefile نیز عنصری است که اطلاعاتی را در فایلی به نام OUTPUT.TXT می‌نویسد. اکنون می‌توان با استفاده از عملگر << داده‌ها را به درون readfile خواند و با عملگر >> اطلاعات را درون writefile نوشت. به مثال زیر توجه کنید.

× مثال 23-8 یک دفتر تلفن

برنامهٔ زیر، چند نام و تلفن مربوط به هر یک را به ترتیب از کاربر دریافت کرده و در فایلی به نام PHONE.TXT ذخیره می‌کند. کاربر برای پایان دادن به ورودی باید عدد 0 را تایپ کند.

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{  ofstream phonefile("PHONE.TXT");
   long number;
   string name;
   cout << "Enter a number for each name. (0 for quit): ";
   for ( ; ; )
   {  cout << "Number: ";
      cin >> number;
      if (number == 0) break;
```

```
        phonefile << number << ' ';
        cout << "Name: ";
        cin >> name;
        phonefile << name << ' ';
        cout << endl;
    }
}
```

در برنامهٔ بالا، جریانی به نام phonefile از نوع ofstream تعریف شده. این جریان، فایل PHONE.TXT را مدیریت می‌کند. چون جریان مذکور از نوع ofstream است، فقط می‌تواند اطلاعاتی درون فایل PHONE.TXT بنویسد. یک متغیر صحیح به نام number و یک متغیر رشته‌ای به نام name شماره تلفن و نام مربوطه را از ورودی می‌گیرند. همهٔ کارها درون حلقهٔ for انجام می‌شود. ساختار این حلقه به شکل یک حلقهٔ بی‌انتها است زیرا حلقه از درون کنترل می‌شود (با استفاده از دستور break). هنگامی که کاربر عدد صفر را به عنوان شماره تلفن وارد کند، حلقه فورا خاتمه می‌یابد. در غیر این صورت، شمارهٔ مذکور با استفاده از عملگر خروجی >> به جریان phonefile فرستاده می‌شود و این جریان، شماره را درون فایل می‌نویسد. سپس نام صاحب شماره پرسیده می‌شود و آن هم به همین ترتیب به جریان phonefile فرستاده شده و در نتیجه درون فایل نوشته می‌شود. دقت کنید که هر مقداری که درون فایل نوشته می‌شود، یک کاراکتر فضای خالی نیز پس از آن در فایل درج می‌شود. اگر این کار را نکنیم، همهٔ نام‌ها و شماره‌ها بدون فاصله و پیوسته نوشته می‌شوند و در این صورت خواندن فایل حاصل بسیار مشکل‌تر خواهد بود.

مثال بالا نشان می‌دهد که عملگر خروجی >> با جریان‌ها مثل cout رفتار می‌کند. عملگر ورودی << نیز همین طور است. یعنی با استفاده از عملگر ورودی می‌توان به سادگی اطلاعات درون یک فایل را خواند.

× مثال 24-8 جستجوی یک شماره در دفتر تلفن

این برنامه، فایل تولید شده توسط برنامهٔ قبل را به کار می‌گیرد و درون آن به دنبال یک شماره تلفن می‌گردد:

```
#include <fstream>
```

```
#include <iostream>
using namespace std;
int main()
{  ifstream phonefile("PHONE.TXT");
   long number;
   string name, searchname;
   bool found=false;
   cout << "Enter a name for findind it's phone number: ";
   cin >> searchname;
   cout << endl;
   while (phonefile >> number)
   {  phonefile >> name;
      if (searchname == name)
      {  cout << name << ' ' << number << endl;
         found = true;
      }
   if (!found) cout << searchname
                    << " is not in this phonebook." << endl;
}
```

در برنامهٔ بالا هم جریان phonefile برای مدیریت فایل PHONE.TXT منظور شده است ولی این بار جریان مذکور از نوع istream است. یعنی فقط می‌تواند فایل را بخواند. عمل خواندن اطلاعات فایل، درون حلقهٔ while صورت می‌گیرد. بخش کنترلی این حلقه به شکل زیر نوشته شده:

```
   while (phonefile >> number)
```

این کد دو وظیفه را انجام می‌دهد: اول این که یک عدد را از درون جریان phonefile می‌خواند و آن را درون متغیر number قرار می‌دهد و دوم این که اگر عمل خواندن موفقیت‌آمیز بود، حلقه ادامه می‌یابد و در غیر این صورت حلقه خاتمه می‌پذیرد. مزیت این شکل خواندن در آن است که دیگر لازم نیست نگران باشیم که آیا به پایان فایل رسیده‌ایم یا نه. هنگامی که به انتهای فایل برسیم، عبارت phonefile>>number به عنوان نادرست تفسیر می‌شود و به همین دلیل حلقه خاتمه می‌یابد. اگر عمل خواندن به این طریق بررسی نمی‌شد، مجبور می‌شدیم در جای

دیگر و به شکل دیگری مراقبت کنیم که به انتهای فایل رسیده‌ایم یا خیر. اگر این مراقبت صورت نگیرد و وقتی به انتهای فایل مورد نظر رسیدیم دوباره تلاش کنیم که داده‌های بیشتری را از فایل بخوانیم، با خطا مواجه می‌شویم و برنامه به شکل ناخواسته پایان می‌گیرد. توجه داشته باشید که در مثال قبلی، هنگام نوشتن نام‌ها و شماره‌ها، یک فاصلهٔ خالی ' ' نیز بین آن ها درج کردیم اما در این مثال هنگام خواندن اطلاعات، توجهی به این فاصله‌های اضافی نکردیم. به این دلیل که برای خواندن اطلاعات از عملگر << استفاده کرده‌ایم و همان طور که در ابتدای فصل گفتیم، این عملگر کاراکترهای فضای سفید را نادیده می‌گیرد.

برای این که اطلاعاتی را از درون فایل به صورت کاراکتر به کاراکتر بخوانیم، می‌توانیم از تابع get() استفاده کنیم. شکل استفاده از این تابع را قبلا در جریان cin دیدیم. همچنین برای این که اطلاعات را به شکل کاراکتر به کاراکتر درون یک فایل بنویسیم می‌توانیم از تابع put() استفاده کنیم. این تابع را نیز قبلا در جریان cout به کار برده‌ایم. می‌بینید که خواندن و نوشتن فایل‌ها بسیار شبیه خواندن و نوشتن ورودی/خروجی معمولی است.

اما دنیای فایل‌ها به همین جا ختم نمی‌شود. فایل‌هایی که در مثال‌های اخیر دیدیم همگی فایل‌های متنی هستند. یعنی فایل‌هایی که اطلاعات درون آن‌ها به صورت متنی ذخیره می‌شود و به وسیلهٔ برنامه‌های واژه‌پردازی می‌توان اطلاعات درون این فایل‌ها را دیده و ویرایش کرد. فایل‌های دیگری نیز وجود دارند که به آن‌ها فایل‌های دودویی (باینری) می‌گویند. اطلاعات درون این فایل‌ها به شکل کدهای اسکی ذخیره می‌شوند و معمولا به سادگی نمی‌توان فهمید که چه اطلاعاتی درون آن‌ها است. این دو گونه فایل، هر کدام کارایی و قابلیت‌های خاص خود را دارند.

پردازش فایل، فقط خواندن و نوشتن ترتیبی اطلاعات نیست. گاهی لازم است اطلاعات دو یا چند فایل را با هم ترکیب کنید. ممکن است بخواهید فایلی که از قبل وجود داشته را ویرایش کنید و فقط بخشی از آن فایل را تغییر دهید. ممکن است بخواهید به انتهای یک فایل، اطلاعاتی اضافه کنید یا این که اطلاعاتی به ابتدای فایل بیافزایید و یا اطلاعاتی را در محل خاصی از میانهٔ فایل قرار دهید. هر زبان

برنامه‌نویسی از جمله C++ برای تمامی این اعمال توابع و امکانات خاصی تدارک دیده است. هر چند در این کتاب برای تشریح مطالب فوق مجالی نیست، اما همین نکته شروع خوبی است تا راجع به پردازش فایل بیشتر تحقیق کنید و دانش بالاتری کسب نمایید.

پرسش‌های گزینه‌ای

1 – رشته‌های کاراکتری با کاراکتر ..... پایان می‌یابند.

الف) '\n' ب) '\t' ج) '\b' د) '\Ø'

2 – در مورد دستور `char str[]="test";` کدام گزینه صحیح است؟

الف) str یک آرایه پنج عنصری است

ب) str یک آرایه چهار عنصری است

ج) str یک آرایه بدون عنصر است

د) str یک آرایه تک عنصری است

3 – دستور `cout << str;` چه کاری انجام می‌دهد؟

الف) اگر str از نوع int* باشد، آدرس درون آن را چاپ می‌کند

ب) اگر str از نوع char* باشد، رشتهٔ کاراکتری درون آن را چاپ می‌کند

ج) اگر str از نوع float* باشد، رشتهٔ کاراکتری درون آن را چاپ می‌کند

د) الف و ب صحیح است

4 – در مورد دستور `char* p[] = "test"` کدام گزینه صحیح است؟

الف) p یک آرایهٔ پنج عنصری است

ب) p یک آرایهٔ چهار عنصری است

ج) p یک آرایهٔ بدون عنصر است

د) p یک آرایهٔ تک عنصری است

5 – کدام دستور، یک رشتهٔ کاراکتری تعریف می‌کند که با NUL خاتمه می‌یابد؟

الف) `char p="test";` ب) `char* p="test";`

ج) `char p[]="test";` د) `char* p[]="test";`

6 – کدام تابع، عضو cin نیست؟

الف) put() ب) getline() ج) seek() د) get()

7 – نقش کاراکتر '9' در دستور `cin.getline(str, p, '9');` چیست؟

الف) تعداد کاراکترهایی که باید به درون str خوانده شوند را نشان می‌دهد

ب) تعداد سطرهایی که باید به درون str خوانده شوند را نشان می‌دهد

ج) همهٔ کاراکترهای ورودی به غیر از کاراکتر '9' به درون Str منتقل می‌شوند
د) '9' کاراکتر مرزبندی در ورودی است

8 – در مورد کد ; (cin >> x) = result کدام گزینه صحیح است؟

الف – x از هر نوعی می‌تواند باشد ولی result باید از نوع bool باشد
ب – result از هر نوعی می‌تواند باشد ولی x باید از نوع bool باشد
ج – x فقط باید از نوع char و result فقط باید از نوع bool باشد.
د – result فقط باید از نوع char و x فقط باید از نوع bool باشد

9 – کدام کاراکتر را نمی‌توان از طریق عملگر ورودی << دریافت نمود؟

الف – 'n' ب – ' ' ج – '0' د – '!'

10 – اگر n از نوع int با مقدار اولیهٔ 32 اعلان شده باشد و کد ;cin >> n اجرا شود و ورودی به شکل 'p327' تایپ شود، آنگاه مقدار n برابر است با:

الف – p ب – 0 ج – 32 د – 327

11 – تابع کمکی cin.get() در کدام سرفایل معرفی شده؟

الف - <iostream> ب - <fstream>
ج - <iomanip> د - <cmath>

12 – برای این که کد ; cout.put(c) درست کار کند، متغیر c باید:

الف – از نوع string باشد ب – از نوع float باشد
ج – از نوع char باشد د – از نوع double باشد

13 – اگر s1 از نوع string و s2 از نوع *char باشد، آنگاه:

الف – s1 را می‌توان مقداردهی اولیه کرد ولی S2 را نمی‌توان.
ب – s1 را می‌توان صریحا از روی یک متغیر هم‌نوع کپی کرد ولی s2 را نمی‌توان.
ج – s1 را می‌توان در خروجی نمایش داد ولی s2 را نمی‌توان.
د – s1 را می‌توان مستقیما از ورودی دریافت کرد ولی S2 را نمی‌توان.

14 – اگر s1 از نوع string و با مقدار اولیهٔ "1234" باشد، آنگاه مقدار S1[2] برابر است با:

الف – '2' ب – '3' ج – "12" د – "34"

15 – اگر رشتهٔ s از نوع string و رشتهٔ c از نوع char باشد، آنگاه برای پیدا کردن طول s و c به ترتیب از کدام تابع استفاده می‌کنیم؟

الف – برای s از تابع strlen() و برای c از تابع length()

ب – برای s از تابع length() و برای c از تابع strlen()

ج – برای s و c هر دو از تابع length()

د – برای s و c هر دو از تابع strlen()

16 – اگر متغیر n از نوع string و با مقدار "IRAN" باشد، پس از اجرای کد n.erase(1,2); مقدار n برابر است با:

الف – "IR" ب – "RA" ج – "AN" د – "IN"

17 – اگر متغیر str1 از نوع string و با مقدار "ABCD" باشد، کدام دستور مقدار str1 را به "EFGH" تبدیل می‌کند؟

الف – str1.replace(0,0,"EFGH");

ب – str1.replace(1,1,"EFGH");

ج – str1.replace(0,4,"EFGH");

د – str1.replace(1,4,"EFGH");

18 – اگر t1_ از نوع string و با مقدار "PNU" باشد، آنگاه فراخوانی t1_.find("PU"); چه مقداری را برمی‌گرداند؟

الف – 0 ب – 1 ج – 2 د – 3

19 – برای به کارگیری فایل‌ها در c++ باید کدام سرفایل را به برنامه بیافزاییم؟

الف – <iostream> ب – <fstream>

ج – <iomanip> د – <cmath>

20 – در مورد کد ifstream file1("test.txt"); کدام گزینه صحیح است؟

الف – file1 فقط می‌تواند درون فایل test.txt بنویسد.

ب – file1 فقط می‌تواند از فایل test.txt بخواند.

ج – file1 هم می‌تواند درون فایل test.txt نوشته و هم می‌تواند از آن بخواند

د – file1 فقط برای پاک کردن فایل test.txt استفاده شده است.

پرسش‌های تشریحی

1- به اعلان‌های زیر دقت کنید:

```
char s[6];
char s[6] = {'H', 'e', 'l', 'l', 'o'};
char s[6] = "Hello";
char s[];
char s[] = new char[6];
char s[] = {'H', 'e', 'l', 'l', 'o'};
char s[] = "Hello";
char s[] = new("Hello");
char* s;
char* s = new char[6];
char* s = {'H', 'e', 'l', 'l', 'o'};
char* s = "Hello";
char* s = new("Hello");
```

الف - کدام یک از آن‌ها یک اعلان معتبر C++ برای رشتهٔ کاراکتری است؟

ب - کدام یک از آن‌ها یک اعلان معتبر C++ برای رشتهٔ کاراکتری به طول 5 است که در زمان کامپایل آدرس‌دهی شده و مقدار اولیهٔ "Hello" به آن اختصاص می‌یابد؟

ج - کدام یک از آن‌ها یک اعلان معتبر C++ برای رشتهٔ کاراکتری به طول 5 است که در زمان اجرا آدرس‌دهی شده و مقدار اولیهٔ "Hello" به آن اختصاص می‌یابد؟

د - کدام یک از آن‌ها یک اعلان معتبر C++ برای رشتهٔ کاراکتری به عنوان پارامتر یک تابع است؟

2- چه اشتباهی در استفاده از دستور زیر است اگر بخواهیم کد زیر، ورودی "Hello, word" را به درون رشتهٔ کاراکتری s بخواند؟

```
cin >> s;
```

3- کد زیر چه چیزی چاپ می‌کند؟

```
char s[] = "123 W. 42nd St., NY, NY 10020-1095";
int count = 0;
for (char* p = s; *p; p++)
if (isupper(*p)) ++ count;
cout << count << endl;
```

4- کد زیر چه چیزی چاپ می‌کند؟

```
char s[] = "123 W. 42nd St., NY, NY 10020-1095";
for (char* p = s; *p; p++)
if (isupper(*p)) *p = tolower(*p);
cout << s << endl;
```

5- کد زیر چه چیزی چاپ می‌کند؟

```
char s[] = "123 W. 42nd St., NY, NY 10020-1095";
for (char* p = s; *p; p++)
if (isupper(*p)) (*p)++;
cout << s << endl;
```

6- کد زیر چه چیزی چاپ می‌کند؟

```
char s[] = "123 W. 42nd St., NY, NY 10020-1095";
int count = 0;
for (char* p = s; *p; p++)
if (ispunct(*p)) ++ count;
cout << count << endl;
```

7- کد زیر چه چیزی چاپ می‌کند؟

```
char s[] = "123 W. 42nd St., NY, NY 10020-1095";
for (char* p = s; *p; p++)
if (ispunct(*p)) *(p-1) = tolower(*p);
cout << s << endl;
```

8- اگر s1 و s2 از نوع char* باشند آنگاه دو دستور زیر چه تفاوتی با هم دارند؟

```
char* s1 = s2;
strcpy(s1,s2);
```

9- اگر first شامل رشتهٔ "Rutherford"و last شامل رشتهٔ "Hayes" باشد، آنگاه تاثیر هر یک از فراخوانی‌های زیر چه خواهد بود؟

الف- `int n = strlen(first);`

ب- `char* s1 = strchr(first, 'r');`

پ- `char* s1 = strrchr(first, 'r');`

ت- `char* s1 = strpbrk(first, "rstuv");`

ث- `strcpy(first, last);`

ج- strncpy(first, last, 3);
چ- strcat(first, last);
ح- strncat(first, last, 3);

10- هر یک از جایگزینی‌های زیر چه چیزی را درون n قرار می‌دهد؟

الف- int n = strspn("abecedarian","abcde");
ب- int n = strspn("beefeater","abcdef");
پ- int n = strspn("baccalaureate","abc");
ت- int n = strcspn("baccalaureate","rstuv");

11- کد زیر چه چیزی چاپ می‌کند؟

```
char* s1 = "ABCDE";
char* s2 = "ABC";
if (strcmp(s1,s2) < 0) cout << s1 << " < " << s2 << endl;
else cout << s1 << " >= " << s2 << endl;
```

12- کد زیر چه چیزی چاپ می‌کند؟

```
char* s1 = "ABCDE";
char* s2 = "ABCE";
if (strcmp(s1,s2) < 0) cout << s1 << " < " << s2 << endl;
else cout << s1 << " >= " << s2 << endl;
```

13- کد زیر چه چیزی چاپ می‌کند؟

```
char* s1 = "ABCDE";
char* s2 = "";
if (strcmp(s1,s2) < 0) cout << s1 << " < " << s2 << endl;
else cout << s1 << " >= " << s2 << endl;
```

14- کد زیر چه چیزی چاپ می‌کند؟

```
char* s1 = " ";
char* s2 = "";
if (strcmp(s1,s2) == 0) cout << s1 << " == " << s2 << endl;
else cout << s1 << " != " << s2 << endl;
```

15- چه تفاوتی بین یک رشته از نوع string و یک رشته از نوع کاراکتری است؟

16- چه تفاوتی بین ورودی قالب‌بندی شده و ورودی بدون قالب‌بندی است؟

17- چرا نمی‌توان فضای سفید را با عملگر برون کشی خواند؟

18- «جریان» چیست؟

19- چه چیزی باعث می‌شود که پردازش رشته‌ها، کار با فایل‌های نوشتی و فایل‌های خواندنی در C++ آسان‌تر از C باشد؟

تمرین‌های برنامه‌نویسی

1- توضیح دهید که چرا راهکار زیر برای مثال 11-8 کار نمی‌کند؟

```
int main()
{  char name[10][20], buffer[20];
   int count = 0;
   while (cin.getline(buffer,20))
      name[count] = buffer;
   --count;
   cout << "The names are:\n";
   for (int i = 0; i < count; i++)
      cout << "\t" << i << ". [" << name[i] << "]" << endl;
}
```

2- برنامه‌ای بنویسید که یک رشته از اسامی را خط به خط بخواند و سپس آن‌ها را چاپ کند.

3- تابعی بنویسید که یک رشتهٔ کاراکتری را بدون هیچ گونه رونوشتی معکوس کند.

4- مثال 3-8 را طوری تغییر دهید که از عبارت

```
while (cin >> word)
```

بجای عبارت

```
do..while (*word)
```

استفاده کند.

5- تابعی بنویسید که تعداد تکرار یک کاراکتر داده شده را در یک رشتهٔ کاراکتری مفروض برگرداند.

6- کد هر یک از توابع زیر را نوشته و آزمایش کنید:

```
strcpy()        strncat()        strchr()
```

```
strrchr()       strstr()        strncpy()
strcat()        strcmp()        strncmp()
strspn()        strcspn()       strpbrk()
```

7- تابعی بنویسید که درون یک رشتهٔ کاراکتری مفروض تعداد کلماتی که شامل یک کاراکتر خاص هستند را برمی‌گرداند.

8- برنامه‌ای بنویسید که یک خط از متن را خوانده و سپس همهٔ آن خط را با حروف کوچک چاپ کند.

9- برنامه‌ای بنویسید که یک خط از متن را خوانده و سپس همهٔ آن خط را با حذف جاهای خالی آن چاپ کند.

10- برنامه‌ای بنویسید که یک خط از متن را خوانده و سپس همهٔ آن خط را همراه با تعداد کلماتی که در آن خط وجود دارد چاپ کند.

11- برنامه‌ای بنویسید که یک خط از متن را خوانده و سپس ترتیب کلمات را به طور معکوس چاپ کند. برای مثال برای ورودی `today is Tuesday` خروجی `Tuesday is today` را تولید کند.

12- توضیح دهید که کد زیر چه عملی انجام می‌دهد؟

```
char cs1[] = "ABCDEFGHIJ";
vhar cs2[] = "ABCDEFGH";
cout << cs2 << endl;
cout << strlen(cs2) << endl;
cs2[4] = 'X';
if (strcmp(cs1, cs2) < 0) cout << cs1 << " < " << cs2 <<
endl;
else cout << cs1 << " >= " << cs2 << endl;
char buffer[80];
strcpy(buffer, cs1);
strcat(buffer, cs2);
char* cs3 = strchr(buffer, 'G');
cout << cs3 << endl;
```

13- توضیح دهید که کد زیر چه عملی انجام می‌دهد؟

```
string s = "ABCDEFGHIJKLMNOP";
cout << s << endl;
cout << s.length() << endl;
```

```
s[8] = '!';
s.replace(8, 5, "xyz");
s.erase(6, 4);
cout << s.find("!");
cout << s.find("?");
cout << s.substr(6, 3);
s += "abcde";
string part(s, 4, 8);
string stars(8, '*');
```

14- توضیح دهید که وقتی کد:

```
string s;
int n;
float x;
cin >> s >> n >> x >> s;
```

روی هر یک از دستورالعمل‌های زیر اجرا می‌شود چه اتفاقی می‌افتد؟

الف - ABC 456 7.89 XYZ
ب - ABC 4567 .89 XYZ
پ - ABC 456 7.8 9XYZ
ت - ABC456 7.8 9 XYZ
ث - ABC456 7 .89 XYZ
ج - ABC4 56 7.89XY Z
چ - AB C456 7.89 XYZ
ح - AB C 456 7.89XYZ

15- برنامه‌ای بنویسید که تعداد خطها، کلمات و تعداد تکرار حروف در وروی را بشمارد و در خروجی چاپ کند.

16- برنامهٔ تمرین قبلی را طوری تغییر دهید که اطلاعات ورودی را از یک فایل متنی بخواند.

17- ابتدا با استفاده از برنامهٔ مثال 23-8 دو فایل به نام‌های PHONE1.TXT و PHONE2.TXT ایجاد کنید و در هر کدام تعدادی نام و شمارهٔ تلفن ذخیره کنید. سپس برنامه‌ای بنویسید که محتویات این دو فایل را به ترتیب الفبایی نام‌ها در فایل سوم به نام PHONEBOOK.TXT مرتب کرده و ذخیره نماید.

فصل نهم

«شی‌گرایی»

9-1 مقدمه

اولین نرم‌افزار برای نخستین رایانه‌ها، زنجیره‌ای از صفر و یک‌ها بود که فقط عدهٔ اندکی از این توالی سر در می‌آوردند. به تدریج کاربرد رایانه گسترش یافت و نیاز بود تا نرم‌افزارهای بیش‌تری ایجاد شود. برای این منظور، برنامه‌نویسان مجبور بودند با انبوهی از صفرها و یک‌ها سر و کله بزنند و این باعث می‌شد مدت زیادی برای تولید یک نرم‌افزار صرف شود. از این گذشته، اگر ایرادی در کار برنامه یافت می‌شد، پیدا کردن محل ایراد و رفع آن بسیار مشکل و طاقت‌فرسا بود. ابداع «زبان اسمبلی[1]» جهش بزرگی به سوی تولید نرم‌افزارهای کارآمد بود.

اسمبلی قابل فهم‌تر بود و دنبال کردن برنامه را سهولت می‌بخشید. سخت‌افزار به سرعت رشد می‌کرد و این رشد به معنی نرم‌افزارهای کامل‌تر و گسترده‌تر بود. کم کم زبان اسمبلی هم جواب‌گوی شیوه‌های نوین تولید نرم‌افزار نبود. هشت خط کد اسمبلی

---

1 – Assembly

برای یک جمع ساده به معنای ده‌ها هزار خط کد برای یک برنامهٔ حسابداری روزانه است. این بار انبوه کدهای اسمبلی مشکل‌ساز شدند. «زبان‌های سطح بالا» دروازه‌های تمدن جدید در دنیای نرم‌افزار را به روی برنامه‌نویسان گشودند.

زبان‌های سطح بالا دو نشان درخشان از ادبیات و ریاضیات همراه خود آوردند: اول دستوراتی شبیه زبان محاوره‌ای که باعث شدند برنامه‌نویسان از دست کدهای تکراری و طویل اسمبلی خلاص شوند و دوم مفهوم «تابع» که سرعت تولید و عیب‌یابی نرم افزار را چندین برابر کرد. اساس کار این گونه بود که وظیفهٔ اصلی برنامه به وظایف کوچک‌تری تقسیم می‌شد و برای انجام دادن هر وظیفه، تابعی نوشته می‌شد. پس این ممکن بود که توابع مورد نیاز یک برنامه به طور هم‌زمان نوشته و آزمایش شوند و سپس همگی در کنار هم چیده شوند. دیگر لازم نبود قسمتی از نرم‌افزار، منتظر تکمیل شدن قسمت دیگری بماند. همچنین عیب‌یابی نیز آسان صورت می‌گرفت و به سرعت محل خطا یافت شده و اصلاح می‌شد. علاوه بر این، برای بهبود دادن نرم‌افزار موجود یا افزودن امکانات اضافی به آن، دیگر لازم نبود که برنامه از نو نوشته شود؛ فقط توابع مورد نیاز را تولید کرده یا بهبود می‌دادند و آن را به برنامهٔ موجود پیوند می‌زدند. از این به بعد بود که گروه‌های تولید نرم‌افزاری برای تولید نرم‌افزارهای بزرگ ایجاد شدند و بحث مدیریت پروژه‌های نرم‌افزاری و شیوه‌های تولید نرم‌افزار و چرخهٔ حیات و ... مطرح شد.

نرم‌افزارهای بزرگ، تجربیات جدیدی به همراه آوردند و برخی از این تجربیات نشان می‌داد که توابع چندان هم بی‌عیب نیستند. برای ایجاد یک نرم‌افزار، توابع زیادی نوشته می‌شد که اغلب این توابع به یکدیگر وابستگی داشتند. اگر قرار می‌شد ورودی یا خروجی یک تابع تغییر کند، سایر توابعی که با تابع مذکور در ارتباط بودند نیز باید شناسایی می‌شدند و به تناسب تغییر می‌نمودند. این موضوع، اصلاح نرم‌افزارها را مشکل می‌کرد. علاوه بر این اگر تغییر یک تابع مرتبط فراموش می‌شد، صحت کل برنامه به خطر می‌افتاد. این اشکالات برای مدیران و برنامه‌نویسان بسیار جدی بود. بنابراین باز هم متخصصین به فکر راه چاره افتادند. پس از ریاضی و ادبیات، این بار نوبت فلسفه بود.

«***شی‌گرایی***[1]» رهیافت جدیدی بود که برای مشکلات بالا راه حل داشت. این مضمون از دنیای فلسفه به جهان برنامه‌نویسی آمد و کمک کرد تا معضلات تولید و پشتیبانی نرم‌افزار کم‌تر شود. در دنیای واقعی، یک شی چیزی است که مشخصاتی دارد مثل اسم، رنگ، وزن، حجم و ... . همچنین هر شی رفتارهای شناخته شده‌ای نیز دارد مثلا در برابر نیروی جاذبه یا تابش نور یا وارد کردن فشار واکنش نشان می‌دهد. اشیا را می‌توان با توجه به مشخصات و رفتار آن‌ها دسته بندی کرد. برای نمونه، می‌توانیم همهٔ اشیایی که دارای رفتار «تنفس» هستند را در دسته‌ای به نام «جانداران» قرار دهیم و همهٔ اشیایی که چنین رفتاری را ندارند در دستهٔ دیگری به نام «جامدات» بگذاریم. بدیهی است که اعضای هر دسته را می‌توانیم با توجه به جزییات بیشتر و دقیق‌تر به زیر دسته‌هایی تقسیم کنیم. مثلا دستهٔ جانداران را می‌توانیم به زیر دسته‌های «گیاهان» و «جانوران» و «انسان‌ها» بخش‌بندی کنیم. البته هر عضو از این دسته‌ها، علاوه بر این که مشخصاتی مشابه سایر اعضا دارد، مشخصات منحصر به فردی نیز دارد که این تفاوت باعث می‌شود بتوانیم اشیای همگون را از یکدیگر تفکیک کنیم. مثلا هر انسان دارای نام، سن، وزن، رنگ مو، رنگ چشم و مشخصات فردی دیگر است که باعث می‌شود انسان‌ها را از یکدیگر تفکیک کنیم و هر فرد را بشناسیم.

در بحث شی‌گرایی به دسته‌ها «***کلاس***[2]» می‌گویند و به نمونه‌های هر کلاس «***شی***[3]» گفته می‌شود. مشخصات هر شی را «***صفت***[4]» می‌نامند و به رفتارهای هر شی «***متد***[5]» می‌گویند. درخت سرو یک شی از کلاس درختان است که برخی از صفت‌های آن عبارت است از: نام، طول عمر، ارتفاع، قطر و ... و برخی از متدهای آن نیز عبارتند از: غذا ساختن، سبز شدن، خشک شدن، رشد کردن، ... .

اما این شی‌گرایی چه گرهی از کار برنامه‌نویسان می‌گشاید؟ برنامه‌نویسی شی‌گرا بر سه ستون استوار است:

الف. بسته‌بندی[6]: یعنی این که داده‌های مرتبط، با هم ترکیب شوند و جزییات پیاده‌سازی مخفی شود. وقتی داده‌های مرتبط در کنار هم باشند، استقلال کد و پیمانه‌ای

---

1 – Object orienting
2 – Class
3 – Object
4 – Attribute
5 – Method
6 - Encapsulation

کردن برنامه راحت‌تر صورت می‌گیرد و تغییر در یک بخش از برنامه، سایر بخش‌ها را دچار اختلال نمی‌کند. مخفی کردن جزییات پیاده‌سازی که به آن «تجرید[1]» نیز می‌گویند سبب می‌شود که امنیت کد حفظ شود و بخش‌های بی‌اهمیت یک فرایند از دید استفاده‌کنندهٔ آن مخفی باشد. به بیان ساده‌تر، هر بخش از برنامه تنها می‌تواند اطلاعات مورد نیاز را ببیند و نمی‌تواند به اطلاعات نامربوط دسترسی داشته باشد و آن‌ها را دست‌کاری کند. در برخی از کتاب‌ها از واژهٔ «کپسوله کردن» یا «محصورسازی» به جای بسته‌بندی استفاده شده.

ب. وراثت[2]: در دنیای واقعی، وراثت به این معناست که یک شی وقتی متولد می‌شود، خصوصیات و ویژگی‌هایی را از والد خود به همراه دارد هرچند که این شیء جدید با والدش در برخی از جزییات تفاوت دارد. در برنامه‌نویسی نیز وراثت به همین معنا به کار می‌رود. یعنی از روی یک شیء موجود، شیء جدیدی ساخته شود که صفات و متدهای شیء والدش را دارا بوده والبته صفات و متدهای خاص خود را نیز داشته باشد. امتیاز وراثت در این است که از کدهای مشترک استفاده می‌شود و علاوه بر این که می‌توان از کدهای قبلی استفاده مجدد کرد، در زمان نیز صرفه‌جویی شده و استحکام منطقی برنامه هم افزایش می‌یابد.

ج. چند ریختی[3]: که به آن چندشکلی هم می‌گویند به معنای یک چیز بودن و چند شکل داشتن است. چندریختی بیشتر در وراثت معنا پیدا می‌کند. برای مثال گرچه هر فرزندی مثل والدش اثر انگشت دارد، ولی اثر انگشت هر شخص با والدش یا هر شخص دیگر متفاوت است. پس اثر انگشت در انسان‌ها چندشکلی دارد.

در ادامه به شکل عملی خواهیم دید که چگونه می‌توانیم مفاهیم فوق را در قالب برنامه پیاده‌سازی کنیم.

در برنامه‌نویسی، یک کلاس را می‌توان آرایه‌ای تصور کرد که اعضای آن از انوع مختلف و متفاوتی هستند و همچنین توابع نیز می‌توانند عضوی از آن آرایه باشند. یک شی نیز متغیری است که از نوع یک کلاس است. به طور کلی یک شی را می‌توان موجودیت مستقلی تصور کرد که داده‌های خاص خودش را نگهداری می‌کند و توابع

---

1 – Abstraction 2 – Inheritance 3 – Polymorphism

خاص خودش را دارد. تعاریف کلاس مشخص می‌کند شیئی که از روی آن کلاس ساخته می‌شود چه رفتاری دارد. واضح است که به منظور استفاده از شی‌گرایی، ابتدا باید کلاس‌های مورد نیاز را مشخص و تعریف کنیم . سپس می‌توانیم در برنامهٔ اصلی، اشیایی از نوع این کلاس‌ها اعلان نماییم. بقیهٔ برنامه را این اشیا پیش می‌برند.

9-2 اعلان کلاس‌ها

کد زیر اعلان یک کلاس را نشان می‌دهد. اشیایی که از روی این کلاس ساخته می‌شوند، اعداد کسری (گویا) هستند:

```
class Ratio
{  public:
      void assign(int, int);
      viod print();
   private:
      int num, den;
};
```

اعلان کلاس با کلمهٔ کلیدی class شروع می‌شود، سپس نام کلاس می‌آید و اعلان اعضای کلاس درون یک بلوک انجام می‌شود و سرانجام یک سمیکولن بعد از بلوک نشان می‌دهد که اعلان کلاس پایان یافته است. کلاسی که در کد بالا اعلان شده، Ratio نام دارد. می‌توانیم تشخیص بدهیم که در بلوک این کلاس دو تابع و دو متغیر اعلان شده است. توابع assign() و print() را ***تابع عضو***[1] می‌گوییم زیرا آن‌ها عضو این کلاس هستند. به توابع عضو، «متد» یا «سرویس» نیز گفته شده است. متغیرهای num و den را نیز ***دادهٔ عضو***[2] می‌گوییم. به غیر از توابع و داده‌های عضو، دو عبارت دیگر نیز به چشم می‌خورد: عبارت public و عبارت private . هر عضوی که ذیل عبارت public اعلان شود، یک «عضو عمومی[3]» محسوب می‌شود و هر عضوی که ذیل عبارت private اعلان شود، یک «عضو خصوصی[4]» محسوب می‌شود. تفاوت اعضای عمومی با اعضای خصوصی در این است که اعضای عمومی

---

1 – Function member
2 – Data member
3 – Public member
4 – Private member

کلاس در خارج از کلاس قابل دستیابی هستند اما اعضای خصوصی فقط در داخل همان کلاس قابل دستیابی هستند. این همان خاصیتی است که «مخفی‌سازی اطلاعات» را ممکن می‌نماید. در کلاس فوق، توابع به شکل public و متغیرها به صورت private مشخص شده‌اند.

حال ببینیم که چطور می‌توانیم از کلاس‌ها در برنامهٔ واقعی استفاده کنیم. به مثال زیر دقت کنید.

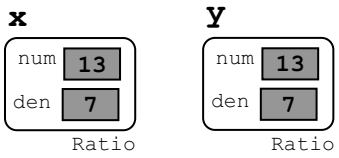
× مثال 1-9 پیاده سازی کلاس Ratio

```
class Ratio
{  public:
      void assign(int, int);
      void print();
   private:
      int num, den;
};

int main()
{  Ratio x;
   Ratio y;
   x.assign(13, 7);
   y.assign(19,5);
   cout << "x = ";
   x.print();
   cout << endl;
   cout << "y = ";
   y.print();
   cout << endl;
}

void Ratio::assign(int numerator, int denumirator)
{  num = numerator;
   den = denumirator;
}
```

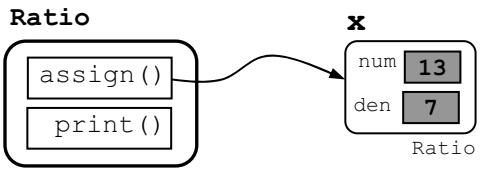
```
void Ratio::print()
{  cout << num << '/' << den;
}
```

```
x = 13/7
y = 19/5
```

در برنامهٔ بالا، ابتدا کلاس Ratio اعلان شده. سپس در برنامهٔ اصلی دو متغیر به نام‌های x و y از نوع Ratio اعلان شده‌اند. به متغیری که از نوع یک کلاس باشد یک «شی» می‌گوییم. پس در برنامهٔ بالا دو شی به نام‌های x و y داریم. هر یک از این اشیا دارای دو دادهٔ خصوصی مخصوص به خود هستند و همچنین توانایی دستیابی به دو تابع عضو عمومی را نیز دارند. این دو شی را می توان مانند شکل مقابل تصور نمود.



عبارت x.assign(13,7); موجب می‌شود که تابع عضو assign() برای شیء x فراخوانی شود. توابع عضو کلاس را فقط به این طریق می‌توان فراخوانی کرد یعنی ابتدا نام شی و سپس یک نقطه و پس از آن، تابع عضو مورد نظر. در این صورت، شیء مذکور را ***مالک***[1] فراخوانی می‌نامیم. در عبارت x.assign(13,7); شیء x مالک فراخوانی تابع عضو assign() است و این تابع روی شیء x عملیاتی انجام می دهد. تصویر مقابل این موضوع را بیان می‌کند.



تعریف دو تابع عضو assign() و print() در خطوط انتهایی برنامه آمده اما نام کلاس Ratio به همراه عملگر جداسازی دامنه :: قبل از نام هر تابع ذکر شده است. دلیل این کار آن است که کامپایلر متوجه شود که این توابع، عضوی از کلاس Ratio هستند و از قوانین آن کلاس پیروی می‌کنند.

با توجه به توضیحات بالا، می‌توانیم برنامهٔ مثال 1-9 را تفسیر کنیم. ابتدا کلاس

1 – Owner

Ratio اعلان شده است. در برنامهٔ اصلی دو شیء x و y از روی این کلاس ساخته شده که هر کدام دارای دو عضو دادهٔ خصوصی به نام‌های num و den هستند. با فراخوانی x.assign(13,7); مقادیر 7 و 13 به ترتیب درون اعضای خصوصی num و den از شیء x قرار می‌گیرد. با فراخوانی y.assign(19,5); مقادیر 19 و 5 به ترتیب درون اعضای خصوصی num و den از شیء y قرار می‌گیرد. سپس با فراخوانی x.print(); و y.print(); مقادیر اعضای خصوصی شیء x و شیء y در خروجی چاپ می‌شود. دقت کنید که دو شیء x و y کاملا از یکدیگر مجزا هستند و هر کدام داده‌های خاص خود را دارد.

چون متغیرهای num و den از اعضای خصوصی هستند، نمی‌توانیم درون برنامهٔ اصلی به آن‌ها مستقیما دستیابی کنیم. این کار فقط از طریق توابع assign() و print() میسر است زیرا این توابع عضوی از کلاس هستند و می‌توانند به اعضای خصوصی همان کلاس دستیابی کنند.

کلاس Ratio که در بالا اعلان شد، کارایی زیادی ندارد. می‌توانیم با افزودن توابع عضو دیگری، کارایی این کلاس را بهبود دهیم.

× مثال 2-9 افزودن توابع عضو بیشتر به کلاس Ratio

```
class Ratio
{  public:
      void assign(int, int);
      double convert();
      void invert();
      void print();
   private:
      int num, den;
};

int main()
{  Ratio x;
   x.assign(22, 7);
   cout << "x = ";
   x.print();
```

```
   cout << " = " << x.convert() << endl;
   x.invert();
   cout << "1/x = ";  x.print();
   cout << endl;
}

void Ratio::assign(int numerator, int denumirator)
{  num = numerator;
   den = denumirator;
}

double Ratio::convert()
{  return double(num)/den;
}

void Ratio::invert()
{  int temp = num;
   num = den;
   den = temp;
}

void Ratio::print()
{  cout << num << '/' << den;
}
```

```
x = 22/7 = 3.14286
1/x = 7/22
```

در مثال بالا با افزودن توابع عضو convert() و invert() کارایی اشیای کلاس Ratio بهبود یافته است. این دو تابع عضو نیز به صورت اعضای عمومی تعریف شده‌اند تا بتوان در برنامهٔ اصلی به آن‌ها دستیابی داشت. تابع convert() عدد کسری درون شیء مالک را به یک عدد اعشاری تبدیل می‌کند و تابع invert() نیز عدد کسری درون شیء مالک را معکوس می‌نماید.

می‌بینید که به راحتی می‌توانیم قابلیت‌های یک برنامهٔ شی‌گرا را بهبود یا تغییر دهیم بدون این که مجبور باشیم تغییرات اساسی در برنامهٔ قبلی ایجاد نماییم.

می‌توانستیم اعضای داده‌ای num و den را نیز به صورت public تعریف کنیم تا بتوانیم درون برنامهٔ اصلی به صورت مستقیم به این اعضا دستیابی کنیم اما اصل پنهان‌سازی اطلاعات این امر را توصیه نمی‌کند. این اصل می‌گوید که تا حد امکان داده‌های یک کلاس را به صورت خصوصی تعریف کنید و دستیابی به آن‌ها را به توابع عضو عمومی واگذار نمایید. به این ترتیب داده‌های اشیا از دید سایرین مخفی می‌شوند و از تغییرات ناخواسته در امان می‌مانند.

× مثال 3-9 اعلان کلاس Ratio به شکل خودکفا

کد زیر، اعلان کلاس Ratio را نشان می‌دهد که تعریف توابع عضو نیز درون همان کلاس قرار گرفته است:

```
class Ratio
{  public:
      void assign(int n, int d)  { num = n; den = d; }
      double convert() { return double(num)/den; }
      void invert() { int temp = num; num = den; den = temp;}
      void print() { cout << num << '/' << den; }
   private:
      int num, den;
};
```

اعلان فوق را با اعلان کلاس Ratio در مثال 2-9 مقایسه نمایید. در اعلان فوق، همهٔ تعاریف مورد نیاز درون خود کلاس آمده است و دیگر احتیاجی به عملگر جداسازی دامنه :: نیست. ممکن است اعلان مذکور خواناتر از اعلان مثال 2-9 باشد اما این روش در شی‌گرایی پسندیده نیست. غالبا ترجیح می‌دهند که از عملگر جداسازی دامنه و تعریف‌های خارج از کلاس برای توابع عضو استفاده کنند. در حقیقت بدنهٔ توابع اغلب در فایل جداگانه‌ای قرار می‌گیرد و به طور مستقل کامپایل می‌شود. این با اصل پنهان‌سازی اطلاعات همسویی بیشتری دارد. در پروژه‌های گروهی تولید نرم‌افزار، معمولا پیاده‌سازی کلاس‌ها به عهدهٔ مفسرین است و استفاده از کلاس‌ها در برنامهٔ اصلی بر عهدهٔ برنامه‌نویسان گذاشته می‌شود. برنامه‌نویسان فقط مایلند بدانند که کلاس‌ها چه کارهایی می‌توانند بکنند و اصلا علاقه‌ای ندارند که بدانند کلاس‌ها

چطور این کارها را انجام می‌دهند (نباید هم بدانند). برای مثال برنامه‌نویسان می‌دانند در دستور تقسیم یک عدد اعشاری بر یک عدد اعشاری دیگر، حاصل چه خواهد بود اما نمی‌دانند که عمل تقسیم چگونه انجام می‌شود و از چه الگوریتمی برای محاسبه پاسخ و تعیین دقت استفاده می‌شود. برنامه‌نویسان مجالی برای پرداختن به این جزییات ندارند و این مطالب اصلا برای آن‌ها اهمیت ندارد. فقط کافی است از صحیح بودن نتیجه مطمئن باشند. با رعایت کردن اصل پنهان‌سازی اطلاعات، هم تقسیم کارها بین افراد گروه بهتر انجام می‌شود و هم از درگیرکردن برنامهٔ اصلی با جزئیات نامربوط اجتناب شده و ساختار منطقی برنامه مستحکم‌تر می‌شود.

هنگامی که تعاریف از اعلان کلاس جدا باشد، به بخش اعلان کلاس ***رابط کلاس***[1] گفته می‌شود و به بخش تعاریف ***پیاده‌سازی***[2] می‌گویند. رابط کلاس بخشی است که در اختیار برنامه‌نویس قرار می‌گیرد و پیاده‌سازی در فایل مستقلی نگهداری می‌شود.

9-3 سازنده‌ها

کلاس Ratio که در مثال 9-1 اعلان شد از تابع assign() برای مقداردهی به اشیای خود استفاده می‌کند. یعنی پس از اعلان یک شی از نوع Ratio باید تابع assign() را برای آن شی فرا بخوانیم تا بتوانیم مقادیری را به اعضای داده‌ای آن شی نسبت دهیم. هنگام استفاده از انواع استاندارد مثل int و float می‌توانیم هم‌زمان با اعلان یک متغیر، آن را مقداردهی اولیه کنیم مثل :

```
int n=22;
float x=33.0;
```

منطقی‌تر خواهد بود اگر بتوانیم برای کلاس Ratio نیز به همین شیوه مقداردهی اولیه تدارک ببینیم. C++ این امکان را فراهم کرده که برای مقداردهی اولیه به اشیای یک کلاس، از تابع خاصی به نام ***تابع سازنده***[3] استفاده شود. تابع سازنده یک تابع عضو است که در هنگام اعلان یک شی، خود به خود فراخوانی می‌شود. نام تابع سازنده باید

---

1 – Interface 2 - Implementation 3 – Constructor Function

با نام کلاس یکسان باشد و بدون نوع بازگشتی تعریف شود. مثال زیر نشان می‌دهد که چطور می‌توانیم به جای تابع assign() از یک تابع سازنده استفاده کنیم.

× مثال 4-9 ایجاد تابع سازنده برای کلاس Ratio

```
class Ratio
{  public:
      Ratio(int n, int d) { num = n; den = d; }
      void print() { cout << num << '/' << den; }
   private:
      int num, den;
};

int main()
{  Ratio x(13,7) , y(19,5);
   cout << "x = ";
   x.print();
   cout << "and y = ";
   y.print();
}
```

```
x = 13/7 and y = 19/5
```

در کد بالا به محض این که شیء x اعلان شد، تابع سازنده به طور خودکار فراخوانی شده و مقادیر 13 و 7 به پارامترهای n و d آن ارسال می‌شود. تابع این مقادیر را به اعضای داده‌ای num و den تخصیص می‌دهد. لذا اعلان

```
Ratio x(13,7), y(19,5);
```

با خطوط زیر از مثال 1-9 معادل است:

```
Ratio x, y;
x.assign(13,7);
y.assign(19,5);
```

وظیفهٔ تابع سازنده این است که حافظهٔ لازم را برای شیء جدید تخصیص داده و آن را مقداردهی نماید و با اجرای وظایفی که در تابع سازنده منظور شده، شیء جدید را برای استفاده آماده کند.

هر کلاس می‌تواند چندین سازنده داشته باشد. در حقیقت تابع سازنده می‌تواند چندشکلی داشته باشد (بخش 13-5 را ببینید). این سازنده‌ها، از طریق فهرست پارامترهای متفاوت از یکدیگر تفکیک می‌شوند. به مثال بعدی نگاه کنید.

× مثال 9-5 افزودن چند تابع سازندهٔ دیگر به کلاس Ratio

```
class Ratio
{  public:
      Ratio() { num = 0; den = 1; }
      Ratio(int n) { num = n; den = 1; }
      Ratio(int n, int d) { num = n; den = d; }
      void print() { cout << num << '/' << den; }
   private:
      int num, den;
};

int main()
{  Ratio x, y(4), z(22,7);
   cout << "x = ";
   x.print();
   cout << "\ny = ";
   y.print();
   cout << "\nz = ";
   z.print();
}
```

```
x = 0/1
y = 4/1
z = 22/7
```

این نسخه از کلاس Ratio سه سازنده دارد: اولی هیچ پارامتری ندارد و شیء اعلان شده را با مقدار پیش‌فرض 0 و 1 مقداردهی می‌کند. دومین سازنده یک پارامتر از نوع int دارد و شیء اعلان شده را طوری مقداردهی می‌کند که حاصل کسر با مقدار آن پارامتر برابر باشد. سومین سازنده نیز همان سازندهٔ مثال 9-4 است.

یک کلاس می‌تواند سازنده‌های مختلفی داشته باشد. ساده‌ترین آن‌ها، سازنده‌ای است که هیچ پارامتری ندارد. به این سازنده ***سازندهٔ پیش‌فرض***[1] می‌گویند. اگر در یک کلاس، سازندهٔ پیش‌فرض ذکر نشود، کامپایلر به طور خودکار آن را برای کلاس مذکور ایجاد می‌کند. در مثال 9-1 که سازندهٔ پیش‌فرض منظور نکرده‌ایم، یکی به طور خودکار برای آن کلاس منظور خواهد شد.

9-4 فهرست مقداردهی در سازنده‌ها

سازنده‌ها اغلب به غیر از مقداردهی داده‌های عضو یک شی، کار دیگری انجام نمی‌دهند. به همین دلیل در C++ یک واحد دستوری مخصوص پیش‌بینی شده که تولید سازنده را تسهیل می‌نماید. این واحد دستوری ***فهرست مقداردهی***[2] نام دارد.

به سومین سازنده در مثال 9-5 دقت کنید. این سازنده را می‌توانیم با استفاده از فهرست مقداردهی به شکل زیر خلاصه کنیم:

```
Ratio(int n, int d) : num(n), den(d) { }
```

در دستور بالا، دستورالعمل‌های جایگزینی که قبلا در بدنهٔ تابع سازنده قرار داشتند، اکنون درون فهرست مقداردهی جای داده شده‌اند (فهرست مقداردهی با حروف تیره‌تر نشان داده شده). فهرست مقداردهی با یک علامت کولن : شروع می‌شود، بدنهٔ تابع نیز در انتها می‌آید (که اکنون خالی است). در مثال بعدی، سازنده‌های کلاس Ratio را با فهرست مقداردهی خلاصه کرده‌ایم.

× مثال 9-6 استفاده از فهرست مقداردهی در کلاس Ratio

```
class Ratio
{  public:
      Ratio() : num(0) , den(1) { }
      Ratio(int n) : num(n) , den(1) { }
      Ratio(int n, int d) : num(n), den(d) { }
   private:
      int num, den;
};
```

1 – Default constructor
2 – Initializing list

سازنده‌ها را می‌توانیم از این هم ساده‌تر کنیم. می‌توانیم با استفاده از پارامترهای پیش‌فرض، این سه سازنده را با هم ادغام کنیم. به مثال بعدی توجه کنید.

× مثال 7-9 به کار گیری پارامترهای پیش‌فرض در سازندهٔ کلاس Ratio

```
class Ratio
{  public:
      Ratio(int n=0, int d=1) : num(n), den(d) { }
   private:
      int num, den;
};

int main()
{  Ratio x, y(4), z(22,7);
}
```

در این مثال وقتی که برنامه اجرا شود، شیء x برابر با 0/1 و شیء y برابر با 4/1 و شیء z برابر با 22/7 خواهد شد.

قبلا در بخش 5-15 گفتیم که وقتی پارامترهای واقعی به تابع ارسال نشود، به جای آن‌ها مقادیر پیش‌فرض در تابع به کار گرفته می‌شوند. در مثال بالا، پارامتر n دارای مقدار پیش‌فرض 0 و پارامتر d دارای مقدار پیش‌فرض 1 است. وقتی شیء x از نوع Ratio و بدون هیچ پارامتری اعلان می‌شود، این مقادیر پیش‌فرض به x.n و x.d تخصیص می‌یابند. بنابراین x.num=0 و x.den=1 خواهد بود. هنگامی که شیء y فقط با مقدار ارسالی 4 اعلان می‌شود، y.num=4 خواهد شد ولی چون پارامتر دوم در اعلان شیء y ذکر نشده، مقدار پیش‌فرض 1 در y.den قرار می‌گیرد. شیء z با دو پارامتر ارسالی 22 و 7 اعلان شده. پس در این شی مقادیر پیش‌فرض نادیده گرفته شده و z.num برابر با 22 و z.den برابر با 7 خواهد شد.

9-5 توابع دستیابی

داده‌های عضو یک کلاس معمولا به صورت خصوصی (private) اعلان می‌شوند تا دستیابی به آن‌ها محدود باشد اما همین امر باعث می‌شود که نتوانیم در

مواقع لزوم به این داده‌ها دسترسی داشته باشیم. برای حل این مشکل از توابعی با عنوان ***توابع دستیابی***[1] استفاده می‌کنیم. تابع دستیابی یک تابع عمومی عضو کلاس است و به همین دلیل اجازهٔ دسترسی به اعضای داده‌ای خصوصی را دارد. از طرفی توابع دستیابی را طوری تعریف می‌کنند که فقط مقدار اعضای داده‌ای را برگرداند ولی آن‌ها را تغییر ندهد. به بیان ساده‌تر، با استفاده از توابع دستیابی فقط می‌توان اعضای داده‌ای خصوصی را خواند ولی نمی‌توان آن‌ها را دست‌کاری کرد.

× مثال 9-8 افزودن توابع دستیابی به کلاس Ratio

```
class Ratio
{  public:
      Ratio(int n=0, int d=1) : num(n) , den(d) { }
      int numerator() { return num; }
      int denomerator() { return den; }
   private:
      int num, den;
};


int main()
{  Ratio x(22,7);
   cout << x.numerator() << '/' << x.denumerator() << endl;
}
```

در این‌جا توابع numerator() و denumerator() مقادیر موجود در داده‌های عضو خصوصی را نشان می‌دهند.

9-6 توابع عضو خصوصی

تاکنون توابع عضو را به شکل یک عضو عمومی کلاس اعلان کردیم تا بتوانیم در برنامهٔ اصلی آن‌ها را فرا بخوانیم و با استفاده از آن‌ها عملیاتی را روی اشیا انجام دهیم. توابع عضو را گاهی می‌توانیم به شکل یک عضو خصوصی کلاس معرفی کنیم. واضح است که چنین تابعی از داخل برنامهٔ اصلی به هیچ عنوان قابل دستیابی نیست. این تابع

1 – Access function

فقط می‌تواند توسط سایر توابع عضو کلاس دستیابی شود. به چنین تابعی یک ***تابع سودمند*** [1] ***محلی*** می‌گوییم.

× مثال 9-9 استفاده از توابع عضو خصوصی

```
class Ratio
{  public:
      Ratio(int n=0, int d=1) : num(n), den(d) { }
      void print() { cout << num << '/' << den << endl; }
      void printconv() { cout << toFloat() << endl; }
   private:
      int num, den;
      double toFloat();
};

double Ratio::toFloat()
{  // converts Rational number to Float
   return num/den;
}
int main()
{  Ratio x(5, 100);
   x.print();
   x.printconv();
}
```

```
5/100
0.05
```

در برنامهٔ بالا، کلاس Ratio دارای یک تابع عضو خصوصی به نام toFloat() است. وظیفهٔ تابع مذکور این است که معادل ممیز شناور یک عدد کسری را برگرداند. این تابع فقط درون بدنهٔ تابع عضو printconv() استفاده شده و به انجام وظیفهٔ آن کمک می‌نماید و هیچ نقشی در برنامهٔ اصلی ندارد. یادآوری می‌کنیم که چون تابع printconv() عضوی از کلاس است، می‌تواند به تابع خصوصی toFloat() دستیابی داشته باشد.

---

1 – Utility function

توابعی که فقط به انجام وظیفهٔ سایر توابع کمک می‌کنند و در برنامهٔ اصلی هیچ کاربردی ندارند، بهتر است به صورت خصوصی اعلان شوند تا از دسترس سایرین در امان بمانند.

7-9 سازندهٔ کپی

می‌دانیم که به دو شیوه می‌توانیم متغیر جدیدی تعریف نماییم:

```
int x;
int x=k;
```

در روش اول متغیری به نام x از نوع int ایجاد می‌شود. در روش دوم هم همین کار انجام می‌گیرد با این تفاوت که پس از ایجاد x مقدار موجود در متغیر k که از قبل وجود داشته درون x کپی می‌شود. اصطلاحا x یک کپی از k است.

وقتی کلاس جدیدی تعریف می‌کنیم، با استفاده از تابع سازنده می‌توانیم اشیا را به روش اول ایجاد کنیم:

```
Ratio x;
```

در تعریف بالا، شی x از نوع کلاس Ratio اعلان می‌شود. حال ببینیم چطور می‌توانیم به شیوهٔ دوم یک کپی از شیء موجود ایجاد کنیم. برای این کار از تابع عضوی به نام ***سازندهٔ کپی***[1] استفاده می‌کنیم. این تابع نیز باید با نام کلاس هم‌نام باشد ولی سازندهٔ کپی بر خلاف تابع سازندهٔ معمولی یک پارامتر به طریقهٔ ارجاع ثابت دارد. نوع این پارامتر باید هم‌نوع کلاس مذکور باشد. این پارامتر، همان شیئی است که می‌خواهیم از روی آن کپی بسازیم. علت این که پارامتر مذکور به طریقهٔ ارجاع ثابت ارسال می‌شود این است که شیئی که قرار است کپی شود نباید توسط این تابع قابل تغییر باشد. کدهای زیر هر دو تابع سازندهٔ پیش‌فرض و سازندهٔ کپی را نشان می‌دهد:

```
Ratio();                    // default constructor
Ratio(const Ratio&);        // copy constructor
```

اولی تابع سازندهٔ پیش‌فرض است و دومی تابع سازندهٔ کپی است. حالا می‌توانیم با

1 – Copy constructor

استفاده از این تابع، از روی یک شیء موجود یک کپی بسازیم:

```
Ratio y(x);
```

کد بالا یک شی به نام y از نوع Ratio ایجاد می‌کند و تمام مشخصات شیء x را درون آن قرار می‌دهد. اگر در تعریف کلاس، سازندهٔ کپی ذکر نشود (مثل همهٔ کلاس‌های قبلی) به طور خودکار یک سازندهٔ کپی پیش‌فرض به کلاس افزوده خواهد شد. با این وجود اگر خودتان تابع سازندهٔ کپی را تعریف کنید، می‌توانید کنترل بیشتری روی برنامه‌تان داشته باشید.

× مثال 10-9 افزودن یک سازندهٔ کپی به کلاس Ratio

```
class Ratio
{  public:
      Ratio(int n=0, int d=1) : num(n), den(d) { }
      Ratio(const Ratio& r) : num(r.num), den(r.den) { }
      void print() { cout << num << '/' << den; }
   private:
      int num, den;
};
int main()
{  Ratio x(100,360);
   Ratio y(x);
   cout << "x = ";
   x.print();
   cout << ", y = ";
   y.print();
}
```

```
x = 100/360, y = 100/360
```

می‌بینید که در تعریف تابع سازندهٔ کپی نیز می‌توان از فهرست مقداردهی پیش‌فرض استفاده کرد. در مثال بالا، تابع سازندهٔ کپی طوری تعریف شده که عنصرهای num و den از پارامتر r به درون عنصرهای متناظر در شیء جدید کپی شوند. دستور Ratio y(x); باعث می‌شود که شیء y ساخته شده و سازندهٔ کپی فرا خوانده شود تا مقادیر موجود در شیء x درون y کپی شوند.

سازندهٔ کپی در سه وضعیت فرا خوانده می‌شود:

1 – وقتی که یک شی هنگام اعلان از روی شیء دیگر کپی شود

2 – وقتی که یک شی به وسیلهٔ مقدار به یک تابع ارسال شود

3 – وقتی که یک شی به وسیلهٔ مقدار از یک تابع بازگشت داده شود

برای درک این وضعیت‌ها به مثال زیر نگاه کنید.

× مثال 11-9 دنبال کردن فراخوانی‌های سازندهٔ کپی

```
class Ratio
{  public:
      Ratio(int n=0, int d=1) : num(n), den(d) { }
      Ratio(const Ratio& r) : num(r.num), den(r.den)
                  { cout << "COPY CONSTRUCTOR CALLED\n"; }
   private:
      int num, den;
};

Ratio test(Ratio r)  // calls the copy constructor, copying ? to r
{  Ratio q = r;      // calls the copy constructor, copying r to q
   return q;         // calls the copy constructor, copying q to ?
}

int main()
{  Ratio x(22,7);
   Ratio y(x);    // calls the copy constructor, copying x to y
   f(y);
}
```

```
COPY CONSTRUCTOR CALLED
COPY CONSTRUCTOR CALLED
COPY CONSTRUCTOR CALLED
COPY CONSTRUCTOR CALLED
```

بدنهٔ سازندهٔ کپی در برنامهٔ بالا شامل یک پیغام است که هر وقت سازندهٔ کپی فراخوانی شود، با چاپ آن پیغام آگاه شویم که سازندهٔ کپی فراخوانی شده. همان طور که خروجی برنامه نشان می دهد، سازندهٔ کپی چهار بار در برنامهٔ بالا فراخوانی شده:

- هنگامی که y اعلان می‌شود، فراخوانی شده و x را درون y کپی می‌کند.

- هنگامی که y به وسیلهٔ مقدار به تابع test() ارسال می‌شود، فراخوانی شده و y را به درون r کپی می‌کند.

- هنگامی که q اعلان می‌شود، فراخوانی شده و r را به درون q کپی می‌کند.

- هنگامی که q به وسیلهٔ مقدار از تابع test() بازگشت داده می‌شود، فراخوانی می‌شود. حتی اگر چیزی را جایی کپی نکند.

دستور Ratio q=r; ظاهری شبیه عمل جایگزینی دارد اما این کد در حقیقت سازندهٔ کپی را فراخوانی می‌کند و درست شبیه دستور Ratio q(r); است.

اگر یک سازندهٔ کپی در تعریف کلاستان نگنجانید، به طور خودکار یک سازندهٔ کپی برای آن منظور می‌شود که این سازنده به شکل پیش‌فرض تمام اطلاعات موجود در شیء جاری را به درون شیء تازه ساخته شده کپی می‌کند. اغلب اوقات این همان چیزی است که انتظار داریم. اما گاهی هم این کار کافی نیست و انتظارات ما را براورده نمی‌کند. مثلا فرض کنید کلاسی دارید که یک عضو آن از نوع اشاره‌گر است. در حین اجرای برنامه، این اشاره‌گر را به خانه‌ای از حافظه اشاره می‌دهید. حال اگر از این شی یک کپی بسازید بدون این که از سازندهٔ کپی مناسبی استفاده کنید، شیء جدید نیز به همان خانه از حافظه اشاره می‌کند. یعنی فقط آن اشاره‌گر کپی می‌شود نه چیزی که به آن اشاره می‌شود. در این گونه موارد لازم است خودتان تابع سازندهٔ کپی را بنویسید و دستورات لازم را در آن بگنجانید تا هنگام کپی کردن یک شی، منظورتان برآورده شود.

8-9 نابود کننده

وقتی که یک شی ایجاد می‌شود، تابع سازنده به طور خودکار برای ساختن آن فراخوانی می‌شود. وقتی که شی به پایان زندگی‌اش برسد، تابع عضو دیگری به طور خودکار فراخوانی می‌شود تا نابودکردن آن شی را مدیریت کند. این تابع عضو، ***نابودکننده***[1] نامیده می‌شود (در برخی از کتاب‌ها به آن «تخریب‌گر» یا «منهدم‌کننده»

1 – Destructor

گفته‌اند). سازنده وظیفه دارد تا منابع لازم را برای شی تخصیص دهد و نابودکننده وظیفه دارد آن منابع را آزاد کند.

هر کلاس فقط یک نابودکننده دارد. نام تابع نابودکننده باید هم نام کلاس مربوطه باشد با این تفاوت که یک علامت نقیض ~ به آن پیشوند شده. مثل تابع سازنده و سازندهٔ کپی، اگر نابود کننده در تعریف کلاس ذکر نشود، به طور خودکار یک نابودکنندهٔ پیش‌فرض به کلاس افزوده خواهد شد.

× مثال 12-9 افزودن یک نابودکننده به کلاس Ratio

```
class Ratio
{  public:
      Ratio() { cout << "OBJECT IS BORN.\n"; }
      ~Ratio() { cout << "OBJECT DIES.\n"; }
   private:
      int num, den;
};

int main()
{  {  Ratio x;                    // beginning of scope for x
      cout << "Now x is alive.\n";
   }                                      // end of scope for x
   cout << "Now between blocks.\n";
   {  Ratio y;
      cout << "Now y is alive.\n";
   }
}
```

```
OBJECT IS BORN.
Now x is alive.
OBJECT DIES.
Now between blocks.
OBJECT IS BORN.
Now y is alive.
OBJECT DIES.
```

در بدنهٔ توابع سازنده و نابودکننده پیغامی درج شده تا هنگامی که یک شی از این کلاس متولد شده یا می‌میرد، از تولد و مرگ آن آگاه شویم. خروجی نشان می‌دهد که

سازنده یا نابودکننده چه زمانی فراخوانی شده است. وقتی یک شی به پایان حوزه‌اش برسد، نابودکننده فراخوانی می‌شود تا آن شی را نابود کند. یک شیء محلی وقتی به پایان بلوک محلی برسد می‌میرد. یک شیء ثابت وقتی به پایان تابع main() برخورد شود، می‌میرد. شیئی که درون یک تابع تعریف شده، در پایان آن تابع می‌میرد.

سعی کنید تابع نابودکننده را خودتان برای کلاس بنویسید. یک برنامه‌نویس خوب، توابع سازنده و سازندهٔ کپی و نابودکننده را خودش در تعریف کلاس‌هایش می‌گنجاند و آن‌ها را به توابع پیش‌فرض سیستم واگذار نمی‌کند.

9-9 اشیای ثابت

اگر قرار است شیئی بسازید که در طول اجرای برنامه هیچ‌گاه تغییر نمی‌کند، بهتر است منطقی رفتار کنید و آن شی را به شکل ثابت اعلان نمایید. اعلان‌های زیر چند ثابت آشنا را نشان می‌دهند:

```
const char BLANK = ' ';
const int MAX_INT = 2147483647;
const double PI = 3.141592653589793;
void int(float a[], const int SIZE);
```

اشیا را نیز می‌توان با استفاده از عبارت const به صورت یک شیء ثابت اعلان کرد:

```
const Ratio PI(22,7);
```

اما در مورد اشیای ثابت یک محدودیت وجود دارد: کامپایلر اجازه نمی‌دهد که توابع عضو را برای اشیای ثابت فراخوانی کنید. مثلا در مورد کد فوق گرچه تابع print() عضوی از کلاس Ratio است اما در مورد شیء ثابت PI نمی‌توانیم آن را فراخوانی کنیم:

```
PI.print();    // error: call not allowed
```

در اصل تنها توابع سازنده و نابودکننده برای اشیای ثابت قابل فراخوانی‌اند. ولی این مشکل را می‌توان حل کرد. برای غلبه بر این محدودیت، توابع عضوی که می‌خواهیم با اشیای ثابت کار کنند را باید به صورت const تعریف کنیم. برای این که یک تابع

این چنین تعریف شود، کلمهٔ کلیدی const را بین فهرست پارامترها و تعریف بدنهٔ آن قرار می‌دهیم. مثلا تعریف تابع print() در کلاس Ratio را به شکل زیر تغییر می‌دهیم:

```
void print() const { cout << num << '/' << den << endl; }
```

اکنون می‌توانیم این تابع را برای اشیای ثابت نیز فراخوانی نماییم:

```
const Ratio PI(22,7);
PI.print();     // o.k. now
```

9-10 اشاره‌گر به اشیا

می‌توانیم اشاره‌گر به اشیای کلاس نیز داشته باشیم. از آن‌جا که یک کلاس می‌تواند اشیای داده‌ای متنوع و متفاوتی داشته باشد، اشاره‌گر به اشیا بسیار سودمند و مفید است. بهتر است قبل از مطالعهٔ مثال‌های زیر، فصل هفتم را مرور کنید.

× مثال 9-13 استفاده از اشاره‌گر به اشیا

```
class X
{  public:
      int data;
};
main()
{  X* p = new X;
   (*p).data = 22;      // equivalent to: p->data = 22;
   cout << "(*p).data = " << (*p).data << " = " << p->data << endl;
   p->data = 44;
   cout << " p->data = " << (*p).data << " = " << p->data << endl;
}
```

```
(*p).data = 22 = 22
p->data = 44 = 44
```

در این مثال، p اشاره‌گری به شیء x است. پس p* یک شیء x است و (*p).data دادهٔ عضو آن شی را دستیابی می‌کند. حتما باید هنگام استفاده از p* آن را درون پرانتز قرار دهید زیرا عملگر انتخاب عضو (.) تقدم بالاتری نسبت به عملگر مقداریابی (*) دارد. اگر پرانتزها قید نشوند و فقط p*.data نوشته شود، کامپایلر این خط را به صورت *(p.data) تفسیر خواهد کرد که این باعث خطا می‌شود.

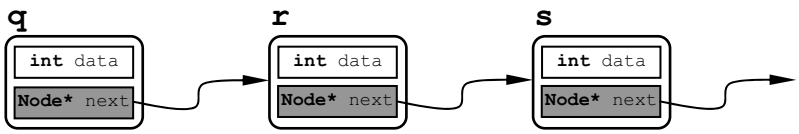
این مثال نشان می‌دهد که دو عبارت (*p).data و p->data هر دو به یک معنا هستند. بیشتر برنامه‌نویسان ترجیح می‌دهند از ترکیب p->data استفاده کنند زیرا به مفهوم «چیزی که p به آن اشاره می‌کند» نزدیک‌تر است. مثال بعدی اهمیت بیشتری دارد و کاربرد اشاره‌گر به اشیا را بهتر نشان می‌دهد.

× مثال 14-9 فهرست‌های پیوندی با استفاده از کلاس Node

به کلاسی که در زیر اعلان شده دقت کنید:

```
class Node
{  public:
      Node(int d, Node* p=0) : data(d), next(p) { }
      int data;
      Node* next;
};
```

عبارت بالا کلاسی به نام Node تعریف می‌کند که اشیای این کلاس دارای دو عضو داده‌ای هستند که یکی متغیری از نوع int است و دیگری یک اشاره‌گر از نوع همین کلاس. شاید عجیب باشد که عضوی از کلاس به شیئی از نوع همان کلاس اشاره کند اما این کار واقعا ممکن است و باعث می‌شود بتوانیم یک شی را با استفاده از همین اشاره‌گر به شیء دیگر پیوند دهیم و یک زنجیره بسازیم. مثلا اگر اشیای q و r و s از نوع Node باشند، می‌توانیم پیوند این سه شی را به صورت زیر مجسم کنیم:



به تابع سازنده نیز دقت کنید که چطور هر دو عضو داده‌ای شیء جدید را مقداردهی می‌کند. اکنون این کلاس را در برنامهٔ زیر به کار می‌گیریم:

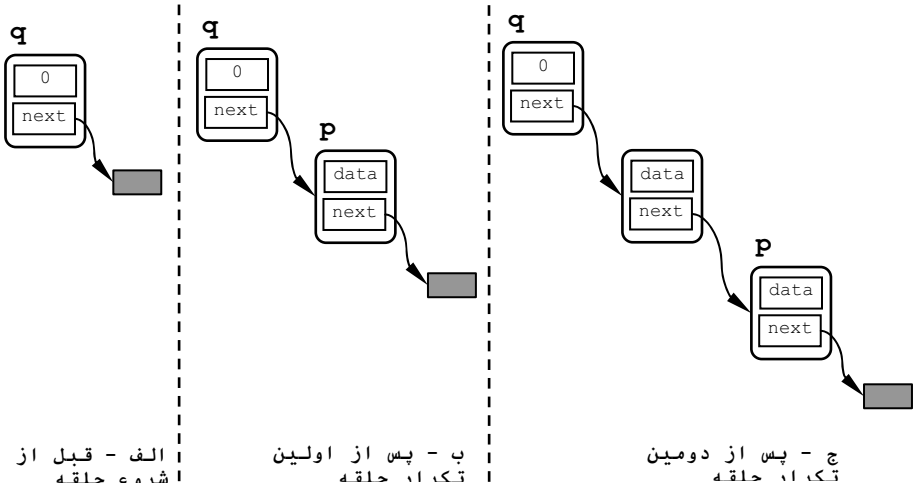
```
int main()
{  int n;
   Node* p;
   Node* q=0;
   while (cin >> n)
```

```
   {  p = new Node(n, q);
      q = p;
   }
   for ( ; p->next; p = p->next)
      cout << p->data << " -> ";
   cout << "*\n";
}
```

```
22 33 44 55 66 77 ^d
77 -> 66 -> 55 -> 44 -> 33 -> *
```

در این برنامه، ابتدا دو اشاره‌گر از نوع Node به نام p و q ساخته می‌شود که q به یک شیء خالی اشاره دارد. در حلقهٔ while پس از اولین ورودی، حافظهٔ جدیدی برای شیء p منظور می‌شود و عدد وارد شده در عضو داده‌ای data از اشاره‌گر p قرار می‌گیرد و همچنین عضو داده‌ای next برابر با q می‌شود. یعنی عضو اشاره‌گر p به حافظهٔ q اشاره می‌کند. سپس آدرس شیء p در شیء q قرار می‌گیرد. حالا q به شیء p اشاره دارد و p به یک شیء خالی.

پس از دومین ورودی، مجددا حافظهٔ جدیدی برای p منظور می‌شود و انتساب‌های فوق تکرار می‌شود تا این که p به حافظهٔ موجود قبلی اشاره می‌کند و q به شیء p اشاره می‌کند. شکل زیر روند اجرای برنامه را نشان می‌دهد.



تا زمانی که کاربر کاراکتر پایان فایل (Ctrl+Z) را فشار ندهد، حلقه ادامه یافته و هر دفعه یک عدد از ورودی گرفته شده و یک بند به زنجیرهٔ موجود اضافه می‌شود. حلقهٔ for وظیفهٔ پیمایش فهرست پیوندی را دارد. به این صورت که تا وقتی
p->next برابر با NUL نباشد، حلقه ادامه می‌یابد و عضو داده‌ای گره فعلی را چاپ می‌کند و به این ترتیب کل فهرست پیمایش می‌شود. واضح است که برای پیمودن این فهرست پیوندی باید آن را به شکل معکوس پیمود.

اشاره‌گر به اشیا بسیار سودمند و مفید است به حدی که بحث راجع به اشاره‌گرها و الگوریتم‌ها و مزایای آن به شاخهٔ مستقلی در برنامه‌نویسی تبدیل شده و «ساختمان داده‌ها[1]» نام گرفته است. به اختصار می‌گوییم که اشاره‌گر به اشیا برای ساختن فهرست‌های پیوندی و درخت‌های داده‌ای به کار می‌رود. این‌ها بیشتر برای پردازش‌های سریع مثل جستجو در فهرست‌های طولانی (مانند فرهنگ لغات) یا مرتب‌سازی رکوردهای اطلاعاتی استفاده می‌شوند. برای مطالعه در این زمینه به مراجع ساختمان داده‌ها مراجعه کنید.

9-11 اعضای داده‌ای ایستا

هر وقت که شیئی از روی یک کلاس ساخته می‌شود، آن شی مستقل از اشیای دیگر، داده‌های عضو خاص خودش را دارد. گاهی لازم است که مقدار یک عضو داده‌ای در همهٔ اشیا یکسان باشد. اگر این عضو مفروض در همهٔ اشیا تکرار شود، هم از کارایی برنامه می‌کاهد و هم حافظه را تلف می‌کند. در چنین مواقعی بهتر است آن عضو را به عنوان یک ***عضو ایستا***[2] اعلان کنیم. عضو ایستا عضوی است که فقط یک نمونه از آن ایجاد می‌شود و همه اشیا از همان نمونهٔ مشترک استفاده می‌کنند. با استفاده از کلمهٔ کلیدی static در شروع اعلان متغیر، می‌توانیم آن متغیر را به صورت ایستا اعلان نماییم. یک متغیر ایستا را فقط باید به طور مستقیم و مستقل از اشیا مقداردهی نمود. کد زیر نحوهٔ اعلان و مقداردهی یک عضو داده‌ای ایستا را بیان می‌کند:

```
class X
{  public:
```

1 – Data structure
2 – Static member

```
   static int n;      // declaration of n as a static data member
};
int X::n = 0;         // definition of n
```

خط آخر نشان می‌دهد که متغیرهای ایستا را باید به طور مستقیم و مستقل از اشیا مقداردهی کرد.

متغیرهای ایستا به طور پیش‌فرض با صفر مقداردهی اولیه می‌شوند. بنابراین مقداردهی صریح به این گونه متغیرها ضروری نیست مگر این که بخواهید یک مقدار اولیهٔ غیر صفر داشته باشید.

× مثال 15-9 یک عضو داده‌ای ایستا

کد زیر، کلاسی به نام widget اعلان می‌کند که این کلاس یک عضو داده‌ای ایستا به نام count دارد. این عضو، تعداد اشیای widget که موجود هستند را نگه می‌دارد. هر وقت که یک شیء widget ساخته می‌شود، از طریق سازنده مقدار count یک واحد افزایش می‌یابد و هر زمان که یک شیء widget نابود می‌شود، از طریق نابودکننده مقدار count یک واحد کاهش می‌یابد:

```
class Widget
{  public:
      Widget() { ++count; }
      ~Widget() { --count; }
      static int count;
};

int Widget::count = 0;
main()
{  Widget w, x;
   cout << "Now there are " << w.count << " widgets.\n";
   {  Widget w, x, y, z;
      cout << "Now there are " << w.count << " widgets.\n";
   }
   cout << "Now there are " << w.count << " widgets.\n";
   Widget y;
   cout << "Now there are " << w.count << " widgets.\n";
}
```

```
Now there are 2 widgets.
Now there are 6 widgets.
Now there are 2 widgets.
Now there are 3 widgets.
```

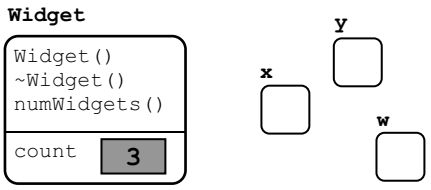
توجه کنید که چگونه چهار شیء widget درون بلوک داخلی ایجاد شده است. هنگامی که اجرای برنامه از آن بلوک خارج می‌شود، این اشیا نابود می‌شوند و لذا تعداد کل widgetها از 6 به 2 تقلیل می‌یابد.

یک عضو داده‌ای ایستا مثل یک متغیر معمولی است: فقط یک نمونه از آن موجود است بدون توجه به این که چه تعداد شی از آن کلاس موجود باشد. از آن‌جا که عضو داده‌ای ایستا عضوی از کلاس است، می‌توانیم آن را به شکل یک عضو خصوصی نیز اعلان کنیم.

× مثال 16-9 یک عضو داده‌ای ایستا و خصوصی

```
class Widget
{  public:
      Widget() { ++count; }
      ~Widget() { --count; }
      int numWidgets() { return count; }
   private:
      static int count;
};

int Widget::count = 0;
main()
{  Widget w, x;
   cout << "Now there are " << w.numWidgets() << " widgets.\n";
   {  Widget w, x, y, z;
      cout << "Now there are " << w.numWidgets() << " widgets.\n";
   }
   cout << "Now there are " << w.numWidgets() << " widgets.\n";
   Widget y;
   cout << "Now there are " << w.numWidgets() << " widgets.\n";
}
```

این برنامه مانند مثال 9-15 کار می‌کند با این تفاوت که متغیر ایستای count به شکل یک عضو خصوصی اعلان شده و به همین دلیل به تابع دستیابی numWidgets() نیاز داریم تا بتوانیم درون برنامهٔ اصلی به متغیر count دسترسی داشته باشیم.



می‌توانیم کلاس Widget و اشیای x و y و w را مانند مقابل تصور کنیم:

می‌بینید که متغیر ایستای count درون خود کلاس جای گرفته نه درون اشیا.

9-12 توابع عضو ایستا

با دقت در مثال قبلی به دو ایراد بر می‌خوریم: اول این که گرچه متغیر count یک عضو ایستا است ولی برای خواندن آن حتما باید از یک شیء موجود استفاده کنیم. در مثال قبلی از شیء w برای خواندن آن استفاده کرده‌ایم. این باعث می‌شود که مجبور شویم همیشه مواظب باشیم عضو ایستای مفروض از طریق یک شی که الان موجود است فراخوانی شود. مثلا در مثال قبلی اگر در قسمتی از برنامه، دور از چشم ما، شیء w نابود شود، آنگاه فراخوانی w.numWidgets() از آن به بعد مخاطره‌آمیز خواهد بود. ایراد دوم این است که اگر هیچ شیئی موجود نباشد، نمی‌توانیم عضو ایستای count را دستیابی کنیم. برای رفع این دو ایراد کافی است تابع دستیابی کننده را نیز به شکل ایستا تعریف کنیم.

× مثال 9-17 یک تابع عضو ایستا

کد زیر همان کد مثال قبلی است با این فرق که در این کد، تابع دستیابی کننده نیز به شکل ایستا اعلان شده است:

```
class Widget
{  public:
      Widget() { ++count; }
      ~Widget() { --count; }
      static int num() { return count; }
```

```
   private:
      static int count;
};

int Widget::count = 0;

int main()
{  cout << "Now there are " << Widget::num() << " widgets.\n";
   Widget w, x;
   cout << "Now there are " << Widget::num() << " widgets.\n";
   {  Widget w, x, y, z;
      cout << "Now there are " << Widget::num() << " widgets.\n";
   }
   cout << "Now there are " << Widget::num() << " widgets.\n";
   Widget y;
   cout << "Now there are " << Widget::num() << " widgets.\n";
}
```

وقتی تابع num() به صورت ایستا تعریف شود، از اشیای کلاس مستقل می‌شود و برای فراخوانی آن نیازی به یک شیء موجود نیست و می‌توان با کد Widget::num() به شکل مستقیم آن را فراخوانی کرد.

تا این جا راجع به شیگرایی و نحوهٔ استفاده از آن در برنامه‌نویسی مطالبی آموختیم. اکیدا توصیه می‌کنیم که قبل از مطالعهٔ فصل بعدی، همهٔ تمرین‌های پایان این فصل را حل کنید تا با برنامه‌نویسی شیگرا مأنوس شوید. در فصل بعدی مطالبی را خواهیم دید که شیگرایی را مفیدتر می‌کنند.

پرسش‌های گزینه‌ای

1 – کدام گزینه از مزایای شی‌گرایی نیست؟

الف – وراثت ب – چندریختی ج – بسته‌بندی د – نمونه‌سازی

2 – در اعلان یک کلاس، کدام گزینه صحیح است؟

الف – برای معرفی اعضای عمومی کلاس از عبارت `private` استفاده می‌شود

ب – برای معرفی اعضای خصوصی کلاس از عبارت `public` استفاده می‌شود

ج – توابع و متغیرها هر دو می‌توانند عضو یک کلاس باشند

د – برای معرفی اعضای تابعی کلاس از عبارت `funct` استفاده می‌شود

3 – کدام گزینه صحیح نیست؟

الف – به اعلان کلاس، رابط کلاس گفته می‌شود

ب – به بدنهٔ کلاس، پیاده‌سازی کلاس گفته می‌شود

ج – به متغیری که از نوع یک کلاس باشد، شی گفته می‌شود

د – به تابعی که عضو یک کلاس باشد، تابع دستیابی گفته می‌شود

4 – از گزینه‌های زیر، کدام صحیح است؟

الف – هر کلاس فقط یک سازنده و فقط یک نابودکننده دارد

ب – هر کلاس فقط یک سازنده دارد و می‌تواند چند نابودکننده داشته باشد

ج – هر کلاس فقط یک نابودکننده دارد و می‌تواند چند سازنده داشته باشد

د – هر کلاس می‌تواند چند سازنده و چند نابودکننده داشته باشد

5 – تابع دستیابی چیست؟

الف – یک تابع عضو عمومی کلاس است که به یک دادهٔ عضو عمومی دستیابی دارد

ب – یک تابع عضو خصوصی کلاس است که به یک دادهٔ عضو خصوصی دستیابی دارد

ج – یک تابع عضو عمومی کلاس است که به یک دادهٔ عضو خصوصی دستیابی دارد

د – یک تابع عضو خصوصی کلاس است که به یک داده عضو عمومی دستیابی دارد

6 – اگر کلاسی به نام vector داشته باشیم آنگاه کدام تابع زیر، سازندهٔ کپی را برای این کلاس اعلان می‌کند؟

الف – `vector()` ب – `vector(const vector&)`

ج – `~vector()` د – `vector*(const vector)`

7 – سازندهٔ کپی وقتی فراخوانی می‌شود که:

الف – یک شی به وسیلهٔ مقدار به یک تابع فرستاده شود

ب – یک شی به وسیلهٔ ارجاع به یک تابع فرستاده شود

ج – یک شی به وسیلهٔ ارجاع ثابت به یک تابع فرستاده شود

د – یک شی به وسیلهٔ ارجاع از یک تابع بازگشت داده شود

8 – اگر در تعریف یک کلاس، سازندهٔ کپی ذکر نشود آنگاه:

الف – از اشیای آن کلاس نمی‌توان کپی ایجاد کرد

ب – اشیای آن کلاس را نمی‌توان به تابع فرستاد

ج – اشیای آن کلاس را نمی‌توان از تابع بازگشت داد

د – یک سازندهٔ کپی پیش‌فرض به طور خودکار به کلاس افزوده می‌شود

9 – اگر شیء x در تابع مفروض f() به شکل محلی اعلان شده باشد، آنگاه :

الف – با شروع تابع `main()` شیء `x` ایجاد می‌شود و در انتهای تابع `main()` می‌میرد

ب – با شروع تابع `main()` شیء `x` ایجاد می‌شود و در انتهای تابع `f()` می‌میرد

ج – با شروع تابع `f()` شیء `x` ایجاد می‌شود و در انتهای تابع `f()` می‌میرد

د – با شروع تابع `f()` شیء `x` ایجاد می‌شود و در انتهای تابع `main()` می‌میرد

10 – کدام گزینه در مورد کلاس ها صحیح است؟

الف – اشاره‌گرها می‌توانند عضو کلاس باشند ولی نمی‌توانند از نوع کلاس باشند

ب – اشاره‌گرها می‌توانند از نوع کلاس باشند ولی نمی‌توانند عضو کلاس باشند

ج – اشاره‌گرها می‌توانند از نوع کلاس باشند به شرطی که عضو آن کلاس نباشند

د – اشاره‌گرها می‌توانند عضو کلاس باشند و می‌توانند از نوع کلاس باشند

11 – اگر متغیر k یک عضو ایستا برای کلاس vector بوده و x1 و x2 اشیایی از کلاس vector باشند، آنگاه:

الف – از k فقط یک نمونه در سراسر برنامه موجود است.

ب – x1 و x2 هر کدام عضو k خاص خود را دارند

ج – فقط x1 دارای عضو k است و x2 از همان k استفاده می‌کند

د – فقط x2 دارای عضو k است و x1 از همان k استفاده می‌کند

12 – کدام گزینه در مورد اعضای ایستای کلاس، صحیح نیست؟

الف – اعضای ایستا با کلمهٔ کلیدی static مشخص می‌شوند

ب – اعضای ایستا می‌توانند عضو عمومی کلاس باشند

ج – اعضای ایستا می‌توانند عضو خصوصی کلاس باشند

د – اعضای ایستا در بخش static: اعلان می‌شوند

13 - «فهرست مقداردهی» در کدام تابع عضو کلاس استفاده می‌شود؟

الف – تابع سودمند محلی ب – تابع نابودکننده

ج – تابع سازنده د – تابع دستیابی

14 – اگر شیء m1 از کلاس media باشد، آنگاه با اجرای کد media m2=m1; کدام تابع عضو کلاس فراخوانی می‌شود؟

الف – تابع سازنده ب – تابع سازندهٔ کپی

ج – تابع دستیابی د – تابع سودمند محلی

پرسش‌های تشریحی

1- تفاوت بین یک عضو عمومی و یک عضو خصوصی از یک کلاس را توضیح دهید.

2- تفاوت بین رابط کلاس و پیاده‌سازی کلاس را توضیح دهید.

3- تفاوت بین تابع عضو کلاس و تابع کاربردی را توضیح دهید.

4- تفاوت بین سازنده و نابودکننده را توضیح دهید.

5- تفاوت بین سازندهٔ پیش‌فرض و سازنده‌های دیگر را توضیح دهید.

6- تفاوت بین سازندهٔ کپی و عملگر جایگزینی را توضیح دهید.

7- تفاوت بین تابع دستیابی و تابع سودمند محلی را توضیح دهید.

8- نام تابع سازنده چگونه باید باشد؟

9- نام تابع نابودکننده باید چگونه باشد؟

10- هر کلاس چه تعداد سازنده می‌تواند داشته باشد؟

11- هر کلاس چه تعداد نابودکننده می‌تواند داشته باشد؟

12- چگونه و چرا از عملگر جداسازی حوزه :: در تعریف کلاس‌ها استفاده می‌شود؟

13- کدام تابع عضو به طور خودکار توسط کامپایلر ایجاد می‌شود اگر برنامه‌نویس آن را صریحا در تعریف کلاس نگنجانیده باشد؟

14- در کد زیر چند دفعه سازندهٔ کپی فراخوانی می‌شود؟

```
Widget f(Widget u)
{  Widget v(u);
   Widget w = v;
   return w;
}

main()
{  Widget x;
   Widget y = f(f(x));
}
```

15- چرا در عبارت (*P).data وجود پرانتزها ضروری است؟

تمرین‌های برنامه‌نویسی

1- کلاس Point را برای نقاط سه بعدی (x, y, z) پیاده‌سازی کنید. یک سازندهٔ پیش‌فرض، یک سازندهٔ کپی، یک تابع negate() تا نقطهٔ مورد نظر را منفی کند، یک تابع norm() برای برگرداندن فاصله از مبداء (0, 0, 0) و یک تابع print() به این کلاس اضافه کنید.

2- کلاس stack را برای پشته‌هایی از نوع int پیاده‌سازی کنید. یک سازندهٔ پیش‌فرض، یک نابودکننده و توابع اجرای عملیات معمول پشته push() و pop() و isEmpty() و isFull() را به این کلاس اضافه کنید. از آرایه‌ها برای این پیاده‌سازی استفاده کنید.

3- کلاس Time را پیاده‌سازی کنید. هر شی از این کلاس، یک زمان ویژه از روز را نشان می‌دهد که ساعت، دقیقه و ثانیه را به شکل یک عدد صحیح نگهداری می‌کند. یک سازنده، توابع دستیابی، تابع advance(int h,int m,int s) برای جلو بردن زمان فعلی یک شیء موجود، تابع reset(int h,int m,int s) برای نو کردن زمان فعلی یک شیء موجود و یک تابع print() به این کلاس اضافه کنید.

4- کلاس Random را برای تولید کردن اعداد شبه‌تصادفی پیاده‌سازی کنید.

5- کلاس person را پیاده‌سازی کنید. هر شی از این کلاس، نمایانگر یک انسان است. اعضای داده‌ای این کلاس باید شامل نام شخص، سال تولد و سال وفات باشد. یک تابع سازندهٔ پیش‌فرض، نابودکننده، توابع دستیابی و یک تابع print() به این کلاس اضافه کنید.

6- کلاس Matrix را برای آرایه‌های 2×2 پیاده‌سازی کنید:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

یک سازندهٔ پیش‌فرض، یک سازندهٔ کپی، یک تابع inverse() که معکوس آرایه را برمی‌گرداند، یک تابع det() که دترمینان آرایه را برمی‌گرداند، یک تابع بولی

isSingular() که بسته به این که دترمینان صفر باشد یا نه مقدار یک یا صفر را برمی‌گرداند و یک تابع print() به این کلاس اضافه کنید.

7- یک کلاس point برای نقاط دو بعدی (x, y) پیاده‌سازی کنید. یک سازندهٔ پیش‌فرض، یک سازندهٔ کپی، یک تابع negate() برای تبدیل نقطهٔ مورد نظر به منفی، یک تابع norm() برای برگرداندن فاصلهٔ نقطه از مبداء (0,0) و یک تابع print() به این کلاس اضافه کنید.

8- کلاس Circle را پیاده‌سازی کنید. هر شی در این کلاس یک دایره را نشان می‌دهد که شعاع آن و مختصات x و y از مرکز را به صورت float نگهداری می‌کند. یک سازندهٔ پیش‌فرض، توابع دستیابی، یک تابع area() و یک تابع circumference() که محیط دایرهٔ مذکور را برمی‌گرداند، به این کلاس اضافه کنید.

9- کلاس Stack در مسألهٔ 2 را با افزودن تابع count() به آن، تغییر دهید. تابع مذکور تعداد اقلام درون پشته را برمی‌گرداند.

10- کلاس Stack در مسألهٔ قبل را با افزودن تابع print() تغییر دهید. تابع مذکور محتویات پشته را چاپ می‌کند.

11- کلاس Stack در مسألهٔ قبل را طوری تغییر دهید که به جای مقادیر نوع int، مقدارهای float را نگهداری کند.

12- کلاس Circle در مسألهٔ 8 را طوری تغییر دهید که شامل تابع area() باشد. این تابع مساحت دایرهٔ مذکور را برمی‌گرداند.

13- کلاس Matrix در مسألهٔ 6 را طوری تغییر دهید که آرایه‌های 3×3 را نگهداری کند. توابع عضو را طوری تغییر دهید که با این آرایه‌ها سازگار باشند.

فصل دهم

«سربارگذاری عملگرها»

10-1 مقدمه

در C++ مجموعه‌ای از 45 عملگر مختلف وجود دارد که برای کارهای متنوعی استفاده می‌شوند. همهٔ این عملگرها برای کار کردن با انواع بنیادی (مثل int و float و char) سازگاری دارند. هنگامی که کلاسی را تعریف می‌کنیم، در حقیقت یک نوع جدید را به انواع موجود اضافه کرده‌ایم. ممکن است بخواهیم اشیای این کلاس را در محاسبات ریاضی به کار ببریم. اما چون عملگرهای ریاضی (مثل + یا = یا =*) چیزی راجع به اشیای کلاس جدید نمی‌دانند، نمی‌توانند به درستی کار کنند. C++ برای رفع این مشکل چاره اندیشیده و امکان ***سربارگذاری عملگرها***[1] را تدارک دیده است. سربارگذاری عملگرها به این معناست که به عملگرها تعاریف جدیدی اضافه کنیم تا بتوانند با اشیای کلاس مورد نظر به درستی کار کنند.

---

1 – Overloading operators

در سربارگذاری عملگرها محدودیتی وجود ندارد. یعنی می‌توانیم چندین کلاس داشته باشیم که هر کدام سرباری را به یک عملگر مفروض می‌افزاید. هیچ یک از این سربارها دیگری را نقض نمی‌کند و عملگر مربوطه با اشیای هر کلاس با توجه به سربار همان کلاس رفتار می‌کند. قبل از این که به سربارگذاری عملگرها بپردازیم، یک مفهوم جدید را در شی‌گرایی معرفی می‌کنیم.

10-2 توابع دوست

اعضایی از کلاس که به شکل خصوصی (private) اعلان می‌شوند فقط از داخل همان کلاس قابل دستیابی‌اند و از بیرون کلاس (درون بدنهٔ اصلی) امکان دسترسی به آن‌ها نیست. اما یک استثنا وجود دارد. ***تابع دوست***[1] تابعی است که عضو یک کلاس نیست اما اجازه دارد به اعضای خصوصی آن دسترسی داشته باشد. به کد زیر نگاه کنید:

```
class Ratio
{  friend int numReturn(Ratio);
   public:
      Ratio();
      ~Ratio();
   private:
      int num, den;
}

int numReturn(Ratio r)
{  return r.num;
}

int main()
{  Ratio x(22, 7);
   cout << numReturn(x) << endl;
}
```

1 – Friend function

در کد بالا تابع numReturn() عضو کلاس Ratio نیست بلکه دوست آن است. برای این که یک تابع را دوست یک کلاس معرفی کنیم، آن تابع را در کلاس مذکور اعلان کرده و از کلمهٔ کلیدی friend در اعلان آن استفاده می‌کنیم. توابع دوست باید قبل از اعضای عمومی و خصوصی کلاس اعلان شوند و تعریف آن‌ها باید خارج از کلاس و به شکل یک تابع معمولی باشد زیرا تابع دوست، عضو کلاس نیست. توابع دوست بیشتر در سربارگذاری عملگرها به کار گرفته می‌شوند.

10-3 سربارگذاری عملگر جایگزینی (=)

در بین عملگرهای گوناگون، عملگر جایگزینی شاید بیشترین کاربرد را داشته باشد. هدف این عملگر، کپی کردن یک شی در شیء دیگر است. مانند سازندهٔ پیش‌فرض، سازندهٔ کپی و نابودکننده، عملگر جایگزینی نیز به طور خودکار برای یک کلاس ایجاد می‌شود اما این تابع را می‌توانیم به شکل صریح درون کلاس اعلان نماییم.

× مثال 10-1 افزودن عملگر جایگزینی به کلاس Ratio

کد زیر یک رابط کلاس برای Ratio است که شامل سازندهٔ پیش‌فرض، سازندهٔ کپی و عملگر جایگزینی می‌باشد:

```
class Ratio
{  public:
      Ratio(int = 0, int = 1);         // default constructor
      Ratio(const Ratio&);             // copy constructor
      void operator=(const Ratio&);    // assignment operator
      // other declarations go here
   private:
      int num, den;
};
```

به نحو اعلان عملگر جایگزینی دقت نمایید. نام این تابع عضو، operator= است و فهرست آرگومان آن مانند سازندهٔ کپی می‌باشد یعنی یک آرگومان منفرد دارد که از نوع همان کلاس است که به طریقهٔ ارجاع ثابت ارسال می‌شود. عملگر جایگزینی را می‌توانیم به شکل زیر تعریف کنیم:

```
void Ratio::operator=(const Ratio& r)
{  num = r.num;
   den = r.den;
}
```

کد فوق اعضای داده‌ای شیء r را به درون اعضای داده‌ای شیئی که مالک فراخوانی این عملگر است، کپی می‌کند. حالا اگر x و y دو شی از کلاس Ratio باشند، کد x=y; با تعاریف بالا به درستی کار می‌کند ولی این هنوز کافی نیست.

10-4 اشاره‌گر this

در C++ می‌توانیم عملگر جایگزینی را به شکل زنجیره‌ای مثل زیر به کار ببریم:

```
x = y = z = 3.14;
```

اجرای کد بالا از راست به چپ صورت می‌گیرد. یعنی ابتدا مقدار 3.14 درون z قرار می‌گیرد و سپس مقدار z درون y کپی می‌شود و سرانجام مقدار y درون x قرار داده می‌شود. عملگر جایگزینی که در مثال قبل ذکر شد، نمی‌تواند به شکل زنجیره‌ای به کار رود. به برهان زیر توجه کنید:

فرض کنیم سه شیء x و y و z از یک کلاس باشند و در جایگزینی x=y=z شرکت کنند. جایگزینی مفروض را به شکل x=(y=z) می‌نویسیم. عبارت داخل پرانتز را می‌توانیم یک تابع تصور کنیم. پس در حقیقت x=f(y,z) است. مشخص است که تابع f باید دارای نوع بازگشتی از نوع x باشد. چون تابع f همان تابع عملگر جایگزینی است، نتیجه می‌شود که عملگر جایگزینی باید یک مقدار بازگشتی از نوع همان کلاس داشته باشد. اما این مقدار بازگشتی چیست؟ عملگر جایگزینی یک شی را درون شیء دیگر کپی می‌کند و چیزی برای بازگرداندن باقی نمی‌ماند.

اشاره‌گر this مساله را حل می‌کند. این اشاره‌گر مخفی، همیشه به شیئی اشاره می‌کند که الان روی آن عملی صورت گرفته است. مقدار بازگشتی از عملگر جایگزینی، همین اشاره‌گر this است که به شیئی که الان مقداری در آن کپی شده اشاره دارد. این مقدار می‌تواند در جایگزینی بعدی از زنجیره به کار گرفته شود. اکنون

می‌توانیم عملگر جایگزینی را به شکل کامل سربارگذاری کنیم. الگوی کلی برای سربارگذاری عملگر جایگزینی در کلاس مفروض T به شکل زیر است:

```
T& operator=(const T&);
```

همچنین الگوی کلی تعریف عملگر جایگزینی برای کلاس مفروض T به صورت زیر است:

```
T& T::operator=(const T& t)
{  // assign each member datum of t to the corresponding
   //  member datum of the owner
   return *this;
}
```

که به جای دو خط توضیحی در کد فوق، دستورات لازم و مورد نیاز قرار می‌گیرد. مثال بعدی عملگر جایگزینی سربارگذاری شدهٔ کامل برای کلاس Ratio را نشان می‌دهد.

× مثال 2-10 سربارگذاری عملگر جایگزینی به شکل صحیح

```
class Ratio
{  public:
      Ratio(int =0, int =1);            // default constructor
      Ratio(const Ratio&);              // copy constructor
      Ratio& operator=(const Ratio&);   // assignment operator
      // other declarations go here
   private:
      int num, den;
      // other declarations go here
};

Ratio& Ratio::operator=(const Ratio& r)
{  num = r.num;
   den = r.den;
   return *this;
}
```

حالا اشیای کلاس Ratio می‌توانند در یک جایگزینی زنجیره‌ای شرکت کنند:

```
Ratio x, y, z(22,7);
x = y = z;
```

توجه داشته باشید که عمل جایگزینی با عمل مقداردهی تفاوت دارد، هر چند هر دو از عملگر یکسانی استفاده می‌کنند. مثلا در کد زیر:

```
Ratio x(22,7);     // this is an initialization
Ratio y(x);        // this is an initialization
Ratio z = x;       // this is an initialization
Ratio w;
w = x;             // this is an assignment
```

سه دستور اول، دستورات مقداردهی هستند ولی دستور آخر یک دستور جایگزینی است. دستور مقداردهی، سازندهٔ کپی را فرا می‌خواند ولی دستور جایگزینی عملگر جایگزینی را فراخوانی می‌کند.

10-5 سربارگذاری عملگرهای حسابی

چهار عملگر حسابی + و - و * و / در همهٔ زبان‌های برنامه‌نویسی وجود دارند و با همهٔ انواع بنیادی به کار گرفته می‌شوند. قصد داریم سرباری را به این عملگرها اضافه کنیم تا بتوانیم با استفاده از آن‌ها، اشیای ساخت خودمان را در محاسبات ریاضی به کار ببریم.

عملگرهای حسابی به دو عملوند نیاز دارند. مثلا عملگر ضرب (*) در رابطهٔ زیر:

```
z = x*y;
```

با توجه به رابطهٔ فوق و آن‌چه در بخش قبلی گفتیم، عملگر ضرب سربارگذاری شده باید دو پارامتر از نوع یک کلاس و به طریق ارجاع ثابت بگیرد و یک مقدار بازگشتی از نوع همان کلاس داشته باشد. پس انتظار داریم قالب سربارگذاری عملگر ضرب برای کلاس Ratio به شکل زیر باشد:

```
Ratio operator*(Ratio x, Ratio y)
{  Ratio z(x.num*y.num, x.den*y.den);
   return z;
}
```

تابع فوق به دو پارامتر از نوع کلاس Ratio نیاز دارد. چنین تابعی نمی‌تواند عضوی از کلاس باشد. اگر بخواهیم تابعی عضو کلاس مفروض T باشد باید تابع مذکور حداکثر یک پارامتر از نوع کلاس T داشته باشد (تحقیق کنید که چرا چنین است). از طرفی اگر تابعی عضو کلاس نباشد، نمی‌تواند به اعضای خصوصی آن کلاس دستیابی کند. برای رفع این محدودیت‌ها، تابع سربارگذاری عملگر ضرب را باید به عنوان تابع دوست کلاس معرفی کنیم. لذا قالب کلی برای سربارگذاری عملگر ضرب درون کلاس مفروض T به شکل زیر است:

```
Class T
{  friend T operator*(const T&, const T&);
   public:
      // public members
   private:
      // private members
}
```

و از آن‌جا که تابع دوست عضوی از کلاس نیست، تعریف بدنهٔ آن باید خارج از کلاس صورت پذیرد. در تعریف بدنهٔ تابع دوست به کلمهٔ کلیدی friend نیازی نیست و عملگر جداسازی حوزه :: نیز استفاده نمی‌شود:

```
T operator*(const T& x, const T& y)
{  T z;
   // required operations for z = x*y
   return z;
}
```

در سربارگذاری عملگرهای حسابی + و - و / نیز از قالب‌های کلی فوق استفاده می‌کنیم با این تفاوت که در نام تابع سربارگذاری، به جای علامت ضرب * باید علامت عملگر مربوطه را قرار دهیم و دستورات بدنهٔ تابع را نیز طبق نیاز تغییر دهیم. مثال بعدی سربارگذاری عملگر ضرب را برای کلاس Ratio نشان می‌دهد.

× مثال 3-10 سربارگذاری عملگر ضرب برای کلاس Ratio

```
class Ratio
{     friend Ratio operator*(const Ratio&, const Ratio&);
```

```
   public:
      Ratio(int = 0, int = 1);
      Ratio(const Ratio&);
      Ratio& operator=(const Ratio&);
      // other declarations go here
   private:
      int num, den;
      // other declarations go here
};
Ratio operator*(const Ratio& x, const Ratio& y)
{  Ratio z(x.num * y.num , x.den * y.den);
   return z;
}

int main()
{  Ratio x(22,7) ,y(-3,8) ,z;
   z = x;               // assignment operator is called
   z.print();  cout << endl;
   x = y*z;            // multiplication operator is called
   x.print();  cout << endl;
}
```

```
22/7
-66/56
```

10-6 سربارگذاری عملگرهای جایگزینی حسابی

به خاطر بیاورید که عملگرهای جایگزینی حسابی، ترکیبی از عملگر جایگزینی و یک عملگر حسابی دیگر است. مثلا عملگر =* ترکیبی از دو عمل ضرب * و سپس جایگزینی = است. نکتهٔ قابل توجه در عملگرهای جایگزینی حسابی این است که این عملگرها بر خلاف عملگرهای حسابی ساده، فقط یک عملوند دارند. پس تابع سربارگذاری عملگرهای جایگزینی حسابی بر خلاف عملگرهای حسابی، می‌تواند عضو کلاس باشد. سربارگذاری عملگرهای جایگزینی حسابی بسیار شبیه سربارگذاری عملگر جایگزینی است. قالب کلی برای سربارگذاری عملگر =* برای کلاس مفروض T به صورت زیر است:

```
class T
{  public:
      T& operator*=(const T&);
      // other public members
   private:
      // private members
};
```

بدنهٔ تابع سربارگذاری به قالب زیر است:

```
T& T::operator*=(const T& x)
{  // required operations
   return *this;
}
```

استفاده از اشاره‌گر this* باعث می‌شود که بتوانیم عملگر =* را در یک رابطهٔ زنجیره‌ای به کار ببریم. در C++ چهار عملگر جایگزینی حسابی =+ و =- و =* و =/ وجود دارد. قالب کلی برای سربارگذاری همهٔ این عملگرها به شکل قالب بالا است فقط در نام تابع به جای =* باید علامت عملگر مربوطه را ذکر کرد و دستورات بدنهٔ تابع را نیز به تناسب، تغییر داد. مثال بعدی نشان می‌دهد که عملگر =* چگونه برای کلاس Ratio سربارگذاری شده است.

× مثال 4-10 کلاس Ratio با عملگر =* سربارگذاری شده

```
class Ratio
{  public:
      Ratio(int = 0, int = 1);
      Ratio& operator=(const Ratio&);
      Ratio& operator*=(const Ratio&);
      // other declarations go here
   private:
      int num, den;
      // other declarations go here
};
Ratio& Ratio::operator*=(const Ratio& r)
{  num = num*r.num;
```

```
   den = den*r.den;
   return *this;
}
```

بدیهی است که عملگر سربارگذاری شدهٔ جایگزینی حسابی باید با عملگر سربارگذاری شدهٔ حسابی معادلش، نتیجهٔ یکسانی داشته باشد. مثلا اگر x و y هر دو از کلاس Ratio باشند، آنگاه دو خط کد زیر باید نتیجهٔ مشابهی داشته باشند:

```
x *= y;
x = x*y;
```

10-7 سربارگذاری عملگرهای رابطه‌ای

شش عملگر رابطه‌ای در C++ وجود دارد که عبارتند از: < و > و <= و => و == و =! . این عملگرها به همان روش عملگرهای حسابی سربارگذاری می‌شوند، یعنی به شکل توابع دوست. اما نوع بازگشتی‌شان فرق می‌کند. حاصل عبارتی که شامل عملگر رابطه‌ای باشد، همواره یک مقدار بولین است. یعنی اگر آن عبارت درست باشد، حاصل true است و اگر آن عبارت نادرست باشد، حاصل false است. چون نوع بولین در حقیقت یک نوع عددی صحیح است، می‌توان به جای true مقدار 1 و به جای false مقدار 0 را قرار داد. به همین جهت نوع بازگشتی را برای توابع سربارگذاری عملگرهای رابطه‌ای، از نوع int قرار داده‌اند. قالب کلی برای سربارگذاری عملگر رابطه‌ای == به شکل زیر است:

```
class T
{  friend int operator==(const T&, const T&);
   public:
      // public members
   private:
      // private members
}
```

همچنین قالب کلی تعریف بدنهٔ این تابع به صورت زیر می‌باشد:

```
int operator==(const T& x, const T& y)
{  // required operations to finding result
```

```
   return result;
}
```

که به جای result یک مقدار بولین یا یک عدد صحیح قرار می‌گیرد. سایر عملگرهای رابطه‌ای نیز از قالب بالا پیروی می‌کنند.

× مثال 10-5 سربارگذاری عملگر تساوی (==) برای کلاس Ratio

```
class Ratio
{     friend int operator==(const Ratio&, const Ratio&);
      frined Ratio operator*(const Ratio&, const Ratio&);
      // other declarations go here
   public:
      Ratio(int = 0, int = 1);
      Ratio(const Ratio&);
      Ratio& operator=(const Ratio&);
      // other declarations go here
   private:
      int num, den;
      // other declarations go here
};
int operator==(const Ratio& x, const Ratio& y)
{  return (x.num * y.den == y.num * x.den);
}
```

چون اشیای کلاس Ratio به صورت کسر $\frac{a}{b}$ هستند، بررسی تساوی x==y معادل بررسی $\frac{a}{b} == \frac{c}{d}$ است که برای بررسی این تساوی می‌توانیم مقدار (a*d==b*c) را بررسی کنیم. بدنهٔ تابع سربارگذاری در مثال بالا همین رابطه را بررسی می‌کند.

10-8 سربارگذاری عملگرهای افزایشی و کاهشی

عملگر افزایشی ++ و کاهشی -- هر کدام دو شکل دارند: شکل پیشوندی و شکل پسوندی. هر کدام از این حالت‌ها را می‌توان سربارگذاری کرد. ابتدا سربارگذاری عملگر پیش‌افزایشی را بررسی می‌کنیم. عملگر پیش‌کاهشی به همین صورت سربارگذاری می‌شود.

وقتی که عملگر پیش‌افزایشی را همراه با یک شی به کار می‌بریم، یک واحد به مقدار آن شی افزوده می‌شود وسپس این مقدار جدید بازگشت داده می‌شود. پس تابعی که عمل سربارگذاری عملگر پیش‌افزایشی را انجام می‌دهد، مقداری از نوع همان کلاس را بازگشت می‌دهد اما هیچ پارامتری ندارد. این تابع، یک تابع عضو کلاس است. قالب کلی برای سربارگذاری عملگر پیش‌افزایشی به صورت زیر است:

```
class T
{  public:
      T operator++();
      // other public members
   private:
      // private members
};
```

قالب کلی برای سربارگذاری عملگر پیش‌افزایشی به شکل زیر است:

```
T T::operator++()
{  // required operations
   return *this;
}
```

می‌بینید که این‌جا هم از اشاره‌گر this* استفاده شده. علت هم این است که مشخص نیست چه چیزی باید بازگشت داده شود. به همین دلیل اشاره‌گر this* به کار رفته تا شیئی که عمل پیش‌افزایش روی آن صورت گرفته، بازگشت داده شود. توجه کنید که تعیین دستورات درون بدنه در اختیار ماست. لزوما این طور نیست که حتما مقدار 1 به مقدار فعلی شی افزوده شود. البته طبیعت این عملگر توصیه می‌کند که ضابطهٔ فوق را رعایت کنیم.

× مثال 6-10 افزودن عملگر پیش‌افزایشی به کلاس Ratio

اگر y یک شی از کلاس Ratio باشد و عبارت y++ ارزیابی گردد، مقدار 1 به y افزوده می‌شود اما چون y یک عدد کسری است، افزودن مقدار 1 به این کسر اثر متفاوتی دارد. فرض کنید y=22/7 باشد. حالا داریم:

$$++y = \frac{22}{7} + 1 = \frac{22+7}{7} = \frac{29}{7}$$

پس وقتی y افزایش می‌یابد، در اصل مقدار عضو داده‌ای num آن برابر با num+den خواهد شد:

```
class Ratio
{  public:
      Ratio(int n=0, int d=1) : num(n) , den(d) { }
      Ratio operator++();
      void print() {  cout << num << '/' << den << endl; }
   private:
      int num, den;
      // other declarations go here
};
int main()
{  Ratio x(22,7), y = ++x;
   cout << "y = ";  y.print();
   cout << ", x = ";  x.print();
}
Ratio Ratio::operator++()
{  num += den;
   return *this;
}
```

```
y = 29/7, x = 29/7
```

اکنون به بررسی عملگر پس‌افزایشی می‌پردازیم. نام تابع عملگر پیش‌افزایشی و پس‌افزایشی شبیه هم است. هر دو operator++ نامیده می‌شود. ولی می‌دانیم که این دو عملگر یکی نیستند. برای این که این دو تابع از هم تمیز داده شوند، فهرست پارامترهای تابع پس‌افزایشی یک پارامتر عجیب دارد. به قالب کلی برای تابع عملگر پس‌افزایشی توجه کنید:

```
T operator++(int);
```

تابع پس‌افزایشی یک پارامتر از نوع int دارد در حالی که می‌دانیم هیچ مقدار صحیحی به این تابع ارسال نمی‌شود. این پارامتر را در اصطلاح «پارامتر گنگ» می‌گویند. تنها هدف این پارامتر، تمیز دادن تابع عملگر پس‌افزایشی از تابع عملگر پیش‌افزایشی است. تابع عملگر پس‌افزایشی یک تفاوت مهم دیگر نیز دارد: دستورات

بدنهٔ آن. این دستورات باید طوری باشند که مقدار عملوند مربوطه را پس از انجام سایر محاسبات افزایش دهند نه پیش از آن. برای درک این موضوع به مثال بعدی توجه کنید.

× مثال 7-10 افزودن عملگر پس‌افزایشی به کلاس Ratio

در عبارت `y = x++;` از عملگر پس‌افزایشی استفاده کرده‌ایم. تابع این عملگر باید طوری تعریف شود که مقدار `x` را قبل از این که درون `y` قرار بگیرد، تغییر ندهد. می‌دانیم که اشاره‌گر `this*` به شیء جاری (مالک فراخوانی) اشاره دارد. کافی است مقدار این اشاره‌گر را در یک محل موقتی ذخیره کنیم و عمل افزایش را روی آن مقدار موقتی انجام داده و حاصل آن را بازگشت دهیم. به این ترتیب مقدار `this*` تغییری نمی‌کند و پس از شرکت در عمل جایگزینی، درون `y` قرار می‌گیرد:

```
class Ratio
{  public:
      Ratio(int n=0, int d=1) : num(n) , den(d) { }
      Ratio operator++();         //pre-increment
      Ratio operator++(int);      //post-increment
      void print() {  cout << num << '/' << den << endl; }
   private:
      int num, den;
};

int main()
{  Ratio x(22,7) , y = x++;
   cout << "y = ";  y.print();
   cout << ", x = ";  x.print();
}

Ratio Ratio::operator++(int)
{  Ratio temp = *this;
   num += den;
   return temp;
}
```

```
y = 22/7 , x = 29/7
```

پارامتر گنگ در تعریف تابع operator++() ذکر شده ولی هیچ نامی ندارد تا مشخص شود که این پارامتر فقط برای تمیز دادن تابع عملگر پس‌افزایشی آمده است.

عملگرهای پیش‌کاهشی و پس‌کاهشی نیز به همین شیوهٔ عملگرهای پیش‌افزایشی و پس‌افزایشی سربارگذاری می‌شوند. غیر از این‌ها، عملگرهای دیگری نیز مثل عملگر خروجی (<<) ، عملگر ورودی (>>) ، عملگر اندیس ([]) و عملگر تبدیل نیز وجود دارند که می‌توان آن‌ها را برای سازگاری برای کلاس‌های جدید سربارگذاری کرد. چون این عملگرها کاربرد کمتری دارند از توضیح آن‌ها خودداری می‌کنیم. اکنون که اصول سربارگذاری عملگرها را آموختید، می‌توانید با مراجعه به مراجع C++ سربارگذاری سایر عملگرها را تحقیق کنید.

پرسش‌های گزینه‌ای

1 – اگر تابعی دوست یک کلاس باشد، آنگاه:

الف – آن تابع یک عضو عمومی آن کلاس است

ب – آن تابع یک عضو خصوصی آن کلاس است

ج – آن تابع یک عضو محافظت‌شدهٔ آن کلاس است

د – آن تابع اصلا عضوی از کلاس نیست

2 – کدام گزینه در مورد توابع دوست کلاس، صحیح است؟

الف – توابع دوست کلاس فقط اجازه دارند به اعضای عمومی کلاس دستیابی کنند

ب – توابع دوست کلاس از اعضای خصوصی کلاس محسوب می‌شوند

ج – در تعریف توابع دوست کلاس نباید از عملگر جداسازی دامنه `::` استفاده شود

د – هر کلاس فقط می‌تواند یک تابع دوست داشته باشد

3 – اشاره گر `this*` چیست؟

الف – یک اشاره‌گر مخفی است که به شیئی اشاره دارد که هم‌اکنون عملی روی آن صورت گرفته

ب – یک اشاره‌گر مخفی است که به کلاسی اشاره دارد که هم‌اکنون نمونه‌سازی شده

ج – یک اشاره‌گر مخفی است که به تابعی اشاره دارد که هم‌اکنون فراخوانی شده

د – یک اشاره‌گر مخفی است که به عملگری اشاره دارد که هم‌اکنون اجرا شده

4 – کدام گزینه در رابطه با سربارگذاری عملگر جایگزینی صحیح نیست؟

الف – نام تابع آن، `operator==()` است

ب – تابع آن، عضوی از کلاس است

ج – اگر به طور صریح ذکر نشود، آنگاه یک نسخهٔ پیش‌فرض به کلاس اضافه می‌شود

د – نوع بازگشتی آن به شکل یک ارجاع است

5 – برای این که بتوانیم اشیای یک کلاس را به شکل زنجیره‌ای در یک عمل حسابی شرکت دهیم باید:

الف – تابع سربارگذاری آن عمل را به شکل عضوی از کلاس اعلان کنیم

ب – پارامتر تابع سربارگذاری آن عمل را به شکل ارجاع ثابت اعلان کنیم

ج – نوع بازگشتی تابع سربارگذاری آن عمل را به شکل ارجاع و از نوع همان کلاس اعلان کنیم

د – بدنهٔ تابع سربارگذاری آن عمل را در فایل جداگانه‌ای قرار دهیم

6 – تابع سربارگذاری عملگر پیش‌افزایشی باید:

الف – عضو عمومی کلاس باشد　　ب – عضو خصوصی کلاس باشد

ج – دوست کلاس باشد　　د – به شکل تابع سودمند محلی باشد

7 – فرض کنید که کلاس مفروض Date موجود باشد. در این صورت تابع operator<(const Date&, const Date&) باید:

الف – عضو عمومی کلاس باشد　　ب – عضو خصوصی کلاس باشد

ج – دوست کلاس باشد　　د – تابع دستیابی باشد

8 – فرض کنید که کلاس مفروض Person موجود باشد. در این صورت تابع Person operator--(int); چه چیزی را سربارگذاری می‌کند؟

الف – عملگر پیش‌کاهشی را سربارگذاری می‌کند

ب – عملگر پس‌کاهشی را سربارگذاری می‌کند

ج – عملگر ایندکس را سربارگذاری می‌کند

د – عملگر خروجی را سربارگذاری می‌کند

9 – تفاوت بین عملگرهای سربارگذاری شدهٔ پیش‌افزایشی و پس‌افزایشی در چیست؟

الف – تابع سربارگذاری عملگر پیش‌افزایشی، عضو کلاس است ولی تابع سربارگذاری عملگر پس‌افزایشی عضو کلاس نیست

ب – تابع سربارگذاری عملگر پیش‌افزایشی دوست کلاس است ولی تابع سربارگذاری عملگر پس‌افزایشی دوست کلاس نیست

ج – تابع سربارگذاری عملگر پس‌افزایشی پارامتر گنگ دارد ولی تابع سربارگذاری عملگر پیش‌افزایشی پارامتر گنگ ندارد

د – تابع سربارگذاری عملگر پس‌افزایشی اشاره‌گر this* دارد ولی تابع سربارگذاری عملگر پیش‌افزایشی اشاره‌گر this* ندارد

10 – نوع بازگشتی از تابع سربارگذاری عملگرهای رابطه‌ای چیست؟

الف – int　　ب – char　　ج – double　　د – float

پرسش‌های تشریحی

1- کلمهٔ کلیدی operator چگونه استفاده می‌شود؟

2- اشاره‌گر this* به چه چیزی اشاره می‌کند؟

3- چرا اشاره‌گر this* را نمی‌توان برای توابع غیر عضو استفاده کرد؟

4- چرا عملگر جایگزینی سربازگذاری شده، this* را برمی‌گرداند؟

5- نتیجهٔ اعلان‌های زیر چه تفاوتی با هم دارد؟

```
Ratio y(x);
Ratio y = x;
```

6- نتیجهٔ دو سطر زیر چه تفاوتی با هم دارد؟

```
Ratio y = x;
Ratio y;   y = x;
```

7- چرا نمی‌توان ** را به عنوان عملگر توان سربارگذاری کرد؟

8- چرا عملگرهای حسابی + و - و * و / باید به صورت توابع دوست سربارگذاری شوند؟

10- چگونه تعریف عملگر پیش‌افزایشی سربارگذاری شده از عملگر پس‌افزایشی سربارگذاری شده تشخیص داده می‌شود؟

11- چرا آرگومان int در پیاده‌سازی عملگر پیش‌افزایشی بدون نام است؟

تمرین‌های برنامه‌نویسی

1- کلاس Vector را با یک سازندهٔ پیش‌فرض، یک سازندهٔ کپی، یک نابودکننده و یک عملگر جایگزینی سربارگذاری شده و عملگر تساوی سربارگذاری شده پیاده‌سازی کنید. هر شی از این کلاس، یک بردار را نشان می‌دهد.

2- عملگرهای تقسیم و جمع را برای کلاس Ratio پیاده‌سازی کنید.

3- عملگرهای حسابی پیش‌کاهشی و پس‌کاهشی را برای کلاس Ratio سربارگذاری کنید.

4- عملگرهای جایگزینی حسابی =/ و =- را برای کلاس Ratio سربارگذاری کنید. 5- عملگرهای رابطه‌ای < و =! را برای کلاس Ratio سربارگذاری کنید.

6- تحقیق کنید که غیر از عملگرهای ذکر شده در این فصل، کدام عملگرها را می‌توان در C++ برای یک کلاس سربارگذاری کرد. به دلخواه یکی از آن عملگرها را برای کلاس Ratio سربارگذاری کنید.

7- برای کلاس Time (مسألهٔ 3 فصل 9) عملگرهای حسابی + و - و همچنین عملگرهای رابطه ای == و > و < را سربارگذاری کنید.

8 – برای کلاس Matrix (مسألهٔ 6 فصل 9) عملگرهای حسابی - و * را سربارگذاری کنید.

9 – برای کلاس Point (مسألهٔ 7 فصل 9) عملگرهای حسابی + و / را سربارگذاری کنید.

فصل یازدهم

«ترکیب و وراثت»

12-1 مقدمه

اغلب اوقات برای ایجاد یک کلاس جدید، نیازی نیست که همه چیز از اول طراحی شود. می‌توانیم برای ایجاد کلاس مورد نظر، از تعاریف کلاس‌هایی که قبلا ساخته‌ایم، استفاده نماییم. این باعث صرفه‌جویی در وقت و استحکام منطق برنامه می‌شود. در شی‌گرایی به دو شیوه می‌توان این کار را انجام داد: ***ترکیب***[1] و ***وراثت***[2]. در این فصل خواهیم دید که چگونه و چه مواقعی می‌توانیم از این دو شیوه بهره ببریم.

12-2 ترکیب

ترکیب کلاس‌ها (یا تجمیع کلاس‌ها) یعنی استفاده از یک یا چند کلاس دیگر در داخل تعریف یک کلاس جدید. هنگامی که عضو داده‌ای کلاس جدید، شیئی از کلاس دیگر باشد، می‌گوییم که این کلاس جدید ترکیبی از سایر کلاس‌هاست. به تعریف دو کلاس زیر نگاه کنید.

---

1 – Composition

2 - Inheritance

× مثال 1-11 کلاس Date

کد زیر، کلاس Date را نشان می‌دهد که اشیای این کلاس برای نگهداری تاریخ استفاده می‌شوند.

```
class Date
{  public:
      Date(int y=0, int m=0, int d=0) :
           year(y), month(m), day(d) {};
      void setDate(int y, int m, int d)
           {  year = y; month = m; day = d;  }
      void getDate()
           {  cin >> year >> month >> day ;  }
      void showDate()
           {  cout << year << '/' << month << '/' << day ;  }
   private:
      int year, month, day;
}

int main()
{  Date memory(1359,6,31);
   cout << "SADDAM attacked to IRAN at ";
   memory.showDate();
   memory.setDate(1367,4,27);
   cout << "\nThat war finished at ";
   memory.showDate();
   cout << "\nEnter your birthday: ";
   memory.getDate();
   cout << "\nYour birthday is ";
   memory.showDate();
}
```

```
SADDAM attacked to IRAN at 1359/6/31
That war finished at 1367/4/27
Enter your birthday: 1358 6 15
Your birthday is 1358/6/15
```

× مثال 11-2 کلاس Book

کد زیر، کلاس Book را نشان می‌دهد که اشیای این کلاس برخی از مشخصات یک کتاب را نگهداری می‌کنند:

```
class Book
{  public:
      Book(char* n = " ", int i = 0, int p = 0) :
         name(n), id(i), page(p) { }
      void printName() {  cout << name;  }
      void printId()   {  cout << id;    }
      void printPage() {  cout << page;  }
   private:
      string name, author;
      int id, page;
}

int main()
{  Book reference("C++", 1, 450);
   cout << "  reference    id     pages  " << endl;
   cout << "-----------------------------" << endl;
   cout << "    " ;    reference.printName();
   cout << "        " ; reference.printId();
   cout << "       "  ; reference.printPage();
}
```

```
  reference    id     pages
-----------------------------
    C++        1       450
```

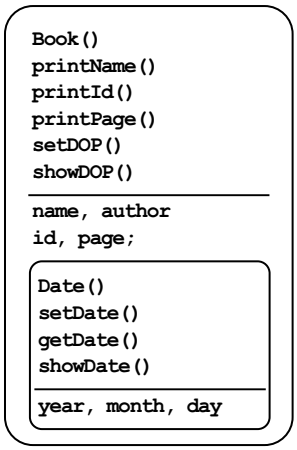
اکنون می‌خواهیم با ترکیب این دو کلاس، امکانات کلاس Book را بهبود دهیم تا مشخصات یک کتاب و تاریخ انتشار آن را نگهداری کند.

× مثال 11-3 بهبود دادن کلاس Book

```
#include "Date.h"

class Book
{  public:
      Book(char* n = " ", int i = 0, int p = 0) :
```

```
        name(n), id(i), page(p) { }
    void printName() {  cout << name;  }
    void printId()   {  cout << id;    }
    void printPage() {  cout << page;  }
    void setDOP(int y, int m, int d)
                    { publish.setDate(y, m, d) ; }
    void showDOP() { publish.showDate(); }
  private:
    string name, author;
    int id, page;
    Date publish;
}
```

با دقت در این کلاس بهبود یافته، می‌بینیم که یک عضو داده‌ای خصوصی از نوع کلاس Date درون اعضای آن وجود دارد. همچنین دو تابع عضو عمومی جدید setDOP() و showDOP() نیز به این کلاس افزوده شده. به بدنهٔ این دو تابع نگاه کنید. دو تابع دیگر به نام‌های setDate() و showDate() در بدنهٔ این دو تابع قرار دارند. این دو تابع، عضو کلاس Date هستند.



چون publish از نوع Date است، می‌توانیم توابع عضو کلاس Date را برای آن صدا بزنیم. توجه کنید که تعاریف کلاس Date را در فایل جداگانه‌ای به نام "Date.h" گذاشته‌ایم و برای ترکیب کردن این کلاس با کلاس Book، فایل مذکور را include کرده‌ایم. کلاس Book را می‌توانیم به شکل مقابل مجسم کنیم. کلاس Date به عنوان یک عضو داده‌ای در بخش اعضای داده‌ای کلاس Book قرار گرفته است.

حالا می‌توانیم این کلاس مرکب را در برنامه استفاده کنیم:

```
int main()
{  Book reference("C++", 1, 450);
   reference.setDOP(1384, 6, 15);
```

```
   cout << "  reference    id     pages      DOP  " << endl;
   cout << "--------------------------------------" << endl;
   cout << "    " ;    reference.printName();
   cout << "       "; reference.printId();
   cout << "      " ; reference.printPage();
   cout << "    " ;    reference.showDOP();
}
```

```
  reference    id     pages      DOP
--------------------------------------
    C++        1       450    1384/6/15
```

11-3 وراثت

وراثت روش دیگری برای ایجاد کلاس جدید از روی کلاس قبلی است. گاهی به وراثت «اشتقاق» نیز می‌گویند. اگر از قبل با برنامه‌نویسی مبتنی بر پنجره‌ها آشنایی مختصر داشته باشید، احتمالا عبارت «کلاس مشتق‌شده» را فراوان دیده‌اید. این موضوع به خوبی اهمیت وراثت را آشکار می‌نماید.

در دنیای واقعی، وراثت به این معناست که هر فرزندی خواص عمومی را از والد خود به ارث می‌گیرد، مثل رنگ چشم یا رنگ مو یا چهرهٔ ظاهری یا رفتارهای غریزی و ... . در دنیای نرم‌افزار نیز وراثت به همین معناست. یعنی کلاس جدید را طوری تعریف کنیم که اعضای کلاس دیگر را به کار بگیرد. فرض کنید کلاس x قبلا تعریف شده و وجود دارد. نحو کلی برای اشتقاق کلاس y از کلاس x به شکل زیر است:

```
class Y : public X
{
   //  ...
};
```

در این حالت می‌گوییم که کلاس y از کلاس x مشتق شده است. عبارت public بعد از علامت کولن برای تاکید بر این معناست که کلاس y فقط به اعضای عمومی کلاس x دسترسی دارد. به کلاس x ***کلاس اصلی***[1] یا «کلاس والد[2]» یا «کلاس پایه» می‌گوییم و به کلاس y ***زیرکلاس***[3] یا «کلاس فرزند[4]» یا «کلاس مشتق‌شده» می‌گوییم.

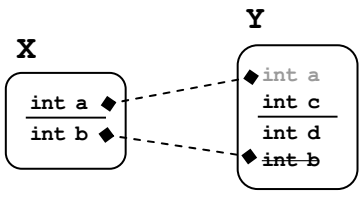
---

1 – Base class
2 – Parent class
3 – Sub class
4 – Child class

کلاس فرزند می‌تواند علاوه بر اعضای به ارث برده شده، اعضای عمومی و خصوصی خاص خود را نیز داشته باشد. به کد زیر دقت کنید:

```
class X
{  public:
      int a;
   private:
      int b;
};
class Y : public X
{  public:
      int c;
   private:
      int d;
}
```

کلاس X یک عضو عمومی به نام a و یک عضو خصوصی به نام b دارد. کلاس Y به عنوان فرزند کلاس X اعلان شده. پس عضو عمومی کلاس x را به ارث می‌برد. یعنی کلاس Y دارای a هست. همچنین یک عضو عمومی مخصوص به خود به نام c هم دارد. چون b یک عضو خصوصی از کلاس والد یعنی X است، نمی‌تواند توسط کلاس فرزند یعنی Y مستقیما دستیابی شود. بنابراین کلاس Y فقط یک عضو خصوصی به نام d دارد. کلاس‌های X و Y را می‌توان به شکل مقابل تصور نمود. تصویر نشان می‌دهد که کلاس Y عضو b را هم دارد ولی اجازهٔ دسترسی مستقیم به آن را ندارد. به مثال زیر توجه کنید.



× مثال 4-11 مشتق کردن کلاس Ebook از کلاس Book

می‌خواهیم کلاسی برای نگهداری مشخصات کتاب‌های الکترونیکی بسازیم. کتاب‌های الکترونیکی زیرمجموعه‌ای از کتاب‌های کلی هستند. پس منطقی است به جای این که کلاس Ebook را از ابتدا ایجاد کنیم، آن را از کلاس Book که قبلا ایجاد کردیم، مشتق کنیم. در این صورت اغلب توابع و متغیرهای مورد نیاز از کلاس Book

به ارث گرفته می‌شوند و نیازی به ایجاد آن‌ها نیست. فقط کافی است توابع و داده‌های خاص کلاس Ebook را به کلاس مشتق شده اضافه کنیم:

```
#include "Book.h"
class Ebook : public Book
{  public:
      Ebook(char* n, int i=0, int p=0) :
            Book(n,i,p), format("PDF"), size(0) {}
      void printFormat() {  cout << format; }
      void setSize(int s){  size = s; }
      void setFormat(char* f) {  format = f; }
      void printSize()  {  cout << size; }
   private:
      string format;
      int size;
}

int main()
{  Ebook reference("C++", 1, 450);
   reference.setSize(5500);
   cout << "  Ebook   id    pages   format   size(KB) \n";
   cout << "---------------------------------------- \n";
   cout << "   " ;  reference.printName();
   cout << "     "; reference.printId();
   cout << "     "; reference.printPage();
   cout << "     "; reference.printFormat();
   cout << "      "; reference.printSize();
}
```

```
  Ebook   id    pages   format   size(KB)
----------------------------------------
   C++     1     450      PDF      5500
```

گرچه توابع printName() و printId() و printPage() در کلاس Ebook اعلان و تعریف نشده‌اند، اما چون این کلاس از کلاس Book مشتق شده، همهٔ اعضای عمومی آن را به ارث می‌گیرد و دیگر نیازی به اعلان آن‌ها نیست. این مثال

نشان می‌دهد که با استفاده از اعضای عمومی به ارث گرفته شده از کلاس والد، می‌توان به اعضای خصوصی آن دسترسی داشت. دقت کنید که چگونه در تابع سازندهٔ کلاس Ebook از تابع سازندهٔ کلاس Book استفاده کرده و به اعضای خصوصی کلاس Book دستیابی نموده‌ایم.

11-4 اعضای حفاظت شده

گرچه کلاس Ebook در مثال قبل نمی‌تواند مستقیما به اعضای خصوصی کلاس والدش دسترسی داشته باشد، اما با استفاده از توابع عضو عمومی که از کلاس والد به ارث برده، می‌تواند به اعضای خصوصی آن کلاس دستیابی کند. این محدودیت بزرگی محسوب می‌شود. اگر توابع عضو عمومی کلاس والد انتظارات کلاس فرزند را برآورده نسازند، کلاس فرزند ناکارآمد می‌شود. اوضاع زمانی وخیم‌تر می‌شود که هیچ تابع عمومی برای دسترسی به یک دادهٔ خصوصی در کلاس والد وجود نداشته باشد.

در کلاس Book مثال 11-2 یک دادهٔ خصوصی به نام author داریم که بیان‌گر نام نویسنده است. در کلاس فرزند Ebook به هیچ ترتیبی نمی‌توانیم به این عضو دسترسی پیدا کنیم. می‌توانیم این دادهٔ عضو را به شکل صریح در کلاس Ebook بگنجانیم ولی این کار منطقی نیست زیرا Ebook زیرمجموعه‌ای از Book است و باید بتواند خواص آن را به کار بگیرد. در چنین مواقعی C++ توصیه می‌کند به جای این که author را در کلاس والد به صورت خصوصی (private) اعلان کنیم، آن را به شکل حفاظت شده (protected) اعلان نماییم. اعضای protected یا ***حفاظت شده***[1] اعضایی هستند که مثل اعضای خصوصی از خارج کلاس قابل دستیابی نیستند ولی مثل اعضای عمومی می‌توانند توسط کلاس‌های فرزند مستقیما دستیابی شوند. به مثال زیر توجه کنید.

× مثال 11-5 کلاس Book با اعضای داده‌ای protected

کدهای زیر همان کدهای مثال 11-2 هستند با این تفاوت که اعضای private کلاس Book به اعضای protected تبدیل شده‌اند و توابع دستیابی

1 – Protected members

setAuthor() و printAuthor() به کلاس مشتق شدهٔ Ebook افزوده شده:

```
class Book
{  public:
      Book(char* n = " ", int i = 0, int p = 0) :
         name(n), id(i), page(p) { }
      void printName() {  cout << name;  }
      void printId()   {  cout << id;    }
      void printPage() {  cout << page;  }
   protected:
      string name, author;
      int id, page;
}
class Ebook : public Book
{  public:
      Ebook(char* n, int i=0, int p=0) :
            Book(n,i,p), format("PDF"), size(0) {}
      void setAuthor(char* a) {  author = a; }
      void setSize(int s){  size = s; }
      void setFormat(char* f) {  format = f; }
      void printAuthor() {  cout << author;  }
      void printFormat() {  cout << format;  }
      void printSize()   {  cout << size;    }
   protected:
      string format;
      int size;
}
```

حالا همهٔ اعضای حفاظت شدهٔ کلاس Book از درون زیرکلاس Ebook قابل دستیابی هستند:

```
int main()
{  Ebook reference("C++", 1, 450);
   reference.setSize(5500);
   reference.setAuthor("P.N.U");
   cout << "\n Ebook name: ";   reference.printName();
   cout << "\n         id: ";   reference.printId();
```

```
   cout << "\n      pages: ";   reference.printPage();
   cout << "\n     format: ";   reference.printFormat();
   cout << "\n   size(KB): ";   reference.printSize();
   cout << "\n     Author: ";   reference.printAuthor();
}
```

```
Ebook name: C++
        id: 1
     pages: 450
    format: PDF
  size(KB): 5500
    Author: P.N.U
```

بنابر آن‌چه گفتیم، اعضای یک کلاس سه گروه هستند: اعضای عمومی یا public که از بیرون کلاس و از درون کلاس‌های فرزند مستقیما قابل دستیابی‌اند؛ اعضای خصوصی یا private که فقط از درون همان کلاس یا توابع دوست کلاس می‌توان به آن‌ها دسترسی داشت؛ اعضای حفاظت شده یا protected که از بیرون کلاس قابل دستیابی نیستند ولی از درون همان کلاس یا درون کلاس‌های فرزند یا توابع دوست کلاس می‌توان مستقیما به آن‌ها دسترسی داشت. پس قالب کلی تعریف یک کلاس به شکل زیر است:

```
class X
{  public:
      // public members
   protected:
      // protected members
   private:
      // private members
};
```

در نظر داشته باشید که اعضای حفاظت شده از کلاس والد، در کلاس فرزند نیز به عنوان اعضای protected منظور می‌شوند.

ممکن است بپرسید با خواصی که اعضای protected دارند، چه نیازی به اعضای private است؟ برای پاسخ به این سوال یادآوری می‌کنیم که از مزیت‌های مهم شی‌گرایی، مخفی‌سازی اطلاعات است. همیشه این طور نیست که بخواهیم همهٔ

اعضای کلاس والد در کلاس فرزند به ارث گذاشته شوند. ممکن است در کلاس Book یک عضو داده‌ای با نام paperkind بگذاریم که نوع کاغذ مصرفی در کتاب را نشان می‌دهد. طبیعی است که چنین عضوی در کلاس Ebook اصلا بی‌معناست. بنابراین لازم نیست (و نباید) که کلاس Ebook عضو داده‌ای paperkind را به ارث بگیرد و این عضو باید از دسترس دور بماند. عقیدهٔ کلی بر این است که هر عضو داده‌ای کلاس را به شکل private اعلان کنید مگر این که احتمال بدهید آن عضو ممکن است لازم باشد در کلاس فرزند به ارث گرفته شود. آنگاه چنین عضوی را به شکل protected اعلان نمایید.

11-5 غلبه کردن بر وراثت

اگر Y زیر کلاسی از X باشد، آنگاه اشیای Y همهٔ اعضای عمومی و حفاظت شدهٔ کلاس X را ارث می‌برند. مثلا تمامی اشیای Ebook تابع دستیابی printName() از کلاس Book را به ارث می‌برند. به تابع printName() یک «عضو موروثی» می‌گوییم. گاهی لازم است یک نسخهٔ محلی از عضو موروثی داشته باشیم. یعنی کلاس فرزند، عضوی هم نام با عضو موروثی داشته باشد که مخصوص به خودش باشد و ارثی نباشد. برای مثال فرض کنید کلاس X یک عضو عمومی به نام p داشته باشد و کلاس Y زیر کلاس X باشد. در این حالت اشیای کلاس Y عضو موروثی p را خواهند داشت. حال اگر یک عضو به همان نام p در زیرکلاس Y به شکل صریح اعلان کنیم، این عضو جدید، عضو موروثی هم‌نامش را مغلوب می‌کند. به این عضو جدید، «عضو غالب» می‌گوییم. بنابراین اگر y1 یک شی از کلاس Y باشد، y1.p به عضو p غالب اشاره دارد نه به p موروثی. البته هنوز هم می‌توان به p موروثی دسترسی داشت. عبارت y1.X::p به p موروثی دستیابی دارد.

هم می‌توان اعضای داده‌ای موروثی را مغلوب کرد و هم اعضای تابعی موروثی را. یعنی اگر کلاس X دارای یک عضو تابعی عمومی به نام f() باشد و در زیرکلاس Y نیز تابع f() را به شکل صریح اعلان کنیم، آنگاه y1.f() به تابع غالب اشاره دارد و y1.X::f() به تابع موروثی اشاره دارد. در برخی از مراجع به توابع غالب

override می‌گویند و داده‌های غالب را dominate می‌نامند. ما در این کتاب هر دو مفهوم را به عنوان اعضای غالب به کار می‌بریم. به مثال زیر نگاه کنید.

× مثال 6-11 اعضای داده‌ای و تابعی غالب

```
class X
{  public:
      void f() { cout << "Now X::f() is running\n"; }
      int a;
};

class Y : public X
{  public:
      void f() { cout << "Now Y::f() is running\n"; }
      // this f() overrides X::f()
      int a;
      // this a dominates X::a
};
```

اعضای زیرکلاس Y همان مشخصات اعضای کلاس X را دارند. به همین دلیل تابع f() که در زیرکلاس Y اعلان شده، تابع f() موروثی (X::f()) را مغلوب می‌کند و عضو a که در زیرکلاس Y اعلان شده نیز عضو a موروثی (X::a) را مغلوب می‌نماید. برنامهٔ آزمون زیر این مطلب را تایید می‌کند:

```
int main()
{  X x1;
   x1.a = 22;
   x1.f();
   cout << "x1.a = " << x1.a << endl;
   Y y1;
   y1.a = 44;            // assigns 44 to the a defined in Y
   y1.X::a = 66;         // assigns 66 to the a defined in X
   y1.f();               // calls the f() defined in Y
   y1.X::f();            // calls the f() defined in X
   cout << "y1.a = " << y1.a << endl;
   cout << " y1.X::a = " << y1.X::a << endl;
```

```
    X x2 = Y1;
    cout << "x2.a = " << x2.a << endl;
}
```

```
X::f() is running
x1.a = 22
Y::f() is running
X::f() is running
y1.a = 44
y1.X::a = 66
x2.a = 66
```

می‌بینید که شیء y1 دو عضو داده‌ای مجزای a و دو عضو تابعی مجزای f() دارد. به طور پیش‌فرض، آن‌هایی که غالب هستند دستیابی می‌شوند ولی با استفاده از عملگر جداسازی حوزه :: می‌توان به اعضای موروثی نیز دسترسی داشت.

در دستور جایگزینی x2 = y1; نکتهٔ مهمی وجود دارد. x2 شیئی از کلاس والد است و y1 شیئی از کلاس فرزند. وقتی قرار باشد یک شی از کلاس فرزند درون یک شی از کلاس والد کپی شود، به طور پیش‌فرض اعضای موروثی کلاس فرزند برای جایگزینی انتخاب می‌شوند نه اعضای غالب. یعنی در چنین حالتی y1.X::a در x2.a کپی می‌شود نه y1.a . (سعی کنید علت را توضیح دهید)

در برنامه‌های واقعی، از توابع غالب بسیار استفاده می‌شود اما استفاده از داده‌های غالب چندان مرسوم نیست. اغلب این طور است که نوع یک عضو داده‌ای در کلاس فرزند با نوع همان عضو در کلاس والد متفاوت است. برای مثال اگر عضو داده‌ای a در کلاس والد از نوع int باشد، آنگاه عضو داده‌ای صریح a در کلاس فرزند از نوع double اعلان می‌شود.

سازنده‌های پیش‌فرض و نابودگرها در وراثت رفتار متفاوتی دارند. به این صورت که سازندهٔ کلاس فرزند قبل از اجرای خودش، سازندهٔ کلاس والدش را فرا می‌خواند. نابودگر کلاس فرزند نیز پس از اجرای خودش، نابودگر کلاس والدش را فراخوانی می‌کند. به مثال زیر توجه نمایید.

× مثال 7-11 سازنده‌ها و نابودکننده‌های والد

```
class X
{  public:
      X() { cout << "X::X() constructor executing\n"; }
      ~X() { cout << "X::X() destructor executing\n"; }
};

clas Y : public X
{  public:
      Y() { cout << "Y::Y() constructor executing\n"; }
      ~Y() { cout << "Y::Y() destructor executing\n"; }
};
clas Z : public Y
{  public:
     Z(int n) {cout << "Z::Z(int) constructor executing\n";}
     ~Z() { cout << "Z::Z() destructor executing\n"; }
};

int main()
{  Z z(44);
}
```

```
X::X() constructor executing
Y::Y() constructor executing
Z::Z(int) constructor executing
Z::Z() destructor executing
Y::Y() destructor executing
X::X() destructor executing
```

در برنامهٔ بالا کلاس Z فرزند کلاس Y است که خود Y فرزند کلاس X می‌باشد. در بدنهٔ تابع main() یک شی از نوع کلاس Z نمونه‌سازی شده. پس انتظار این است که سازندهٔ کلاس Z فراخوانی شود. اما تصویر خروجی نیز نشان می‌دهد که قبل از فراخوانی سازندهٔ Z، تابع سازندهٔ کلاس والدش یعنی سازندهٔ کلاس Y فراخوانی می‌شود. چون Y فرزند X است، پس قبل از فراخوانی سازندهٔ کلاس Y، سازندهٔ کلاس X فراخوانی می‌شود. لذا هنگام ایجاد یک شی از کلاس Z ابتدا سازندهٔ کلاس X و سپس سازندهٔ کلاس Y و در نهایت سازندهٔ کلاس Z فراخوانی می‌شود.

وقتی به پایان تابع main() برسیم، شیء Z به پایان عمرش می‌رسد. پس باید نابودگر کلاس Z فراخوانی شود. همین اتفاق هم می‌افتد اما پس از فراخوانی نابودگر کلاس Z، نابودگر کلاس والدش یعنی Y هم فراخوانی می‌شود. پس از فراخوانی نابودگر کلاس Y، نابودگر کلاس X نیز که والد Y است فراخوانی می‌شود. دقت کنید که سازنده‌های والد از برترین کلاس به زیرترین کلاس اجرا می‌شوند ولی نابودگرها از زیرترین کلاس به سمت برترین کلاس اجرا می‌شوند. اما چرا چنین اتفاقی می‌افتد؟

اگر کلاس Y فرزند کلاس X باشد، اعضای کلاس X را به ارث می‌گیرد. سازندهٔ کلاس Y راجع به اعضای صریح این کلاس آگاهی دارد ولی از اعضای کلاس والد چیزی نمی‌داند. بنابراین هنگام اجرا می‌داند برای اعضای صریح کلاس Y چه میزان حافظه منظور کند اما نمی‌داند برای اعضای موروثی چقدر حافظه نیاز است. از کجا می‌توان میزان حافظهٔ مورد نیاز اعضای موروثی را فهمید؟ از سازندهٔ والد. به همین دلیل است که هر سازنده قبل از اجرای خودش، سازندهٔ والدش را فرا می‌خواند. سازندهٔ والد حافظهٔ مورد نیاز برای اعضای والد را تخصیص داده و این حافظه را به سازندهٔ فرزند تحویل می‌دهد. سازندهٔ فرزند نیز حافظهٔ لازم برای اعضای صریح را تخصیص داده و به میزان قبلی می‌افزاید و به این ترتیب شیء فرزند با حافظهٔ کافی متولد می‌شود. هنگام نابود کردن شیء فرزند نیز ابتدا نابودگر کلاس فرزند حافظهٔ تخصیصی به اعضای صریح را آزاد می‌کند اما حافظهٔ تخصیص یافته به اعضای موروثی بلاتکلیف باقی می‌مانند. به همین دلیل لازم است نابودکنندهٔ کلاس والد نیز فراخوانی شود تا این حافظهٔ اضافی آزاد شود.

11-6 اشاره‌گرها در وراثت

در شی‌گرایی خاصیت جالبی وجود دارد و آن این است که اگر p اشاره‌گری از نوع کلاس والد باشد، آنگاه p را می‌توان به هر فرزندی از آن کلاس نیز اشاره داد. به کد زیر نگاه کنید:

```
class X
{  public:
      void f();
```

```
}
class Y : public X            // Y is a subclass of X
{  public:
      void f();
}
int main()
{  X* p;         // p is a pointer to objects of base class X
   Y y;
   p = &y;       // p can also point to objects of subclass Y
}
```

گرچه p از نوع X* است، اما آن را می‌توان به اشیای کلاس Y که فرزند X است نیز اشاره داد. یادآوری می‌کنیم که p->f() معادل (*p).f() است. حالا در کد فوق تصور کنید p که از نوع X* است به یک شیء Y اشاره کند و p->f() فراخوانی شود. اکنون کدام f() فراخوانی می‌شود؟ X::f() یا Y::f() ؟ انتظار داریم که Y::f() فراخوانی شود ولی چنین نیست. p از نوع X* تعریف شده است و فقط X::f() را می‌شناسد. اصلا مهم نیست که p به چه شیئی اشاره دارد. مهم این است که از کلاس X* است و توابع آن را می‌شناسد. مثال زیر این مطلب را تایید می‌کند.

× مثال 8-11 اشاره‌گری از کلاس والد به شیئی از کلاس فرزند

در برنامهٔ زیر، کلاس Y زیرکلاسی از X است. هر دوی این کلاس‌ها دارای یک عضو تابعی به نام f() هستند و p اشاره‌گری از نوع X* تعریف شده:

```
class X
{  public:
      void f() { cout << "X::f() executing\n"; }
};

class Y : public X
{  public:
      void f() { cout << "Y::f() executing\n"; }
}

int main()
{  X x;
   Y y;
```

```
   X* p = &x;
   p->f();          // invokes X::f() because p has type X*
   p = &y;
   p->f();          // invokes X::f() because p has type X*
}
```

```
X::f() executing
X::f() executing
```

در برنامهٔ بالا دو بار p->f() فراخوانی شده. یک بار برای شیء x از کلاس X و یک بار هم برای شیء y از کلاس Y ولی در هر دو بار بدون توجه به این که p به اشیای چه کلاسی اشاره دارد، فقط X::f() فراخوانی شده.

مشخص است که اثر فوق مطلوب به نظر نمی‌رسد. انتظار داریم که برای اشیای Y، هنگام اجرای p->f() تابع Y::f() فراخوانی شود و برای اشیای X هم هنگام اجرای p->f() تابع X::f() فرا خوانده شود. برای برآوردن این انتظار، از «توابع مجازی» استفاده می‌کنیم.

11-7 توابع مجازی و چندریختی

تابع مجازی[1] تابعی است که با کلمهٔ کلیدی virtual مشخص می‌شود. وقتی یک تابع به شکل مجازی اعلان می‌شود، یعنی در حداقل یکی از کلاس‌های فرزند نیز تابعی با همین نام وجود دارد. توابع مجازی امکان می‌دهند که هنگام استفاده از اشاره‌گرها، بتوانیم بدون در نظر گرفتن نوع اشاره‌گر، به توابع شیء جاری دستیابی کنیم. به مثال زیر دقت کنید.

× مثال 11-9 استفاده از توابع مجازی

این برنامه، همان برنامهٔ مثال 11-8 است با این تفاوت که تابع f() از کلاس والد به شکل یک تابع مجازی اعلان شده است:

```
class X
{  public:
      virtual void f() { cout << "X::f() executing\n"; }
```

1 – Virtual function

```
};

class Y : public X
{  public:
      void f() { cout << "Y::f() executing\n"; }
}

int main()
{  X x;
   Y y;
   X* p = &x;
   p->f();          // invokes X::f()
   p = &y;
   p->f();          // invokes Y::f()
}
```

```
X::f() executing
Y::f() executing
```

اکنون وقتی p به شیئی از نوع X اشاره کند و تابع p->f() فراخوانی شود، X::f() اجرا می‌شود و وقتی p به شیئی از نوع Y اشاره کند و تابع p->f() فراخوانی شود، این دفعه Y::f() اجرا می‌شود. به این مفهوم، «چندشکلی» یا ***چندریختی*** می‌گویند زیرا فراخوانی p->f() بسته به این که p به چه نوع شیئی اشاره کند، نتایج متفاوتی تولید می‌کند. دقت داشته باشید که فقط توابع کلاس والد را به شکل مجازی تعریف می‌کنند. مثال زیر، نمونهٔ مفیدتری از چندریختی را نشان می‌دهد.

× مثال 10-11 چندریختی از طریق توابع مجازی

سه کلاس زیر را در نظر بگیرید. بدون استفاده از توابع‌مجازی، برنامه آن طور که مورد انتظار است کار نمی‌کند:

```
class Book
{  public:
      Book(char* s) { name = new char[strlen(s+1)];
                      strcpy(name, s);
                    }
      void print() { cout << "Here is a book with name "
```

```
                        << name << ".\n";
                }
   protected:
      char* name;
};
class Ebook : public Book
{  public:
      Ebook(char* s, float g) : Book(s), size(g) {}
      void print() { cout << "Here is an Ebook with name "
                        << name << " and size "
                        << size << " MB.\n";
                }
   private:
      float size;
}
class Notebook : public Book
{  public:
      Notebook(char* s, int n) : Book(s) , pages(n) {}
      void print() { cout << "Here is a Notebook with name "
                        << name << " and " << pages
                        << " pages.\n";
                }
   private:
      int pages;
};

int main()
{  Book* b;
   Book mybook("C++");
   b = &mybook;
   b->print();
   Ebook myebook("C#", 5.16);
   b = &myebook;
   b->print();
   Notebook mynotebook(".NET", 230);
   b = &mynotebook;
```

```
   b->print();
}
```

```
Here is a book with name C++.
Here is a book with name C#.
Here is a book with name .NET.
```

در برنامهٔ بالا کلاس Book یک کلاس والد است که دارای دو فرزند به نام‌های Ebook و Notebook می‌باشد. هر سهٔ این کلاس‌ها دارای تابع print() هستند. اشاره‌گر b از نوع کلاس والد یعنی Book* تعریف شده، پس می‌توان آن را به اشیایی از کلاس‌های فرزند نیز اشاره داد اما هر دفعه هنگام فراخوانی b->print() فقط تابع print() از کلاس Book اجرا می‌شود.

حالا تابع print() در کلاس والد را به شکل مجازی اعلان می‌کنیم:

```
class Book
{  public:
      Book(char* s) { name = new char[strlen(s+1)];
                      strcpy(name, s);
                    }
      virtual void print() { cout << "Here is a book with
                                     name " << name << ".\n";
                             }
   protected:
      char* name;
};
```

اکنون که با این تغییر، دوباره برنامه را اجرا کنیم، همه چیز به درستی طبق انتظار ما پیش می‌رود:

```
Here is a book with name C++.
Here is an Ebook with name C# and size 5.16 MB.
Here is a Notebook with name .NET and pages 230.
```

فراخوانی b->print() چندریختی دارد و بسته به این که b به چه نوعی اشاره داشته باشد، نتیجهٔ متفاوتی خواهد داشت.

زمانی یک تابع را به شکل مجازی تعریف می‌کنند که احتمال بدهند حداقل یک زیرکلاس، نسخه‌ای محلی از همان تابع را خواهد داشت.

11-8 نابودکنندهٔ مجازی

با توجه به تعریف توابع مجازی، به نظر می‌رسد که نمی‌توان توابع سازنده و نابودکننده را به شکل مجازی تعریف نمود زیرا سازنده‌ها و نابودگرها در کلاس‌های والد و فرزند، هم‌نام نیستند. در اصل، سازنده‌ها را نمی‌توان به شکل مجازی تعریف کرد اما نابودگرها قصهٔ دیگری دارند. مثال بعدی ایراد مهلکی را نشان می‌دهد که با مجازی کردن نابودگر، برطرف می‌شود.

× مثال 11-11 حافظهٔ گم شده

به برنامهٔ زیر دقت کنید:

```
class X
{  public:
      x() { p = new int[2];  cout << "X().   "; }
      ~X() { delete [] p;  cout << "~X().\n" }
   private:
      int* p;
};

class Y : public X
{  public:
      Y() { q = new int[1023];  cout << "Y() : Y::q = " << q
                                    << ".   "; }
       ~Y() { delete [] q;  cout << "~Y().   "; }
   private:
      int* q;
};

int main()
{  for (int i=0; i<8; i++)
   {  X* r = new Y;
```

```
      delete r;
   }
}
```

```
X().   Y() : Y::q = 0x5821c.   ~X().
X().   Y() : Y::q = 0x5921c.   ~X().
X().   Y() : Y::q = 0x5a21c.   ~X().
X().   Y() : Y::q = 0x5b21c.   ~X().
X().   Y() : Y::q = 0x5c21c.   ~X().
X().   Y() : Y::q = 0x5d21c.   ~X().
X().   Y() : Y::q = 0x5e21c.   ~X().
X().   Y() : Y::q = 0x5f21c.   ~X().
```

در هر تکرار حلقهٔ for یک شیء پویای جدید ساخته می‌شود. طبق انتظار، سازنده‌ها از بالا به پایین اجرا می‌شوند یعنی ابتدا X() و سپس Y() . با این حساب برای هر شی، 4100 بایت از حافظه تخصیص می‌یابد (8 بایت برای X و 4092 بایت برای Y). اما چون r به عنوان اشاره‌گری به اشیای X معرفی شده، فقط نابودگر X را احضار می‌کند و فقط 8 بایت را آزاد می‌سازد و 4092 بایت باقی‌مانده، بلاتکلیف می‌مانند. پس در هر تکرار 3992 بایت از حافظه ناپدید می‌شود! خروجی نیز این را تایید می‌کند. برای رفع این اشکال، نابودگر ~X() را به شکل مجازی معرفی کرده و برنامه را دوباره اجرا می‌نماییم:

```
class X
{  public:
      x() { p = new int[2];  cout << "X().   "; }
      virtual ~X() { delete [] p;  cout << "~X().\n" }
   private:
      int* p;
};
```

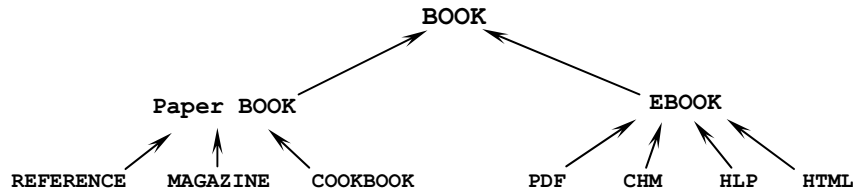
```
X().   Y() : Y::q = 0x5a220.   ~Y().   ~X().
X().   Y() : Y::q = 0x5a220.   ~Y().   ~X().
X().   Y() : Y::q = 0x5a220.   ~Y().   ~X().
X().   Y() : Y::q = 0x5a220.   ~Y().   ~X().
X().   Y() : Y::q = 0x5a220.   ~Y().   ~X().
X().   Y() : Y::q = 0x5a220.   ~Y().   ~X().
X().   Y() : Y::q = 0x5a220.   ~Y().   ~X().
X().   Y() : Y::q = 0x5a220.   ~Y().   ~X().
```

حالا هر تکرار حلقهٔ for هر دو نابودگر فرا خوانده می‌شود و در نتیجه همهٔ حافظهٔ تخصیص یافته آزاد می‌شود و این حافظه دوباره می‌تواند برای اشاره‌گر r استفاده شود.

مثال بالا، ایراد ***شکاف حافظه***[1] را نشان می‌دهد. در یک نرم‌افزار بزرگ، این مساله می‌تواند باعث بروز فاجعه شود. مخصوصا این که پیدا کردن ایراد فوق بسیار مشکل است. پس بخاطر داشته باشید که هر وقت از اشیای پویا در برنامه استفاده می‌کنید، نابودگر کلاس والد را به شکل مجازی اعلان نمایید.

9-11 کلاس‌های پایهٔ انتزاعی

در شی‌گرایی رسم بر این است که ساختار برنامه و کلاس‌ها را طوری طراحی کنند که بتوان آن‌ها را به شکل یک نمودار درختی شبیه زیر نشان داد:



برای مثال ممکن است بخواهیم در یک برنامه از کلاس Html استفاده کنیم. شی‌گرایی توصیه دارد که ابتدا کلاس پایهٔ Book را اعلان کنیم و از آن کلاس، Ebook را ***مشتق***[2] کرده و از Ebook نیز زیرکلاس Html را مشتق نماییم. گرچه ممکن است این روش در ابتدا بی‌معنی و حتی وقت‌گیر و غیرمنطقی به نظر برسد، اما در اصل روش مذکور سبب سهولت در اصلاح و پشتیبانی برنامه می‌شود. این موضوع در برنامه‌های تجاری، بسیار مهم و حیاتی است.

می‌دانیم که در ساختار فوق، هر کلاس دارای توابع خاصی است که رفتارهای ویژهٔ آن کلاس را نشان می‌دهد. ممکن است توابعی مجازی هم وجود داشته باشند که در همهٔ زیرکلاس‌ها مغلوب شوند و نسخه‌ای محلی از آن‌ها ایجاد شود. در این حالت دیگر نیازی نیست که توابع مذکور در کلاس‌های پایه دارای بدنه باشند زیرا این بدنه

1 – Memory leak
2 - Derrive

هرگز به کار گرفته نمی‌شود. در این گونه مواقع، تابع مذکور را به شکل یک ***تابع مجازی خالص***[1] اعلان می‌کنند. یک تابع مجازی خالص، تابعی است که هیچ پیاده‌سازی ندارد و فاقد بدنه است. در این توابع به جای بدنه، عبارت =0; را قرار می‌دهند. برای مثال، در همهٔ کلاس‌های رابطهٔ فوق، تابعی به نام print() وجود دارد که هر زیرکلاس یک نسخهٔ محلی از آن را دارد. پس لازم نیست که در کلاس پایهٔ Book برای این تابع، بدنه‌ای تعریف کنیم. فقط کافی است دستور

```
virtual void print()=0;
```

را در کلاس Book بگنجانیم. هر وقت چنین عبارتی در بدنهٔ یک کلاس ظاهر شود، به این معناست که تابع مذکور در این کلاس هرگز فراخوانی نمی‌شود و زیرکلاسی وجود دارد که نسخهٔ محلی از آن تابع را به کار خواهد گرفت. کلاسی که یک یا چند تابع مجازی خالص داشته باشد را ***کلاس پایهٔ انتزاعی***[2] می‌نامیم (به آن کلاس پایهٔ مجرد نیز گفته می‌شود). کلاس‌های پایهٔ مجرد فقط برای ساختن و اشتقاق زیرکلاس‌ها به کار می‌روند و هیچ شیئی مستقیما از روی آن‌ها ساخته نمی‌شود. کلاسی که هیچ تابع مجازی خالص نداشته باشد را «کلاس مشتق‌شدهٔ واقعی» می‌گوییم. اشیای داخل یک برنامه از روی کلاس‌های مشتق شدهٔ واقعی ساخته می‌شوند.

اگر تابع مجازی خالص اصلا فراخوانی نمی‌شود، پس اصلا چرا آن را اعلان می‌کنیم؟ پاسخ این است که تابع مجازی خالص را اعلان می‌کنیم تا تاکید کنیم که دست کم یک زیرکلاس باید نسخه‌ای محلی از این تابع را داشته باشد.

× مثال 12-11 کلاس پایهٔ انتزاعی و یک کلاس مشتق شدهٔ واقعی

```
class VCR
{  public:
      virtual void on() =0;
      virtual void off() =0;
      virtual void record() =0;
      virtual void stop() =0;
      virtual void play() =0;
};
```

1 – Pure virtual function
2 – Abstracted base class

```
class Video : public VCR
{  public:
      void on();
      void off();
      void record();
      void stop();
      void play();
};
class Camera : public VCR
{  public:
      void on();
      void off();
      void record();
      void stop();
      void play();
};
```

در کد بالا، کلاس VCR یک کلاس پایهٔ انتزاعی است که کلاس Video و Camera از آن مشتق می‌شود. توابع عضو کلاس VCR همگی به شکل یک تابع مجازی خالص اعلان شده‌اند. این توابع در زیرکلاس Video و Camera نیز اعلان شده‌اند که نسخه‌های محلی و غالب را نشان می‌دهند.

مثال‌هایی که در این فصل و دو فصل قبل دیدید خیلی کاربردی نستند و فقط برای آموزش مفاهیم شی‌گرایی بیان شده‌اند. حالا که با شی‌گرایی و مزایای آن آشنا شده‌اید، وقت آن رسیده که برنامه‌نویسی شی‌گرا را با جدیت دنبال کنید و تلاش داشته باشید که از این پس، برنامه‌هایی که می‌نویسید شی‌گرا باشند. ممکن است در شروع کار با مشکلاتی مواجه شوید که با مرور مطالب ذکر شده خواهید توانست بر آن‌ها غلبه کنید. هر چه بیشتر در شی‌گرایی تبحر پیدا کنید، برنامه‌های تجاری مطمئن‌ترین خواهید نوشت و احتمال شکست در پروژه‌های گروهی که اجرا می‌کنید، کم‌تر می‌شود.

مسلم است که شی‌گرایی، رهیافت نهایی در جهان برنامه‌نویسی نیست و هر چه زمان می‌گذرد، راهکارهای تازه‌ای در جهت آسان‌سازی برنامه‌نویسی پدید می‌آید لیکن برنامه‌نویسان شی‌گرایی را به عنوان یک راهکار قابل اعتماد، پذیرفته‌اند.

پرسش‌های گزینه‌ای

1 – در کد `class X : public Y { }` کدام گزینه صحیح است؟

الف – کلاس X عضوی از نوع کلاس Y دارد

ب – کلاس X والد کلاس Y است

ج – کلاس Y از کلاس X مشتق شده است

د – کلاس X به اعضای عمومی کلاس Y دسترسی دارد

2 – اگر کلاس A فرزند کلاس B باشد، آنگاه:

الف – کلاس A به اعضای عمومی کلاس B دسترسی دارد

ب – کلاس B به اعضای عمومی کلاس A دسترسی دارد

ج – کلاس B به اعضای خصوصی کلاس A دسترسی دارد

د – کلاس B به اعضای حفاظت‌شدهٔ کلاس A دسترسی دارد

3 – فرض کنید کلاس A فرزند کلاس B است. اگر کلاس B دارای یک عضو حفاظت‌شده به نام m باشد آنگاه:

الف – کلاس A عضو m را به شکل یک عضو خصوصی به ارث می‌گیرد

ب – کلاس A عضو m را به شکل یک عضو عمومی به ارث می‌گیرد

ج – کلاس A عضو m را به شکل یک عضو حفاظت شده به ارث می‌گیرد

د – کلاس A اجازهٔ دستیابی به عضو m را ندارد

4 – فرض کنید کلاس A فرزند کلاس B است و در هر دو کلاس، یک عضو عمومی به نام x اعلان شده. حال اگر a یک شی از کلاس A و b یک شی از کلاس B باشد، کدام گزینه صحیح است؟

الف – `a.x` عضو غالب است و `a.B::x` عضو موروثی است

ب – `b.x` عضو غالب است و `b.A::x` عضو موروثی است

ج – `a.B::x` عضو غالب است و `a.x` عضو موروثی است

د – `b.A::x` عضو غالب است و `b.x` عضو موروثی است

5 – عضو حفاظت‌شده عضوی است که:

الف – از خارج کلاس قابل دستیابی است

ب – فقط توسط اعضای همان کلاس قابل دستیابی است

ج – فقط توسط اعضای کلاس فرزند قابل دستیابی است

د – توسط اعضای همان کلاس و اعضای کلاس فرزند قابل دستیابی است

6 – کدام گزینه در وراثت صحیح نیست؟

الف – قبل از فراخوانی سازندهٔ والد، سازندهٔ فرزند فراخوانی می‌شود

ب – بعد از فراخوانی نابودکنندهٔ فرزند، نابودکنندهٔ والد فراخوانی می‌شود

ج – قبل از فراخوانی سازندهٔ فرزند، سازندهٔ والد فراخوانی می‌شود

د – فراخوانی سازندهٔ والد ربطی به سازندهٔ فرزند ندارد

7 – کدام گزینه در بارهٔ چندریختی کلاس‌ها صحیح است؟

الف – چندریختی به وسیلهٔ استفاده از توابع مجازی امکان‌پذیر است

ب – چندریختی به وسیلهٔ استفاده از اعضای حفاظت‌شده امکان‌پذیر است

ج – چندریختی به وسیلهٔ استفاده از اشیای ثابت امکان‌پذیر است

د – چندریختی به وسیلهٔ استفاده از توابع دستیابی امکان‌پذیر است

8 – کد `p->f();` معادل کدام کد زیر است؟

الف – `*p.f();`

ب – `p.f();`

ج – `(*p).f();`

د – `p.(*f)()`

9 – کدام گزینه صحیح است؟

الف – تابع سازنده را می‌توان در کلاس والد به شکل مجازی تعریف کرد ولی نابودکننده را نمی‌توان

ب – تابع نابودکننده را می‌توان در کلاس والد به شکل مجازی تعریف کرد ولی سازنده را نمی‌توان

ج – هر دو تابع سازنده و نابودکننده را می‌توان در کلاس والد به شکل مجازی تعریف کرد

د – هیچ یک از دو تابع سازنده و نابودکننده را نمی‌توان در کلاس والد به شکل مجازی تعریف کرد.

10 – تابع مجازی خالص تابعی است که :

الف – هیچ پیاده‌سازی ندارد و تعریف آن فاقد بدنه است

ب – نمی‌تواند توسط کلاس فرزند به ارث گرفته شود

ج – عضو هیچ کلاسی نیست

د – هیچ پارامتری ندارد و نوع بازگشتی آن `void` است

11 – وقتی در تعریف یک کلاس از تابع مجازی خالص استفاده شود، آنگاه:

الف – آن کلاس حداقل یک فرزند دارد

ب – در یک فرزند آن کلاس، تابع مذکور یک نسخهٔ محلی دارد

ج – تابع مذکور توسط اشیای آن کلاس هرگز فراخوانی نمی‌شود

د – هر سه مورد

12 – اگر کلاسی یک کلاس پایهٔ مجرد باشد آنگاه:

الف – آن کلاس هیچ تابع مجازی خالص ندارد

ب – آن کلاس یک یا چند تابع مجازی خالص دارد

ج – آن کلاس برای اشتقاق زیرکلاس‌ها به کار می‌رود

د – گزینهٔ ب و ج صحیح است.

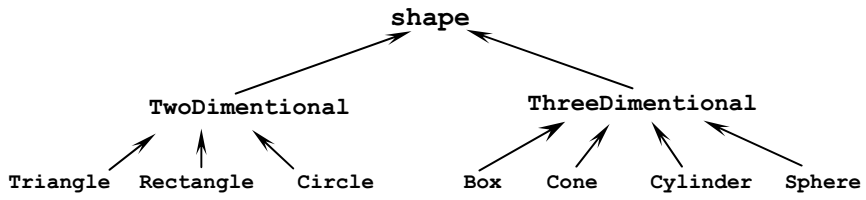
پرسش‌های تشریحی

1- ترکیب چه تفاوتی با وراثت دارد؟
2- اعضای protected چه تفاوتی با اعضای private دارند؟
3- رفتار سازندهٔ پیش‌فرض و نابودکننده در وراثت چگونه است؟
4- تابع عضو virtual (مجازی) چیست؟
5- تابع عضو مجازی خالص چیست؟
6- شکاف حافظه چگونه ایجاد می‌شود و برای جلوگیری از آن چه باید کرد؟
8- کلاس پایهٔ انتزاعی چیست و چرا ساخته می‌شود؟
9- کلاس مشتق‌شدهٔ واقعی چیست؟
10- بسته‌بندی ایستا و بسته‌بندی پویا چه تفاوتی با هم دارد؟
12- چندریختی چیست و چگونه سبب توسعهٔ کلاس می‌شود؟
13- چه خطایی در تعاریف زیر است؟

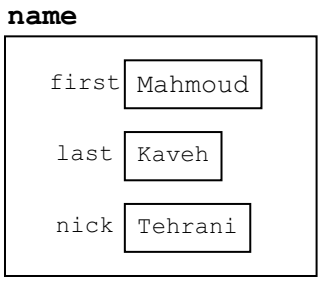
```
class X
{  protected:
      int a;
};

class Y : public X
{  public:
      void set(X x, int c) { x.a = c; }
};
```

تمرین‌های برنامه‌نویسی

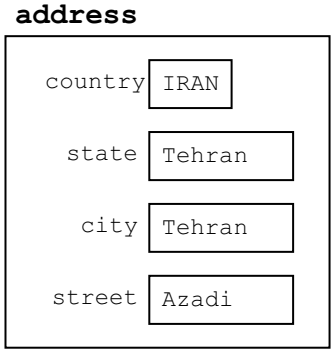
1- سلسله مراتب کلاس زیر را پیاده سازی کنید:



2- کلاس Name که اشیای آن شبیه شکل زیر هستند را تعریف و آزمایش کنید. سپس کلاسی به نام Person ایجاد کنید که مشخصات افراد شامل نام، تاریخ تولد، سن و مدرک تحصیلی را نگهداری کند. کلاس Person را طوری تعریف کنید که برای نگهداری نام به جای نوع string از نوع Name استفاده کند:

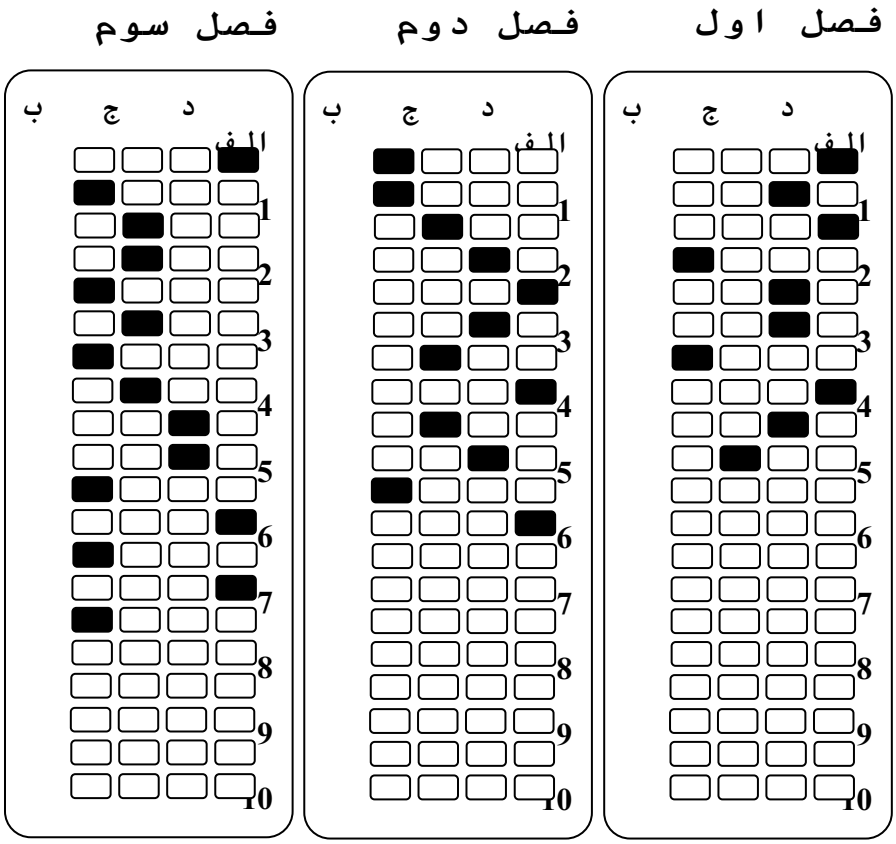


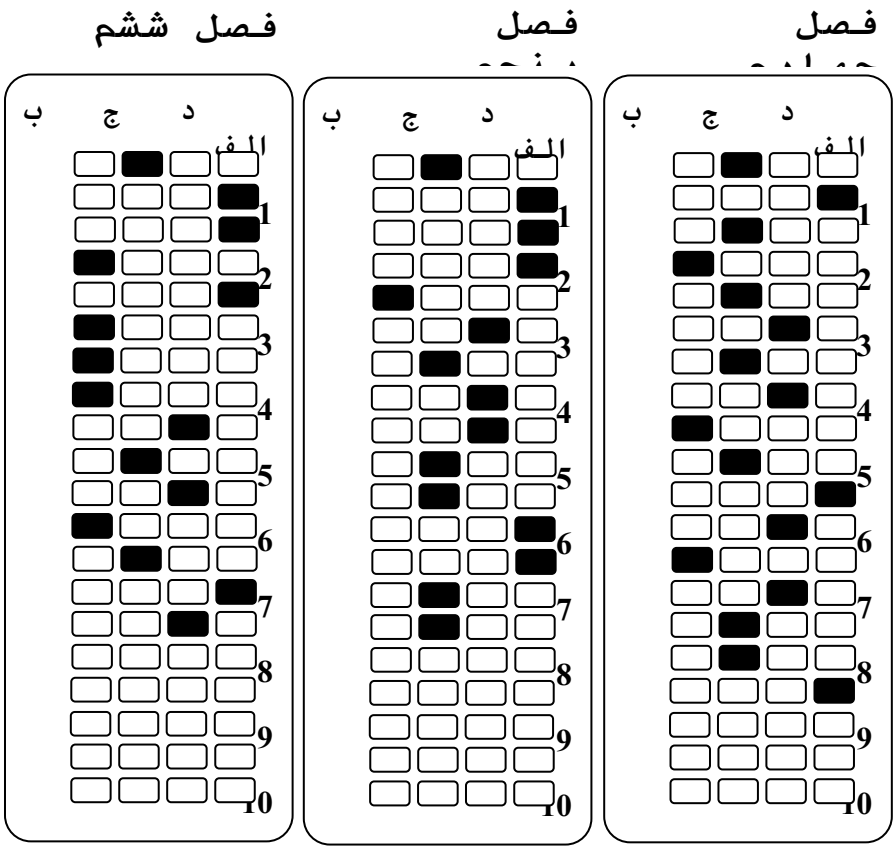
3- کلاس Address که اشیای آن شبیه شکل زیر هستند را تعریف و آزمایش کنید. سپس با افزودن یک عضو داده‌ای address به کلاس Person در تمرین قبلی، آن را بهبود دهید.



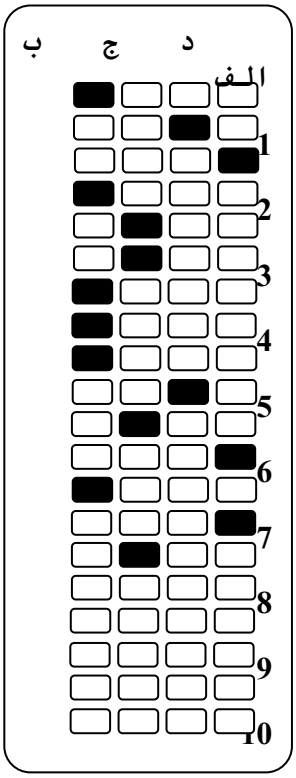
ضمیمهٔ « الف »

پاسخ‌نامهٔ پرسش‌های گزینه‌ای

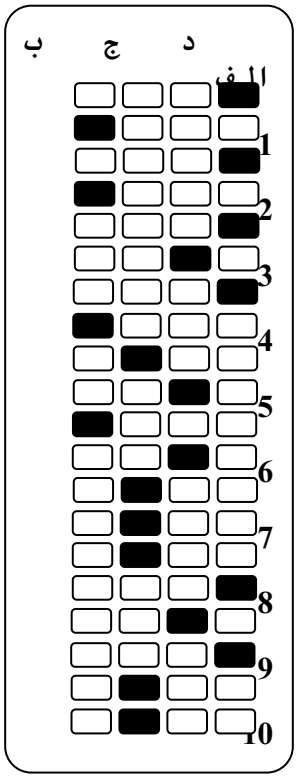



فصل ششم
فصل
ب ج د
الف
1
2
3
4
5
6
7
8
9
10

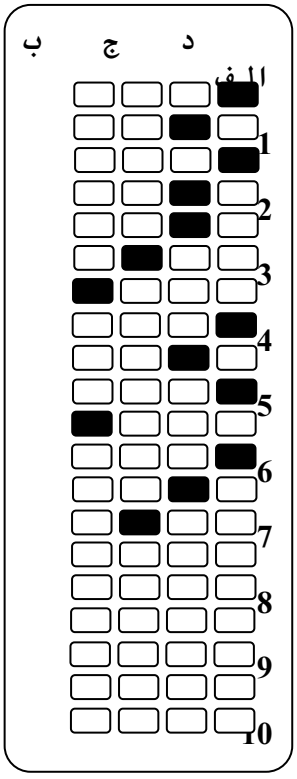
فصل هفتم

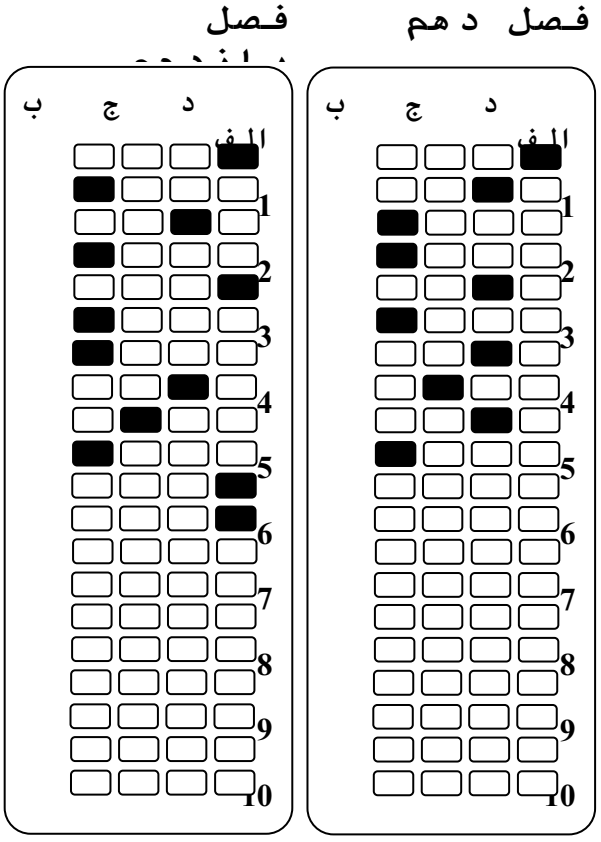


فصل هشتم



فصل نهم




فصل دهم
فصل
الف ب ج د
1 2 3 4 5 6 7 8 9 10
الف ب ج د
1 2 3 4 5 6 7 8 9 10

ضمیمه « ب »

جدول اسکی

هر کاراکتر به صورت یک کد صحیح در حافظه ذخیره می‌شود. این کد، عددی در محدودهٔ صفر تا 127 است. به جدولی که تمام 128 کاراکتر موجود در این محدوده را با یک ترتیب از پیش مشخص نشان می‌دهد، جدول ASCII می‌گویند. شرح این جدول در ادامه آمده است. توجه کنید که 32 کاراکتر اول کاراکترهای چاپ‌نشدنی هستند. لذا به جای نشان دادن سمبل آن‌ها در ستون اول، زنجیرهٔ کنترلی آن‌ها یا زنجیرهٔ خروجی آن‌ها نشان داده شده است. *زنجیرهٔ کنترلی* یک کاراکتر غیر چاپی، ترکیبی از کلید Ctrl با کلید دیگری است که این دو باهم فشار داده می‌شوند تا کاراکتر مورد نظر وارد شود. برای مثال، کاراکتر پایان فایل (با کد اسکی 4) با زنجیرهٔ Ctrl+D وارد می‌شود. *زنجیرهٔ خروجی* یک کاراکتر غیر چاپی، ترکیبی است از کاراکتر بک اسلش \ (که کاراکتر کنترل نامیده شده) و یک حرف که در کد منبع C++ تایپ می‌شود تا آن کاراکتر را مشخص کند. برای مثال کاراکتر خط جدید (با کد اسکی 10) در برنامهٔ C++ به صورت \n نوشته می‌شود.

کاراکتر	شرح	مبنای ده	مبنای هشت	مبنای شانزده
Ctrl+@	خالی – پایان رشته	0	000	0x0
Ctrl+A	شروع سرآیند	1	001	0x1
Ctrl+B	شروع متن	2	002	0x2
Ctrl+C	پایان متن	3	003	0x3
Ctrl+D	پایان انتقال – پایان فایل	4	004	0x4
Ctrl+E	رهایی، بیرون رفتن	5	005	0x5

کاراکتر	شرح	مبنای ده	مبنای هشت	مبنای شانزده
Ctrl+F	تصدیق	6	006	0x6
\a	زنگ، بوق سیستم	7	007	0x7
\b	پسخور	8	010	0x8
\t	جدول افقی	9	011	0x9
\n	تغذیه خط، خط جدید	10	012	0xa
\v	جدول عمودی	11	013	0xb
\f	تغذیه فرم، صفحه جدید	12	014	0xc
\r	برگشت نورد	13	015	0xd
Ctrl+N	شیفت(جابجایی) بیرون	14	016	0xe
Ctrl+O	شیفت داخل	15	017	0xf
Ctrl+P	پیوند داده	16	020	0x10
Ctrl+Q	وسیلهٔ کنترلی 1، پیمایش مجدد	17	021	0x11
Ctrl+R	وسیلهٔ کنترلی 2	18	022	0x12
Ctrl+S	وسیلهٔ کنترلی 3، توقف پیمایش	19	023	0x13
Ctrl+T	وسیلهٔ کنترلی 4	20	024	0x14
Ctrl+U	تصدیق منفی	21	025	0x15
Ctrl+V	بی‌کار کردن رخداد هم‌زمان	22	026	0x16
Ctrl+W	پایان بلوک انتقال	23	027	0x17
Ctrl+X	لغو	24	030	0x18
Ctrl+Y	پایان پیام، وقفه	25	031	0x19
Ctrl+Z	جانشینی، خروج	26	032	0x1a
Ctrl+[	بیرون رفتن، رهایی	27	033	0x1b
Ctrl+/	جداساز فایل	28	034	0x1c
Ctrl+]	جداساز گروه	29	035	0x1d
Ctrl+^	جداساز رکورد	30	036	0x1e
Ctrl+_	جداساز واحد	31	037	0x1f

کاراکتر	شرح	مبنای ده	مبنای هشت	مبنای شانزده
	جای خالی	32	040	0x20
!	علامت تعجب	33	041	0x21
"	علامت نقل قول	34	042	0x22
#	علامت شماره	35	043	0x23
$	علامت دلار	36	044	0x24
%	علامت درصد	37	045	0x25
&	علامت and	38	046	0x26
'	نقل قول تنها، آپستروف	39	047	0x27
(	پرانتز چپ	40	050	0x28
)	پرانتز راست	41	051	0x29
*	ضربدر، ستاره	42	052	0x2a
+	بعلاوه	43	053	0x2b
,	کاما، ویرگول	44	054	0x2c
-	تفریق	45	055	0x2d
.	نقطه، نقطه اعشار	46	056	0x2e
/	اسلش	47	057	0x2f
0	رقم صفر	48	060	0x30
1	رقم یک	49	061	0x31
2	رقم دو	50	062	0x32
3	رقم سه	51	063	0x33
4	رقم چهار	52	064	0x34
5	رقم پنج	53	065	0x35
6	رقم شش	54	066	0x36
7	رقم هفت	55	067	0x37
8	رقم هشت	56	070	0x38
9	رقم نه	57	071	0x39

کاراکتر	شرح	مبنای ده	مبنای هشت	مبنای شانزده
:	کولن	58	072	0x3a
;	سمیکولن	59	073	0x3b
<	کوچک‌تر	60	074	0x3c
=	مساوی	61	075	0x3d
>	بزرگ‌تر	62	076	0x3e
?	علامت سوال	63	077	0x3f
@	علامت تجاری	64	0100	0x40
A	حرف A بزرگ	65	0101	0x41
B	حرف B بزرگ	66	0102	0x42
C	حرف C بزرگ	67	0103	0x43
D	حرف D بزرگ	68	0104	0x44
E	حرف E بزرگ	69	0105	0x45
F	حرف F بزرگ	70	0106	0x46
G	حرف G بزرگ	71	0107	0x47
H	حرف H بزرگ	72	0110	0x48
I	حرف I بزرگ	73	0111	0x49
J	حرف J بزرگ	74	0112	0x4a
K	حرف K بزرگ	75	0113	0x4b
L	حرف L بزرگ	76	0114	0x4c
M	حرف M بزرگ	77	0115	0x4d
N	حرف N بزرگ	78	0116	0x4e
O	حرف O بزرگ	79	0117	0x4f
P	حرف P بزرگ	80	0120	0x50
Q	حرف Q بزرگ	81	0121	0x51
R	حرف R بزرگ	82	0122	0x52
S	حرف S بزرگ	83	0123	0x53

کاراکتر	شرح	مبنای ده	مبنای هشت	مبنای شانزده
T	حرف T بزرگ	84	0124	0x54
U	حرف U بزرگ	85	0125	0x55
V	حرف V بزرگ	86	0126	0x56
W	حرف W بزرگ	87	0127	0x57
X	حرف X بزرگ	88	0130	0x58
Y	حرف Y بزرگ	89	0131	0x59
Z	حرف Z بزرگ	90	0132	0x5a
[	براکت چپ	91	0133	0x5b
\	بک اسلش	92	0134	0x5c
]	براکت راست	93	0135	0x5d
^	توان	94	0136	0x5e
_	خط زیرین	95	0137	0x5f
`	علامت تاکید هجا	96	0140	0x60
a	حرف A کوچک	97	0141	0x61
b	حرف B کوچک	98	0142	0x62
c	حرف C کوچک	99	0143	0x63
d	حرف D کوچک	100	0144	0x64
e	حرف E کوچک	101	0145	0x65
f	حرف F کوچک	102	0146	0x66
g	حرف G کوچک	103	0147	0x67
h	حرف H کوچک	104	0150	0x68
i	حرف I کوچک	105	0151	0x69
j	حرف J کوچک	106	0152	0x6a
k	حرف K کوچک	107	0153	0x6b
l	حرف L کوچک	108	0154	0x6c
m	حرف M کوچک	109	0155	0x6d

کاراکتر	شرح	مبنای ده	مبنای هشت	مبنای شانزده
n	حرف N کوچک	110	0156	0x6e
o	حرف O کوچک	111	0157	0x6f
p	حرف P کوچک	112	0160	0x70
q	حرف Q کوچک	113	0161	0x71
r	حرف R کوچک	114	0162	0x72
s	حرف S کوچک	115	0163	0x73
t	حرف T کوچک	116	0164	0x74
u	حرف U کوچک	117	0165	0x75
v	حرف V کوچک	118	0166	0x76
w	حرف W کوچک	119	0167	0x77
x	حرف X کوچک	120	0170	0x78
y	حرف Y کوچک	121	0171	0x79
z	حرف Z کوچک	122	0172	0x7a
{	آکولاد (ابرو) چپ	123	0173	0x7b
|	خط لوله	124	0174	0x7c
}	آکولاد (ابرو) راست	125	0175	0x7d
~	علامت نقیض	126	0176	0x7e
Delete	حذف، پاک کردن	127	0177	0x7f

ضمیمه « ج »

کلمات کلیدی C++

کلمه کلیدی	شرح	مثال
and	مترادفی برای عملگر منطقی AND : &&	(x>0 and x<8)
and_eq	مترادفی برای عملگر تخصیصی AND بیتی : &=	b1 and_eq b2;
asm	اجازه می دهد تا اطلاعات مستقیما به اسمبلر فرستاده شود	asm ("check");
auto	کلاس ذخیره سازی برای اشیایی که فقط درون بلاک خودشان ایجاد می شوند	auto int n;
bitand	مترادفی برای عملگر AND بیتی : &	b0 = b1 bitand b2;
bitor	مترادفی برای عملگر OR بیتی : |	b0 = b1 bitor b2;
bool	معرف نوع بولین	bool flag;
break	یک حلقه یا عبارت switch را خاتمه می دهد	break;
case	در عبارت switch شرط کنترلی را مشخص می کند	case (n/10)
catch	فعالیت هایی را مشخص می کند که وقتی یک استثنا رخ می دهد باید اجرا شوند	catch(error)
char	معرف نوع صحیح	char c;
class	تعریف یک کلاس را مشخص می کند	class X { … };
compl	مترادفی برای عملگر منطقی NOT بیتی : ~	b0 = compl b1;
const	تعریف یک ثابت را مشخص می کند	const int s = 32;
const_cast	برای تغییر دادن اشیا از درون تابع عضو تغییرناپذیر استفاده می شود	pp = const_cast<T*>(p)
continue	در حلقه به ابتدای دور بعدی پرش می کند	continue;
default	حالت «وگر نه» در عبارت switch	default: sum = 0;

کلمه کلیدی	شرح	مثال
delete	حافظه ای را که با عبارت new تخصیص یافته آزاد می کند	delete a;
do	یک حلقه do..while را مشخص می کند	do {…} while …
double	نوع عددی حقیقی (ممیز شناور با دقت مضاعف)	double x;
dynamic_cast	برای اشاره گر داده شده، اشاره گر T* را بر می گرداند	pp = dynamic_cast<T*>p
else	بخش دیگر عبارت if را مشخص می کند	else n=0;
enum	برای تعریف نوع شمارشی استفاده می شود	enum bool {…};
explicit	باعث می شود تا از فراخوانی ضمنی یک سازنده جلوگیری شود	explicit X(int n);
export	دسترسی از واحد کامپایل دیگر را ممکن می سازد	export template<class T>
extern	کلاس ذخیره سازی برای اشیایی که بیرون از بلوک محلی تعریف شده اند	extern int max;
false	یکی از دو مقدار نوع bool	bool flag=false;
float	معرف نوع عددی حقیقی (ممیز شناور)	float x;
for	یک حلقه for ایجاد می کند	for (; ;) …
friend	در یک کلاس، تابع دوست ایجاد می کند	friend int f();
goto	باعث می شود تا اجرای برنامه به یک جملهٔ برچسب دار پرش کند	goto error;
if	یک عبارت شرطی if ایجاد می کند	if (n>0) …
inline	تابعی را تعریف می کند که متن آن تابع باید به جای فراخوانی آن درج شود.	inline int f();
int	معرف نوع عددی صحیح	int n;

کلمه کلیدی	شرح	مثال
long	برای تعریف انواع گسترش یافته صحیح و حقیقی استفاده می شود	long double x;
mutable	به توابع تغییر ناپذیر اجازه می دهد تا فیلد را تغییر دهند	mutable string ssn;
namespace	بلوک های فضای نام (میدان) را می شناساند	namespace best { int num; }
new	حافظه ای را تخصیص می دهد	int* p = new int;
not	مترادفی برای عملگر NOT منطقی : !	(not(x==0))
not_eq	مترادفی برای عملگر نابرابری : =!	(x not_eq 0)
opereator	برای تعریف سربارگذاری عملگرها به کار می رود	x operator++()
or	مترادفی برای عملگر OR منطقی : | |	(x>0 or x<8)
or_eq	مترادفی برای عملگر تخصیصی OR بیتی : = |	b1 or_eq b2;
private	اعضای خصوصی را در یک کلاس مشخص می کند	private: int n;
protected	اعضای حفاظت شده را در یک کلاس مشخص می کند	protected: int n;
public	اعضای عمومی را در یک کلاس مشخص می کند	public: int n;
register	مشخص کننده کلاس ذخیره سازی برای اشیایی که در ثبات ها ذخیره می شوند	register int i;
reinterpret_cast	یک شی با نوع و مقدار داده شده را بر می گرداند	pp = reinterpret_cast<T*>(p)
return	عبارتی که تابع را خاتمه می دهد و یک مقدار را بر می گرداند	return 0;
short	معرف نوع صحیح محدود	short n;
signed	برای تعریف انواع صحیح علامت دار به کار می رود	signed char c;

کلمه کلیدی	شرح	مثال
sizeof	تعداد بایت هایی که برای ذخیره سازی یک شی استفاده شده را بر می گرداند	n = sizeof(float);
static	کلاس ذخیره سازی برای اشیایی که در طول اجرای برنامه وجود دارند	static int n;
static_cast	برای اشاره گر داده شده یک اشاره گر T* بر می گرداند	pp = static_cast<T*>p
struct	تعریف یک ساختار را مشخص می کند	struct X {…};
switch	یک عبارت switch را مشخص می کند	switch (n) { … }
template	یک کلاس template را مشخص می کند	template <class t>
this	اشاره گری که به شیء جاری اشاره می کند	return *this;
throw	برای تولید یک استثنا به کار می رود	throw x();
true	یکی از مقادیر ممکن برای متغیرهای نوع bool	bool flag = true;
try	بلوکی را مشخص می کند که شامل مدیریت کننده استثنا است	try { … }
typedef	برای یک نوع موجود، مترادفی را تعریف می کند	typedef int Num;
typeid	شیئی را بر می گرداند که نوع یک عبارت را نشان می دهد	cout << typeid(x).name();
typename	مترادفی برای کلمه کلیدی class	typename X { … };
using	دستوری که اجازه می دهد تا پیشوند و فضای نام حذف شود	using namespace std;
union	ساختاری را مشخص می کند که اجزای آن از حافظه مشترک استفاده می کنند	union z { … };
unsigned	برای تعریف انواع صحیح بدون علامت استفاده می شود	unsigned int b;
virtual	تابع عضوی را تعریف می کند که در یک زیر کلاس نیز تعریف شده	virtual int f();

کلمه کلیدی	شرح	مثال
void	عدم وجود نوع بازگشتی را معین می کند	void f(); int volatile n;
volatile	اشیایی را تعریف می کند که می توانند خارج از کنترل برنامه تغییر داده شوند	
wchar_t	نوع کاراکتری عریض (16 بیتی) برای unicode	wchar_t province;
while	یک حلقه while ایجاد می کند	while (n > 0) ...
xor	معادلی برای عملگر OR انحصاری بیتی : ^	b0 = b1 xor b2;
xor_eq	معادلی برای عملگر تخصیصی OR انحصاری بیتی : =^	b1 xor_eq b2;

ضمیمه « د »

عملگرهای C++ استاندارد

در این جدول همهٔ عملگرهای C++ با توجه به حق تقدم آن‌ها فهرست شده است. عملگرهای با سطح تقدم بالاتر قبل از عملگرهای با سطح تقدم پایین‌تر ارزیابی می‌شوند. برای مثال در عبارت (a-b*c) عملگر * ابتدا ارزیابی می‌شود و سپس عملگر دوم یعنی – ارزیابی می‌گردد زیرا * در تقدم سطح 13 است که بالاتر از سطح تقدم – که 12 است می‌باشد.

ستونی که با عبارت «وابستگی» مشخص شده، می‌گوید که آن عملگر از چپ ارزیابی می‌شود یا از راست. برای مثال عبارت (a-b-c) به صورت ((a-b)-c) ارزیابی می‌شود زیرا – وابسته به چپ است.

ستونی که با «جمعیت» مشخص شده، می‌گوید که آن عملگر بر روی چند عملگر عمل می‌کند.

ستونی که با «سربار» مشخص شده، می‌گوید که آیا عملگر قابل سربارگذاری هست یا خیر. (سربارگذاری را در فصل دهم مطالعه کنید).

عملوند	نام	سطح	وابستگی	جمعیت	سربار	مثال
::	جداسازی حوزه سراسری	17	راست	یک	خیر	::x
::	جداسازی حوزه کلاسی	17	چپ	دو	خیر	X::x
.	انتخاب عضو مستقیم	16	چپ	دو	خیر	s.len
->	انتخاب عضو غیر مستقیم	16	چپ	دو	بله	p->len
[]	جانشین معادل	16	چپ	دو	بله	a[i]
()	فراخوانی تابع	16	چپ	--	بله	rand()
()	ساختن انواع	16	چپ	--	بله	int(ch)
++	پس افزایشی	16	راست	یک	بله	n++
--	پس کاهشی	16	راست	یک	بله	n--
sizeof	اندازه شی یا نوع	15	راست	یک	خیر	sizeof(a)

++	پیش افزایشی	15	راست	یک	بله	++n
--	پیش کاهشی	15	راست	یک	بله	--n
عملوند	**نام**	**سطح**	**وابستگی**	**جمعیت**	**سربار**	**مثال**
~	مکمل و منفی کردن	15	راست	یک	بله	~s
!	NOT منطقی	15	راست	یک	بله	!p
+	جمع بدون وابستگی	15	راست	یک	بله	+n
-	تفریق بدون وابستگی	15	راست	یک	بله	-n
*	عملگر اشاره	15	راست	یک	بله	*p
&	آدرس	15	راست	یک	بله	&x
new	تخصیص حافظه	15	راست	یک	بله	new p
delete	آزاد سازی حافظه	15	راست	یک	بله	delete p
()	تبدیل نوع	15	راست	دو	بله	int(ch)
.*	انتخاب عضو مستقیم	14	چپ	دو	خیر	x.*q
->*	انتخاب عضو غیر مستقیم	14	چپ	دو	بله	p->q
*	ضرب	13	چپ	دو	بله	m * n
/	تقسیم	13	چپ	دو	بله	m / n
%	باقیمانده	13	چپ	دو	بله	m % n
+	جمع	12	چپ	دو	بله	m + n
-	تفریق	12	چپ	دو	بله	m - n
<<	تغییر جهت به چپ	11	چپ	دو	بله	cout << n
>>	تغییر جهت به راست	11	چپ	دو	بله	cin >> n
<	کوچک‌تر	10	چپ	دو	بله	x < y
<=	کوچک‌تر یا مساوی	10	چپ	دو	بله	x <= y
>	بزرگ‌تر	10	چپ	دو	بله	x > y
>=	بزرگ‌تر یا مساوی	10	چپ	دو	بله	x >= y
==	برابری	9	چپ	دو	بله	x == y
!=	نابرابری	9	چپ	دو	بله	x != y
&	نشانه AND	8	چپ	دو	بله	s & t

^	نشانه XOR	7	چپ	دو	بله	S ^ t
|	نشانه OR	6	چپ	دو	بله	s | t
&&	AND منطقی	5	چپ	دو	بله	u && v
||	OR منطقی	4	چپ	دو	بله	u || v
?:	عبارت شرطی	3	چپ	سه	خیر	U ? x : y
=	تخصیص	2	راست	دو	بله	n = 22
+=	جمع تخصیصی	2	راست	دو	بله	n += 8
عملوند	**نام**	**سطح**	**وابستگی**	**جمعیت**	**سربار**	**مثال**
-=	تفریق تخصیصی	2	راست	دو	بله	n -= 4
*=	ضرب تخصیصی	2	راست	دو	بله	n *= -1
/=	تقسیم تخصیصی	2	راست	دو	بله	n /= 10
%=	باقیمانده تخصیصی	2	راست	دو	بله	n %= 10
&=	AND تخصیصی	2	راست	دو	بله	s &= mask
^=	XOR تخصیصی	2	راست	دو	بله	s ^= mask
|=	OR تخصیصی	2	راست	دو	بله	s |= mask
<<=	تغییر جهت به چپ تخصیصی	2	راست	دو	بله	s << 1
>>=	تغییر جهت به راست تخصیصی	2	راست	دو	بله	s >> 1
,	کاما	0	چپ	دو	بله	++m, --n

ضمیمه « هـ »

فهرست منابع و مأخذ

[Cline]
C++ FAQs, Second Edition, by Marshall Cline, Greg Lomow, and Mike Girou.
Addison-Wesley Publishing Company, Reading, MA (1999) 0-201-30983-1

[Deitel]
C++ How to Program, Second Edition by H. M. Deitel and P. J. Deitel.
Prentice Hall, Englewood Cliffs, NJ (1998) 0-13-528910-6.

[Hubbard1]
Foundamentals of Computing with C++, by John R. Hubbard.
McGraw-Hill, Inc, New York, NY (1998) 0-07-030868-3.

[Hubbard2]
Data Structures with C++, by John R. Hubbard.
McGraw-Hill, Inc, New York, NY (1999) 0-07-135345-3.

[Hubbard3]
Programming with C++, Second Edition, by John R. Hubbard.
McGraw-Hill, Inc, New York, NY (2000) 0-07-118372-8.

[Hughes]
Mastering the Standard C++ Classes, by Cameron Hughes and Tracey Hughes.
John Wiley & Sons, Inc, New York, NY (1999) 0-471-32893-6.

[Johnsonbaugh]
Object_Oriented Programming in C++, by Richard Johnsonbaugh and Martin Kalin.
Prentice Hall, Englewood Cliffs, NJ (1995) 0-02-360682-7.

[Perry]
An Introduction to Object-Oriented Design in C++, by Jo Ellen Perry and Harold D. Levin.
Addison-Wesley Publishing Company, Reading, MA (1996) 0-201-76564-0.

[Savitch]
Problem Solving with C++, by Walter Savitch.
Addison-Wesley Publishing Company, Reading, MA (1996) 0-8053-7440-X.

[Stroustrup1]
The C++ Programming Languages, special Edition, by Bjarne Stroustrup.
Addison-Wesley Publishing Company, Reading, MA (2000) 0-201-70073-5.

[Stroustrup2]
The Design and Evolution of C++, by Bjarne Stroustrup.
Addison-Wesley Publishing Company, Reading, MA (1994) 0-201-54330-3.

[Weiss]
Data Structures and Algorithm Analysis in C++, by Mark Allen Weiss.
Benjamin/Cummings Publishing Company, Redwood City, CA (1994) 0-8053-5443-3.

فهرست مطالب

فصل چهارم

«تکرار»

فصل پنجم

« توابع»

فصل ششم

«آرایه‌ها»

فصل هفتم

«اشاره‌گرها و ارجاع‌ها»

فصل هشتم

«رشته‌های کاراکتری و فایل‌ها در C++ استاندارد»

فصل نهم

«شی‌گرایی»

فصل دهم

«سربارگذاری عملگرها»

فصل یازدهم

«ترکیب و وراثت»