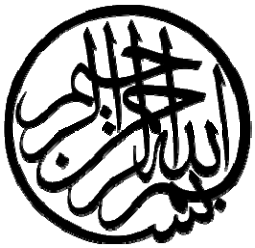
برنامه نویسی به زبان C
با ابزارهای تحت ویندوز

عبدالله آراسته



برنامه نویسی به زبان C

با ابزارهای تحت ویندوز

عبدالله آراسته

مجموعه کتابهای تکمیلی سمپاد

فهرست مطالب

مقدمه	۷
سخنی با دانش‌آموز	۸
فصل اول : آشنایی با زبان های برنامه نویسی و مفاهیم اولیه	**۹**
۱. تاریخچه‌ی توسعه‌ی کامپیوترها و زبانهای برنامه‌نویسی	۱۰
۲. لزوم فراگیری برنامه‌نویسی	۱۳
۳. زبان C و تاریخچه‌ی آن	۱۵
۴. گام‌های برنامه‌نویسی	۲۰
۵. آشنایی با انواع ویرایشگرها و مقایسه‌ی آن‌ها	۲۴
۶. آشنایی با محیط برنامه‌نویسی Dev C++	۲۹
فصل دوم : بررسی مسأله و مشخص کردن ورودی و خروجی ها	**۳۷**
۱. مدل پایه‌ی کامپیوتر و مفهوم متغیر	۳۸
۲. نمودار گردشی	۳۹
۳. حل مسأله‌ی یافتن بیشینه بین سه عدد	۴۱
۴. حل چند مسأله و بازنمایی الگوریتم با نمودار گردشی	۴۴
۵. بازنمایی متنی حل مسأله	۵۲
فصل سوم : ورودی-خروجی، متغیرها و محاسبات	**۵۴**
۱. آشنایی با تولید ورودی و خروجی و انواع متغیرها در برنامه‌نویسی	۵۵
۲. انجام محاسبات	۶۳
۳. خلاصه نویسی در عبارات محاسباتی	۶۶
۴. مربع مجموع سه رقم	۶۷
۵. نکات تکمیلی در مورد متغیرها	۶۸
فصل چهارم : مدیریت صفحه کلید، صفحه نمایش و آشنایی با ساختارهای تصمیم و تکرار	**۷۲**

۱. نوع کاراکتری	۷۳
۲. مدیریت صفحه کلید	۷۵
۳. ساختار تصمیم if	۷۶
۴. ساختار تصمیم switch	۸۶
۵. ساختار تکرار while	۹۰
فصل پنجم : برنامه نویسی گرافیکی	**۹۶**
۱. ورود به محیط گرافیکی	۹۷
۲. محاسبات در گرافیک: گرفتن سه رأس و آزمودن نامساوی مثلث	۱۰۳
۳. ایجاد طیف رنگ با RGB	۱۰۵
۴. ترسیم میانه‌های مثلث	۱۰۷
۵. استفاده از موس در محیط گرافیکی	۱۰۹
۶. استفاده از اعداد تصادفی	۱۱۷
۷. ترسیم اشیاء توپر	۱۱۹
فصل ششم : مباحث تکمیلی ساختارهای تکرار	**۱۲۴**
۱. ساختار تکرار for	۱۲۵
۲. محاسبه‌ی میانگین n عدد	۱۲۷
۳. استفاده از ساختار تصمیم در ساختار تکرار: یافتن بیشینه بین n عدد	۱۲۸
۴. یک مثال گرافیکی از حلقه‌ی for: رسم n دایره‌ی متحدالمرکز	۱۲۹
۵. چند مثال ریاضیاتی	۱۳۰
۶. محاسبه‌ی سری‌ها و تخمین عدد π	۱۳۲
۷. مثالی از کاربرد while: تجزیه‌ی ارقام یک عدد صحیح	۱۳۴
۸. تشخیص اول یا مرکب بودن عدد	۱۳۸
۹. حلقه‌های تو در تو	۱۳۹
فصل هفتم : آرایه‌ها	**۱۴۷**

۱. متغیرهای اندیس دار	۱۴۸
۲. ترسیم یک n-ضلعی	۱۴۹
۳. مرتب‌سازی حبابی	۱۵۲
۴. مرتب‌سازی انتخابی (*)	۱۵۸
۵. غربال اِراتستن (*)	۱۶۰
۶. اعداد صحیح بزرگ (*)	۱۶۴
۷. رشته‌ها	۱۷۱
۸. آرایه‌های دو بعدی	۱۷۷
فصل هشتم : کار با فایل	**۱۸۰**
۱. حافظه‌ی موقتی و دائمی	۱۸۱
۲. انواع فایل و نحوه‌ی خواندن محتوای فایل‌های متنی	۱۸۲
۳. خواندن از فایل و مرتب سازی داده‌ها و نوشتن خروجی در فایل	۱۸۷
۴. مشخص شدن انتهای فایل	۱۹۰
۵. ذخیره کردن محل نشان‌گر موس	۱۹۱
فصل نهم : برنامه نویسی پیمانه‌ای	**۱۹۵**
۱. مفهوم و مزایای تابع و برنامه‌نویسی پیمانه‌ای	۱۹۶
۲. نحوه‌ی تعریف تابع در زبان C	۱۹۸
۳. نوشتن تابع محاسبه‌ی n!	۲۰۰
۴. نوشتن توابع گرافیکی	۲۰۳
۵. توابع گرافیکی با پارامتر ورودی	۲۰۴
۶. یک تابع ریاضی: بررسی اول بودن یک عدد	۲۰۶
۷. فراخوانی با ارجاع در مقابل فراخوانی با مقدار (*)	۲۰۷
فصل دهم : برنامه نویسی پیشرفته (*)	**۲۱۰**
۱. ساختمان‌ها	۲۱۱

۲. ساختمان‌ها به عنوان پارامتر ورودی تابع	۲۱۳
۳. ساختمان‌ها به عنوان خروجی تابع	۲۱۶
۴. اشاره‌گرها	۲۱۸
۵. اخذ حافظه‌ی پویا از سیستم عامل	۲۲۱
۶. اشاره‌گرها و توابع و نوشتن توابعی با تعداد پارامتر ورودی نامشخص	۲۲۳
۷. برنامه‌نویسی بازگشتی	۲۲۸
ضمیمه اول: مبانی اشکال زدایی	**۲۳۷**
ضمیمه دوم: آشنایی با ساخت پروژه و دستورات گرافیکی	**۲۴۹**
مراجع	**۲۵۷**

مقدمه

به نام خداوند گردون سپهر
فروزنده‌ی ماه و ناهید و مهر

دانش و فناوری کامپیوتر، امروزه به یکی از پیچیده‌ترین و حساس‌ترین مرزهای گستره‌ی دانش بشر تبدیل شده است و شاید بتوان گفت هر جامعه‌ای در این زمینه پیشرو باشد، می‌تواند زمینه‌ی گسترش مرزهای دانش را در سایر علوم فراهم سازد و هر جامعه‌ای از این بخش غافل شود، پیشروی کلان در سایر علوم نیز برایش دشوار خواهد شد. بنابراین، یادگیری علوم این حوزه، در کنار سایر علوم، از ملزومات پیشرفت محسوب می‌شود و هر کس از این امر غافل شود، صرف نظر از رشته‌ای که به آن علاقه دارد، امکان پیشرفت خود را محدود کرده است.

اکنون که به یاری خداوند متعال نگارش این کتاب به پایان رسیده است، بر خود لازم می‌دانم از زحمات کلیه‌ی عزیزانی که مرا در طی نگارش کتاب یاری کردند، از جمله **همسرم**، که کلیه‌ی زحمات صفحه بندی بر عهده‌ی ایشان بود، **جناب آقای فرهاد مقیمی**، که در طراحی جلد و گرافیک بخش‌های مختلف کتاب زحمت زیادی کشیدند و سایر دوستان که با نظرات خود چراغ راه بودند، تشکر کنم. همچنین جا دارد تشکر ویژه‌ای از کلیه‌ی دوستان عزیز در **مرکز ملی پرورش استعدادهای درخشان و دانش پژوهان جوان** که در راه نشر این کتاب زحمت زیادی کشیدند، بنمایم. امیدوارم پایان این کار، آغاز راهی برای استعدادهای کشور و امیدهای آینده‌ی این مرزبوم باشد.

دانش آموخته‌ی سمپاد

عبدالله آراسته

بهار ۱۳۹۰

سخنی با دانش‌آموز

کتابی که پیش رو دارید از ده فصل تشکیل شده است. هر فصل حاوی مطالب مختلفی است که در فهرست مطالب می‌توانید آن‌ها را به‌سرعت بیابید. در کتاب از برخی کادرها استفاده شده که مهم‌ترین آن‌ها «توجه!» است که معمولاً توجه شما را به نکته‌ای مهم در مورد مطلب مورد بحث جلب می‌کند. کادرهای «لوح» برای مشخص نمودن آدرس یک فایل بر روی لوح فشرده همراه کتاب است و «وب» برای مشخص کردن آدرس یک سایت برای دانلود یک فایل و یا خواندن و دنبال کردن یک مبحث مربوط به درس.

کادر «سؤال» که کمتر آن را در طول کتاب می‌بینید، برای مطرح کردن سؤالاتی است که به نظر نگارنده، ذهن خواننده باید به سمت آن معطوف شود. کلیه‌ی شکل‌ها و کدها دارای شماره‌ای هستند که با شماره‌ی فصل شروع می‌شود، مثلاً کد ۵-۳ یعنی کد سوم از فصل پنجم و یا شکل ۷-۲ یعنی شکل دوم از فصل هفتم. کلیه‌ی کدهای نوشته شده در کتاب در لوح فشرده‌ی همراه کتاب موجود است. در اولین کد، در بخش «لوح» آدرس آن داده شده است، برای بقیه‌ی کدها نیز آدرسی مشابه وجود دارد که برای پرهیز از اطناب، در کنار سایر کدها درج نشده است. توصیه می‌شود برنامه‌های نوشته شده در متن درسی کتاب را خوب خوانده و درک کنید، سپس به ایجاد تغییر در بخش‌های مختلف آن اقدام کنید و نترسید! زیرا همواره یک کپی از برنامه‌ی اصلی در لوح فشرده‌ی کتاب در دسترس شماست! پس در ایجاد تغییرات و رفع اشکالات برنامه‌های جدید ایجاد شده **شجاع** و **کوشا** باشید.

در برخی فصول، بخش‌هایی با علامت (*) مشخص شده است که مفهوم اختیاری بودن دارد، یعنی دبیر درس مختار است این بخش‌ها را درس بدهد یا خیر و این تصمیم وابسته به ارزیابی وی از کلاس است. این موضوع در مورد فصل ۱۰ نیز صدق می‌کند (این فصل کلاً ستاره دار است). برای هر فصل تمارینی در سطوح مختلف در نظر گرفته شده که در کتاب دبیر موجود است و به صلاحدید وی به دانش‌آموزان به عنوان تمرین یا پروژه ابلاغ خواهد شد.

امید است پس از اتمام این کتاب، توانایی شما در حل مسایل کامپیوتری افزایش یابد و بتوانید در برخورد با آن‌ها، یک روال منطقی برای رسیدن به راه حل را طی کنید.

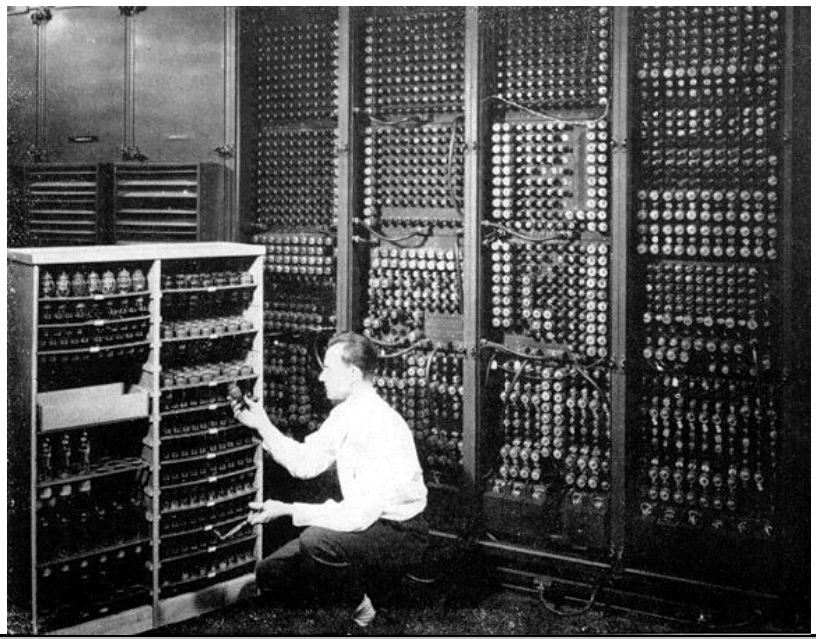
۱

فصل اول
با ما باشید

آشنایی با زبان‌های برنامه نویسی و مفاهیم اولیه

۱. تاریخچه‌ی توسعه‌ی کامپیوترها و زبان‌های برنامه‌نویسی

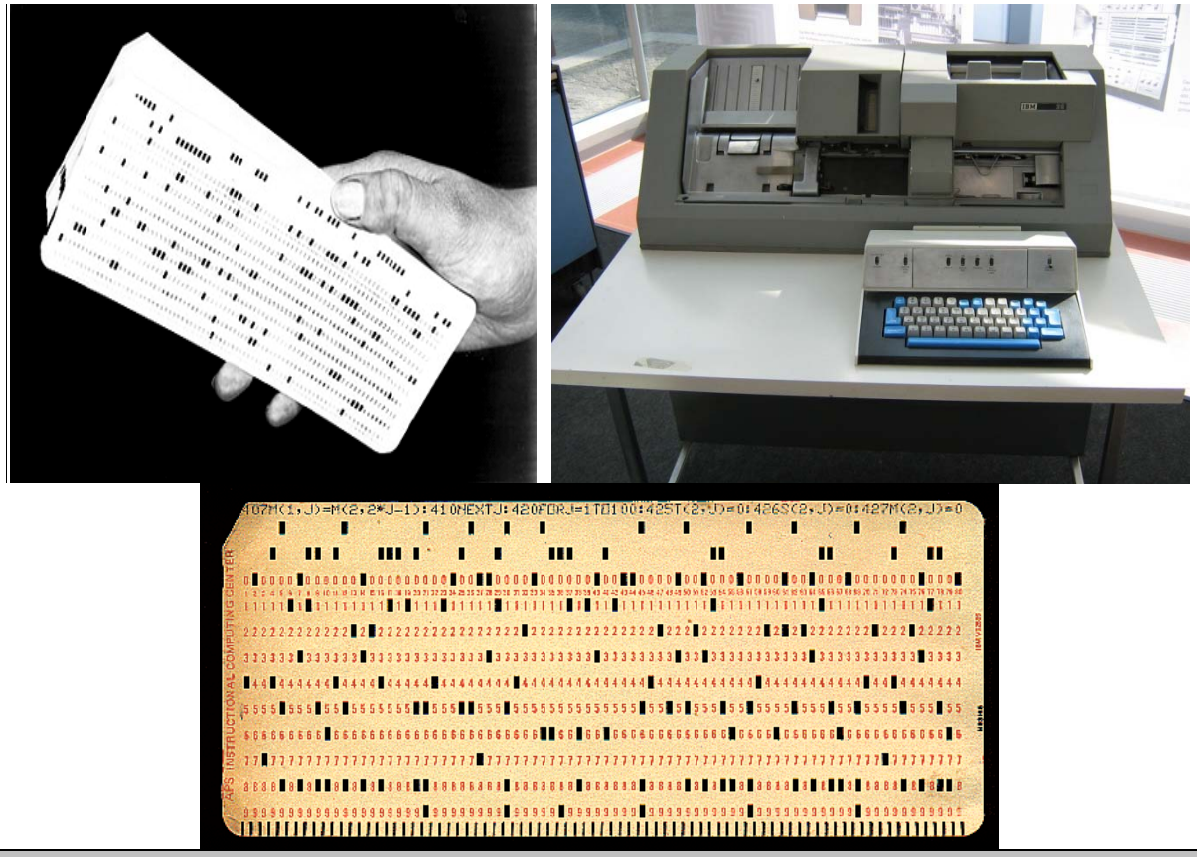
هیچ‌کس نمی‌داند اولین بار چه کسی به فکر ایجاد ماشینی برای انجام محاسبات افتاد، اما تلاش‌های مهمی که در این‌باره انجام شده در تاریخ ثبت شده است. اگر از وسایل مکانیکی ساخته شده برای محاسبات بگذریم، نخستین کامپیوتر الکترونیکی که ساخته شد، اِنیاک[1] بود (شکل ۱-۱). این کامپیوتر الکترونیکی قادر بود برنامه‌های محاسباتی مختلفی را اجرا کند، البته هدف اصلی طراحی آن محاسبه‌ی جدول تیر توپ‌ها و اسلحه‌های پرتابه‌ای ایالات متحده بود. اِنیاک که در سال ۱۹۴۶ توسط محققان دانشگاه پنسیلوانیا ساخته شد ۱۷۴۶۸ لامپ خلأ، ۷۲۰۰ دیود کریستالی، ۱۵۰۰ رله، ۷۰۰۰۰ مقاومت و ۱۰۰۰۰ خازن داشت و مساحتی حدود ۶۳ متر مربع را اشغال کرده و ۱۵۰ کیلو وات توان مصرف می‌کرد! سرعت این کامپیوتر حداکثر تا ۳۸۵ عمل ضرب در ثانیه بود! شاید با خواندن این آمار و ارقام، ضمن آشنا شدن با تاریخچه‌ی کامپیوترها، کمی بیشتر قدر کامپیوترتان را بدانید!



شکل ۱-۱: عکسی از ENIAC (یک تکنیسین در حال تعویض یکی از لامپ‌ها)

[1] Electronic Numerical Integrator And Computer

نحوه‌ی نوشتن برنامه برای اِنیاک، از طریق کارت‌های سوراخ دار[۲] بود که توسط دستگاه‌های کارت خوان IBM خوانده می‌شد (شکل ۱-۲). این کارت‌ها به گونه‌ای، حاوی کد دودویی[۳] دستوراتی بودند که اِنیاک باید اجرا می‌کرد.



شکل ۱-۲: عکس‌هایی از دستگاه کارت خوان IBM و کارت‌های سوراخ دار

پس از اِنیاک تا به امروز پیشرفت‌های فراوانی در توسعه‌ی کامپیوترها و زبان‌های برنامه‌نویسی اتفاق افتاده است، به گونه‌ای که تقریباً در هر خانه یک کامپیوتر شخصی (PC[۴]) وجود دارد و

[2] Card Punch
[3] Binary
[4] Personal Computer

در صورت داشتن اندکی دانش برنامه‌نویسی، هرکسی می‌تواند در خانه‌ی خود اقدام به برنامه‌نویسی و حل مسایل مختلف کند. اولین زبانی که پس از کدهای ماشین (که بسیار شبیه کدهای درج شده بر روی کارت‌های سوراخ دار بود) برای برنامه‌نویسی در نظر گرفته شد زبان اسمبلی[۵] بود. این زبان بسیار ساده‌تر از حفظ کردن کدهای ماشین بود و دستورات پایه‌ای برای انجام عملیات ریاضی و مدیریت حافظه را شامل می‌شد. به عنوان مثال به یک برنامه‌ی ضرب ساده به زبان اسمبلی توجه کنید:

```
mov ax, 14
mov bx, 5
mul bx
```

پس از انجام این دستورات نتیجه‌ی ضرب ۵ در ۱۴ در ax ذخیره می‌شود. همان‌طور که می‌بینید گرچه نسبت به کد ماشین و کارت سوراخ دار فهم این کد آسان‌تر است، اما همچنان برنامه‌نویسی با آن سخت است! طراحان زبان‌های برنامه‌نویسی نیز به این مشکل آگاه بودند و بنابراین اقدام به ایجاد زبان‌هایی کردند که به زبان‌های طبیعی انسان‌ها و زبان‌های محاوره‌ای نزدیک‌تر باشند. به عنوان مثال ضرب فوق که در زبان اسمبلی انجام شد، در چند زبان دیگر آمده است:

```
A = 14
B = 5      } BASIC
C = A*B
```

[5] Assembly

```
A := 14;
B := 5;      } PASCAL
C := A*B;
```

هرچه زبان‌های برنامه‌نویسی به گفتار انسان نزدیک‌تر باشد، اصطلاحاً به آن زبان سطح بالا می‌گویند. بر همین اساس زبان‌های برنامه‌نویسی را به سه دسته تقسیم می‌کنند:

(۱) زبان‌های سطح پایین: زبان‌هایی که مشابه زبان اسمبلی، به سخت افزار نزدیک‌تر بوده و معمولاً اعمال پایه‌ای ریاضی در آن‌ها تعریف شده و انجام عملیات پیچیده در آن‌ها از پیش تعریف نشده است.

(۲) زبان‌های سطح بالا: زبان‌هایی که عموماً به زبان محاوره‌ای نزدیک هستند و فهم آن‌ها با خواندن آن، معمولاً خیلی آسان‌تر از زبان‌های سطح پایین است و معمولاً هیچ ردپایی از سخت افزار در آن‌ها دیده نمی‌شود.

(۳) زبان‌های سطح میانی: زبان‌هایی که هم انعطاف‌پذیری زبان‌های سطح بالا و هم قابلیت ارتباط با سخت‌افزار زبان‌های سطح پایین را دارند.

در بین زبان‌های برنامه‌نویسی، زبان C جزء زبان‌های سطح میانی طبقه‌بندی می‌شود و به همین خاطر، هم در توسعه‌ی نرم‌افزارها و هم جهت ارتباط با سخت‌افزار به طور جدی از آن استفاده می‌شود. در بخش‌های بعدی بیشتر با خصوصیات این زبان برنامه‌نویسی آشنا خواهیم شد.

۲. لزوم فراگیری برنامه‌نویسی

در بخش قبلی در مورد خصوصیات کامپیوتر اِنیاک توضیح دادیم. سؤالی که در ذهن پیش می‌آید این است که چرا باید با صرف هزینه‌های گزاف، یک ماشین محاسبه‌گر ساخت و برای محاسبه به آن برنامه داد؟ پاسخ این است که اولاً محاسبات انسان همواره همراه با خطاست، خصوصاً اگر این محاسبات به دفعات زیاد و پشت سر هم تکرار شود. نکته‌ی مهم‌تر از آن این که بعضاً نتیجه‌ی این محاسبات باید در مواردی نظیر ساخت ابزار یا وسیله‌ای حساس استفاده شود و هرگونه خطا در این محاسبات ممکن است منجر به یک فاجعه شود! دیگر این که با فرض عدم وجود خطای محاسباتی، سرعت ماشین (که موجودی بی‌احساس و خستگی‌ناپذیر است) بسیار فراتر از سرعت انسان در محاسبه است، حتی اگر این ماشین اِنیاک باشد! این موضوع در بسیاری از کاربردهایِ امروزیِ کامپیوترها بیشتر مشهود است. شاید به جرأت بتوان گفت تعداد ضرب و جمع‌هایی که یک کامپیوتر در عرض چند ماه انجام می‌دهد از کل محاسبات عددی یک انسان در طول عمرش فراتر است و این در حالی است که در برخی کاربردها نظیر شبیه‌سازی‌های پیچیده یا یادگیری ماشین[۶] کامپیوترهای پرقدرت برای تولید خروجی، ساعت‌ها به محاسبه می‌پردازند که اگر قرار بود این محاسبات توسط انسان انجام شود معلوم نیست چندین سال طول می‌کشید، و شاید اصلاً دانشمندان چنین مسایلی با بار محاسباتی بالا را کنار می‌گذاشتند.

فایده‌ی یادگیری زبان برنامه‌نویسی و اصولاً هنر برنامه‌نویسی، تنها در سرعت بخشیدن به محاسبات برای رسیدن به جواب پرسش‌های علمی نیست، بلکه در حین تبدیل یک مسأله به یک برنامه که ورودی‌ها و خروجی‌های مشخص دارد و طراحی یک روش حل مسأله یا الگوریتم[۷] برای حل مسأله، فرد با تفکر منطقی و حل قدم به قدم حل مسایل آشنا می‌شود و

[6] Machine Learning
[7] Algorithm

این توانایی نه تنها در حل مسایل در کامپیوتر، که در تحلیل برخی از مسایل روزمره نیز به فرد کمک می‌کند. شاید به همین دلیل است که اغلب افرادی که برنامه‌نویسان خوبی هستند، در تحلیل بسیاری از مسایل، حتی مسایل اجتماعی، فلسفی و ...نسبت به افراد هم‌سطح خود که فاقد این مهارت هستند، قدرت بیشتری دارند.

یکی دیگر از محاسن برنامه‌نویسی به صورت عام و برای علاقه‌مندان به هر رشته‌ای از علم (مثل فیزیک، شیمی، ریاضیات، زیست و ...) این است که امروزه در بسیاری از رشته‌ها، مسایل محاسباتی وجود دارد که لزوماً ابزار یا نرم‌افزار آماده‌ای برای حل آن‌ها وجود ندارد و یا تمامی خواسته‌ها را در مورد یک مسأله‌ی خاص برآورده نمی‌کند. در این حالت چنانچه فرد یک دانش اولیه از برنامه‌نویسی داشته باشد، می‌تواند مسایل علمی خود را حل کند و دیگر احتیاج به سفارش برنامه به شرکت‌های برنامه نویسی و یا شخص دیگری نیست و در وقت و هزینه صرفه‌جویی فراوانی می‌شود.

۳. زبان C و تاریخچه‌ی آن

در بخش‌های قبلی راجع به کامپیوترها و زبان‌های برنامه‌نویسی صحبت شد و اشاره‌ی کوتاهی نیز به زبان C به عنوان زبانی سطح میانی (یعنی زبانی که هم قابلیت ارتباط با سطوح پایین سخت افزاری و هم قابلیت پیاده‌سازی برنامه‌های سطح بالا را دارد) شد، در این بخش نحوه‌ی به وجود آمدن زبان C و گسترش دامنه‌ی کاربرد آن در علوم و صنعت، با مروری بر تاریخچه‌ی آن مطرح خواهد شد. زبان برنامه نویسی C در آزمایشگاه بل، یکی از پر افتخارترین مراکز علمی دنیا متولد شد. خالق این زبان برنامه‌نویسی دنیس ریچی[۸] است که در

[8] Dennis Ritchie

این مرکز تحقیقاتی مشغول کار بر روی توسعه‌ی سیستم‌عامل‌ها و زبان‌های برنامه‌نویسی بود. خلق زبان C توسط دنیس ریچی و نقش آن در توسعه‌ی سیستم‌عامل یونیکس[9] در کنار کِن تامپسون[10]، او را در زمره‌ی افراد پیشرو در زمینه‌ی محاسبات جدید[11] قرار داد.

وب سایت آزمایشگاه بل:

http://www.bell-labs.com

وب سایت دنیس ریچی در آزمایشگاه بل:

http://cm.bell-labs.com/who/dmr

همچنین دنیس ریچی به همراه برایان کرنیگان[12] در سال ۱۹۷۸ اولین کتاب در مورد زبان C را که به صورت جامع به ابعاد مختلف این زبان برنامه‌نویسی پرداخته بود و در دسترس عموم قرار گرفت، نوشتند. در سال ۱۹۸۸، دومین ویرایش این کتاب که شامل آخرین تغییرات زبان C و شرح برخی کتابخانه‌های استاندارد[13] بود، توسط آن‌ها منتشر شد. این ویرایش کتاب تا سال ۲۰۱۰ به بیش از بیست زبان ترجمه شده و در بسیاری از مراکز آموزشی تدریس شده و می‌شود. متن این کتاب اندکی سنگین است، اما ایجازهای زیبایی در نوشتن برنامه‌های آن به زبان C به کار گرفته شده و مطالعه‌ی آن به تمامی علاقه‌مندان به برنامه‌نویسی توصیه می‌گردد.

[9] Unix
[10] Ken Thompson
[11] Modern computing
[12] Brian Kernighan
[13] Standard Libraries

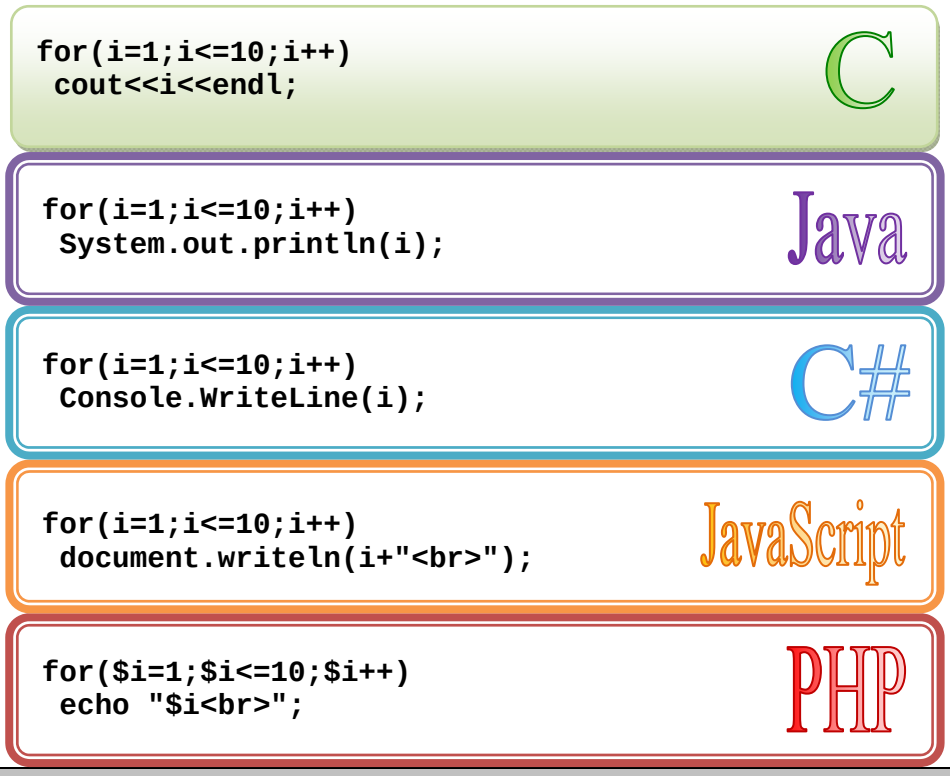
لوح

نسخه‌ی الکترونیکی کتاب The C programming language نوشته‌ی رابرت کرنیگان و دنیس ریچی در لوح فشرده‌ی همراه کتاب در آدرس زیر موجود است:

CDROM:\Resources\Books\K&R.pdf

پس از به وجود آمدن زبان C تا به امروز، هیچ‌گاه آموزش و به‌کارگیری زبان C به عنوان ابزاری در توسعه‌ی بسیاری از سیستم‌های سخت‌افزاری و نرم‌افزاری، قطع نشده و به هیچ‌وجه دستور زبان[۱۴] به کار رفته در این زبان برنامه‌نویسی منسوخ نشده است، بلکه برعکس زبان C در بسیاری از حوزه‌ها نفوذ کرده و امروز خانواده‌ی بزرگی از زبان‌های برنامه‌نویسی را زبان‌های «شبیه C[۱۵]» می‌نامند. از جمله‌ی آن‌ها می‌توان به Java، PHP، JavaScript و C# اشاره کرد. شباهت برخی از این زبان‌ها به C به قدری است که اگر فردی تنها زبان C را بلد باشد و نگاهی گذرا به متن برنامه‌ی نوشته شده با این زبان‌ها بیاندازد، متوجه نخواهد شد که برنامه با زبانی غیر از C نوشته شده است. به عنوان نمونه در شکل ۱-۳ قطعه کدهای مربوط به چاپ اعداد از ۱ تا ۱۰ را به زبان‌های C، Java، Java script، C# و PHP می‌بینید و می‌توانید شباهت بیش از اندازه‌ی چهار برنامه‌ی آخر را به برنامه‌ی اول که به زبان C نوشته شده درک کنید:

[14] Syntax
[15] C-Like



شکل ۱-۳: قطعه کدهای مربوط به چاپ اعداد از ۱ تا ۱۰ به زبان‌های C، Java، Java script، C# و PHP

زبان C به قدری در صنعت نفوذ کرده که برخی پردازنده‌های پرکاربرد (نظیر AVR ها) طوری ساخته شده‌اند که برنامه‌نویسی به زبان C، بهینه‌ترین حالت برای برنامه‌ریزی آن‌هاست. به عنوان نمونه می‌توان به نرم‌افزار Code Vision برای برنامه‌نویسی میکروکنترلرهای AVR اشاره کرد.

علاوه بر تمام موارد گفته شده، C به عنوان ابزار اصلی توسعه‌ی سیستم‌های عامل و ابزارهای متن باز[۱۶] به کار می‌رود و هسته[۱۷]ی سیستم‌عامل‌های لینوکس[۱۸] و نرم‌افزارهای کاربردی نصب شونده بر روی آن‌ها بر پایه‌ی زبان C است که باعث رویکرد میلیون‌ها علاقه‌مند به

[16] Open Source
[17] Kernel
[18] Linux

توسعه‌ی نرم‌افزارهای متن باز در دنیا به زبان C شده است. امروزه اهمیت نرم‌افزارهای متن باز و نقش اساسی آن‌ها در بین ابزارهای مورد استفاده‌ی کاربران بر کسی پوشیده نیست.

وب

در سایت زیر می‌توانید نرم افزار Code Vision و برخی برنامه‌هایی که با آن نوشته شده ملاحظه کرده و شباهت آن را با سایر برنامه‌های C استاندارد بسنجید:

http://www.codevision.be/

در انتهای این بخش این نکته نیز باید ذکر شود که زبان برنامه‌نویسی C++ هیچ تفاوت مبنایی با C ندارد و همان زبان C است که قابلیت‌های برنامه‌نویسی شئ‌گرا[19] به آن اضافه شده است. C++ توسط بیران استراستروپ[20] در سال ۱۹۷۹ در آزمایشگاه بل به عنوان پروژه‌ی بهبود زبان C توسعه یافت و در ابتدا C به انضمام کلاس‌ها[21] نامیده می‌شد و در سال ۱۹۸۳ به C++ تغییر نام داد. استراستروپ در مورد C++ کتابی نیز نوشته است که یکی از مراجع اصلی در این زمینه است. البته یکی دیگر از بهترین کتاب‌ها در این زمینه نیز نوشته‌ی هربرت شیلد[22] است که با مثال‌های متنوع به آموزش ساده و روان C++ پرداخته است. البته از ایرادهای این گونه کتاب‌ها می‌توان به حجم بالا و نداشتن مثال‌های مناسب برای دانش‌آموزان اشاره کرد که پیگیری و دنبال کردن آن را فقط محدود به افراد علاقه‌مند در این زمینه می‌کند.

[19] Object Oriented Programming
[20] Biorne Straustrup
[21] C with classes
[22] Herbert Schildt

وب

آدرس وب‌سایت شخصی استراستروپ:

http://www.straoustrup.com

http://www2.research.att.com/~bs/homepage.html

لوح

نسخه‌ی الکترونیکی کتاب C++ نوشته‌ی استراستروپ در لوح فشرده‌ی همـراه کتاب در آدرس زیر موجود است:

CDROM:\Resources\Books\bsC++.pdf

همچنین کتاب C++ نوشته‌ی هربرت شیلد نیز در آدرس زیر قرار دارد:

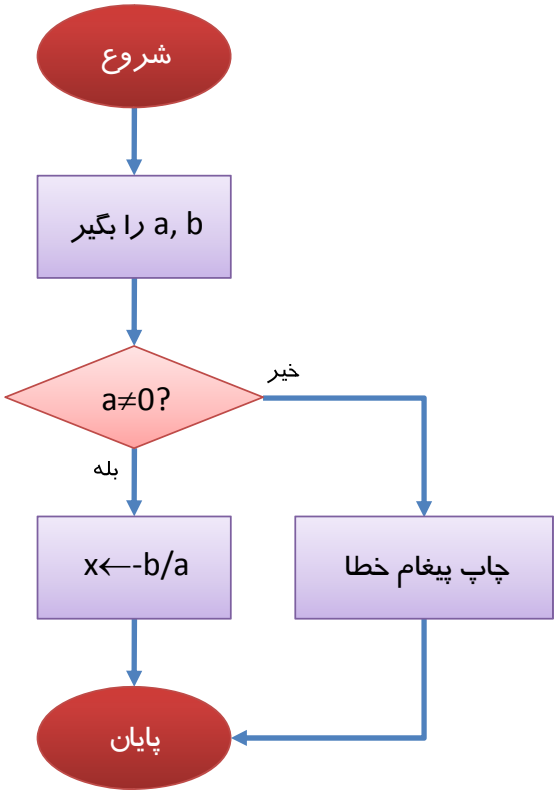
CDROM:\Resources\Books\hsC++.pdf

۴. گام‌های برنامه‌نویسی

یکی دیگر از مشکلات اساسی در نوشتن یک برنامه این است که عمدتاً افراد نمی‌دانند پس از برخورد با یک مسأله در دنیای واقعی، چطور آن را به شکلی در بیاورند که نوشتن برنامه‌ی آن آسان‌تر شود. بسیاری از افراد حتی مسایل ساده را که در حالت عادی به سادگی حل می‌کنند نمی‌توانند بر روی کامپیوتر پیاده‌سازی و اجرا نمایند. به عنوان مثال معادله‌ی درجه اول $ax+b=0$ را در نظر بگیرید، تقریباً هر دانش‌آموزان اول دبیرستانی جواب معادله که $x=-\frac{b}{a}(a\neq 0)$ را می‌داند، اما اگر بخواهد برنامه‌ای بنویسد که چنین معادله‌ای را حل کند، ممکن است نداند که از کجا باید شروع کند و یا اصولاً نداند گام‌های اصلی برای حل

چنین مسأله‌ای توسط کامپیوتر چیست؟ یک فرق اساسی حل مسایل توسط کامپیوتر با انسان این است که مسایلی که به کامپیوتر داده می‌شود با داده‌ها و دستورات بیان می‌شود، حال آن که مسایلی که برای انسان مطرح می‌شود با خواسته‌ها و قوانین مطرح می‌شود. به عنوان مثال همین مسأله‌ی معادله‌ی درجه اول را در نظر بگیرید، چنانچه یک معلم ریاضی بخواهد یک مسأله‌ی معادله‌ی درجه اول به دانش‌آموزان خود بدهد، به طور مثال آن را به صورت $5x+10=0$ مطرح می‌کند و دانش‌آموزان مطابق قوانینی که قبلاً به آن‌ها آموزش داده شده می‌دانند که تساوی در همه حال باید برقرار باشد و با کسر 10 از دو طرف تساوی و تقسیم دو طرف بر 5 به جواب $x=-2$ می‌رسند. اما در مورد حل این مسأله توسط کامپیوتر اوضاع کمی متفاوت است. کامپیوتر مانند انسان نمی‌تواند به یادگیری قواعد و حل مسایل بپردازد و باید قدم به قدم برای آن مشخص کرد که دقیقاً چه کار باید بکند؟ به عنوان مثال نمی‌توان به کامپیوتر تصویر یک معادله را نشان داد و انتظار حل آن را از آن داشت! بلکه باید به شیوه‌ی دیگری عمل کرد. ابتدا باید در هر مسأله مشخص شود ورودی‌های مسأله چیست؟ ورودی‌های مسأله همان‌هایی هستند که ما در ریاضی به آن‌ها معلومات می‌گوییم و قرار است از روی آن‌ها و با استفاده از قوانین مشخص، مجهولات را به دست آوریم. همچنین پس از به دست آوردن مجهولات، باید آن‌ها را به عنوان خروجی برنامه به کاربر اعلام کنیم. عناصر اصلی در نوشتن یک برنامه برای حل مسأله توسط کامپیوتر ورودی‌ها و خروجی‌ها هستند. به عنوان مثال در معادله‌ی درجه اول $ax+b=0$ ورودی‌ها a,b هستند، یعنی باید مقدار آن‌ها مشخص باشد تا بتوان جواب نهایی را محاسبه کرد. خروجی نیز همان x است. اما چه چیزی از روی ورودی، خروجی را می‌سازد؟ پاسخ این پرسش همان برنامه‌ای است که ما می‌نویسیم، یعنی ربط دادن خروجی به ورودی‌ها، وظیفه‌ی برنامه است. در برخی موارد مثل همین مسأله‌ی معادله‌ی درجه اول، ارتباط خروجی به ورودی به سادگی به دست می‌آید، اما

در برخی مسایل دیگر، برای تولید خروجی مناسب از روی ورودی‌ها، به تعمق و تفکر بیشتر احتیاج است. در بسیاری از موارد به دست آوردن گام‌هایی که باید به ترتیب توسط کامپیوتر اجرا شوند، مستلزم صرف ساعت‌ها وقت و بعضاً محاسبات زیاد است. برای روشن‌تر شدن موضوع، در شکل (۱-۴) نمودار گردشی[۲۳] حل مسأله‌ی معادله‌ی درجه اول آمده است:



شکل ۱-۴: نمودار گردشی حل مسأله‌ی معادله‌ی درجه اول

همان‌طور که ملاحظه می‌شود، این نمودار به خوبی نحوه‌ی گرفتن ورودی از کاربر و تولید خروجی را با شرط‌های مناسب $(a\neq0)$ نمایش می‌دهد. نمودار گردشی یک نوع از روش‌های

[23] Flow Chart

بازنمایی حل مسایل به روش کامپیوتری و گام به گام است که در فصل بعدی کتاب به تفصیل به آن پرداخته شده است. روش‌های دیگری نیز برای بازنمایی روش حل مسایل توسط کامپیوتر وجود دارد، نظیر توصیف گام به گام روش حل مسأله یا همان الگوریتم[۲۴] که یک نمونه از آن برای حل مسأله‌ی معادله‌ی درجه اول در شکل ۱-۵ آمده است.

(۱) a و b را بگیر

(۲) اگر $a\neq0$ است، x را برابر $-b/a$ قرار بده و به (۴) برو در غیر این صورت به (۳) برو

(۳) پیغام خطایی مبنی بر صفر بودن a چاپ کن و به (۵) برو

(۴) x را به عنوان جواب چاپ کن

(۵) پایان الگوریتم

شکل ۱-۵: توصیف گام به گام روش حل مسأله برای حل مسأله‌ی معادله‌ی درجه اول

یک نکته که در آزمون درستی یا نادرستی الگوریتمی که ارائه می‌دهیم به کار می‌آید، استفاده از مثال‌های مناسب است. به عنوان مثال اگر $a=10$ و $b=40$ باشد، جواب راه حل ما $-\frac{b}{a}$ یا -4 خواهد بود که از روش ریاضی و حل دستی نیز همین به دست می‌آید، بنابراین راه حل ما درست بوده است. البته شاید درستی الگوریتم در مورد مسأله‌ی معادله‌ی درجه اول بسیار ساده و واضح باشد، اما در مورد بسیاری از مسایل چنین نیست و این مثال زدن‌های عددی و آزمودن الگوریتم در طراحی و تصحیح آن، نقش به سزایی دارند. در پایان این بخش ذکر این نکته ضروری است که با طی تمامی این مراحل، نمی‌توان مطمئن بود که الگوریتم به دست

[24] Algorithm

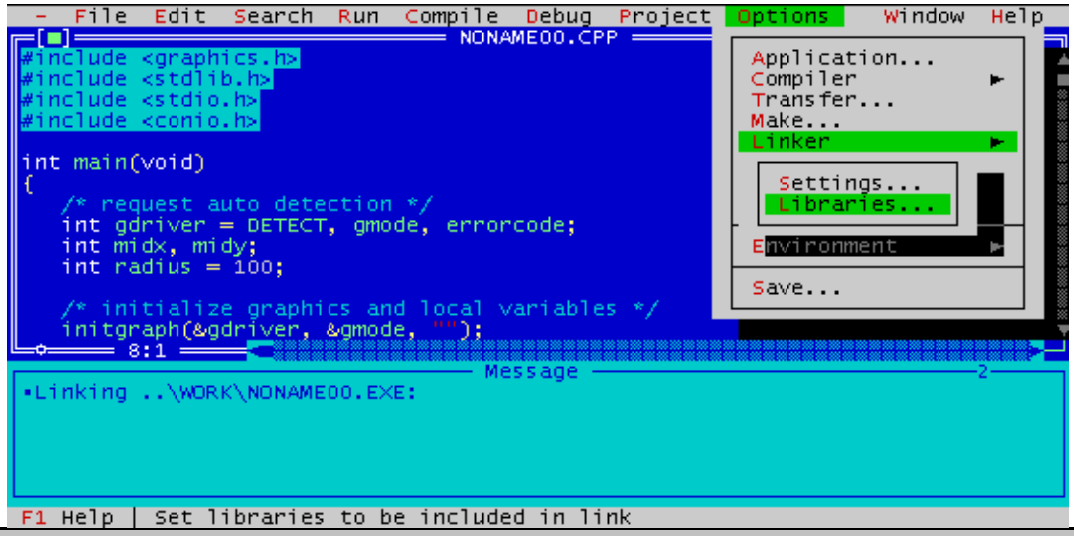
آمده صحیح باشد و تنها راه آزمودن آن، این است که برنامه را نوشته به ازای ورودی‌های مختلف اجرا کنیم تا ببینیم برنامه به درستی کار می‌کند یا خیر؟ اما این که برنامه را چطور بنویسیم و اجرا کنیم موضوع بخش بعدی است.

۵. آشنایی با انواع ویرایشگر[۲۵] ها و مقایسه‌ی آن‌ها

برای اینکه برنامه‌ی نوشته شده به هر زبان برنامه‌نویسی، به شکل قابل اجرا برای کامپیوتر، یعنی همان کد ماشین در بیاید، باید متن برنامه، به زبان ماشین ترجمه[۲۶] شود. برای همین منظور از برنامه‌هایی به اسم مترجم[۲۷] استفاده می‌شود. هر زبان برنامه‌نویسی مترجم مخصوص به خودش را دارد. البته ممکن است برای یک زبان نظیر C، چندین نسخه از مترجم‌های مختلف که تولید شرکت‌های مختلف برنامه‌نویسی است موجود باشد که معمولاً خود مترجم‌ها تفاوت عمده‌ای با هم ندارند. اما آنچه که در مورد ابزارهای مختلف برنامه‌نویسی متفاوت است، وجود ویرایشگرهای مختلف برای یک زبان برنامه نویسی است. گرچه لفظ ویرایشگر باید به برنامه‌هایی اطلاق شود که تنها به ویرایش متن برنامه می‌پردازند، اما امروزه به محیط‌های مجتمع توسعه[۲۸] برنامه‌ها نیز اصطلاحاً ویرایشگر می‌گویند. محیط‌های مجتمع توسعه یا همان IDE ها، ابزارهای مجتمع برای نوشتن، ویرایش، ترجمه و اشکال‌زدایی برنامه‌هایی که نوشته می‌شود هستند. برای هر زبان برنامه‌نویسی IDE های مختلفی وجود دارد و زبان C نیز از این قاعده مستثنی نیست. از جمله IDE های ساده‌ی زبان C می‌توان به Turbo C++ اشاره کرد (شکل ۱-۶) که یک برنامه‌ی قوی در زمان خودش محسوب می‌شد. خصوصیت این

[25] Editor
[26] Compile
[27] Compiler
[28] Integrated Development Environment (IDE)

IDE سادگی کار با آن در نوشتن برنامه‌هاست به طوری که یادگیری برنامه‌نویسی با آن به سادگی امکان‌پذیر است. اما اشکال عمده‌ی این IDE تحت DOS بودن و قدیمی بودن آن است که در کنار محاسن آن کار کردن با آن را خصوصاً به عنوان یک ابزار آموزشی سخت می‌کند، زیرا با سیستم‌عامل‌های جدید نظیر ویندوز ویستا و ویندوز 7 به درستی کار نمی‌کند. این مسأله در مورد نوشتن برنامه‌های گرافیکی بیشتر مشهود بوده و معمولاً خروجی مناسب تولید نمی‌شود و خطاهای سیستم‌عاملی پی‌درپی، عملاً نوشتن برنامه را امکان ناپذیر می‌کند.



شکل ۱-۶: محیط Turbo C++

یک راه حل که برای حل معضل چنین برنامه‌هایی پیشنهاد شده، استفاده از نرم افزار DOS Box است که کار آن شبیه‌سازی محیط DOS برای برنامه‌های قدیمی DOS نظیر Turbo C++ در محیط‌های جدید ویندوز است. اما این برنامه نیز کارکرد درست صد در صد نداشته و در بسیاری موارد اجرای آن با شکست مواجه شده است، مخصوصاً در مورد سیستم عامل ویندوز 7.

از دیگر محدودیت‌های Turbo C++ می‌توان به محدودیت استفاده از حافظه اشاره کرد که موقع کار با آرایه‌های بزرگ، برنامه‌نویس را دچار مشکل می‌کند. همچنین امکانات گرافیکی مثل استفاده از کل جدول رنگ[۲۹] در دسترس در ویندوز و یا بارگذاری و نمایش تصاویر یا استفاده از موس، در Turbo C++ به صورت پیش‌فرض وجود ندارد و تعداد رنگ‌ها در آن محدود به ۱۶ رنگ و یا با استفاده از دستورات خاص تا ۲۵۶ رنگ است، در حالی که تقریباً همه‌ی مانیتورها و کارت‌های گرافیکی روز، از حداقل ۱۶ میلیون رنگ پشتیبانی می‌کنند!

یک راه برداشتن این محدودیت‌ها استفاده از ابزارهای برنامه‌نویسی تحت ویندوز است، اما باید در تفسیر این جمله دقت زیادی کرد. بسیاری از برنامه‌نویسان این عبارت را به معنای برنامه‌نویسی تحت ***معماری ویندوز*** و با ابزارهایی نظیر Microsoft Visual Studio یا Borland C++ Builder و یا Borland Delphi می‌دانند. اگر چنین برداشتی از این جمله شود، باید متذکر شد شروع برنامه‌نویسی معمولاً باید با برنامه‌نویسی ترتیبی[۳۰] با اجرای خط به خط باشد. یادگیری برنامه‌نویسی ترتیبی خود مستلزم زمان زیادی است، حال اگر این مسأله با عنوان کردن معماری ویندوز و مفاهیمی نظیر رخداد[۳۱] ها، اشیاء، برنامه‌نویسی رویدادگرا[۳۲] و... ترکیب شود، باعث گیج شدن اغلب افرادی می‌شود که می‌خواهند برنامه‌نویسی را شروع کنند. در بسیاری از کتب برنامه‌نویسی با معماری ویندوز نیز فصول زیادی در ابتدای کتاب، به مطرح کردن برنامه‌نویسی ترتیبی، با رویکرد آموزش دستور زبان مربوطه اختصاص داده می‌شود. بنابراین استفاده از چنین ابزارهایی، گرچه محدودیت‌های گفته شده را برطرف می‌سازد، اما هدف اصلی که آموزش همگانی با کم‌ترین هزینه‌ی زمانی است را با تردید جدی رو به رو می‌کند.

[29] Palette
[30] Sequential
[31] Event
[32] Event Oriented programming

راه حلی که پیشنهاد می‌شود استفاده از یک ابزار بینابین است که هم تحت ویندوز اجرا شده و محدودیت‌های حافظه‌ای، گرافیکی و... برنامه‌های تحت DOS نظیر Turbo C++ را نداشته و هم این که فاصله‌ی زیادی با مدل برنامه‌نویسی ترتیبی محیط‌هایی نظیر Turbo C++ را نداشته باشد تا فراگیری آن دشوار نشود. در بین ابزارهای برنامه‌نویسی، یک IDE با نام Dev C++ وجود دارد که تقریباً تمامی خصوصیات ذکر شده را دارد. این ابزار برنامه‌نویسی بر روی جدیدترین سیستم‌عامل‌ها نیز بدون مشکل نصب می‌شود.

اصل این برنامه به صورت متن باز و تحت لینوکس است که نسخه‌هایی از آن برای ویندوز نیز نوشته شده و به راحتی و به صورت مجانی از اینترنت قابل دریافت و نصب است. متن باز بودن این برنامه باعث شده است که به روز[۳۳] شدن آن سریع بوده و با سیستم عامل‌های جدید سازگار باشد، همچنین افرادی که به برنامه‌نویسی تحت سیستم عامل لینوکس علاقه دارند می‌توانند نسخه‌های تحت لینوکس این نرم افزار را دریافت و نصب کنند.

وب

آدرس وب‌سایت اصلی نرم‌افزار Dev C++:

http://www.bloodshed.net/dev/devcpp.html

البته با جستجو در گوگل نیز می‌توان سایت‌های بسیار زیادی برای دانلود آخرین نسخه‌ی این نرم افزار پیدا کرد.

[33] Update

لوح

یک نسخه از فایل قابل نصب نرم افزار Dev C++ در لوح فشـرده و در آدرس زیر قرار دارد:

CDROM:\IDE\ Dev C++

همچنین راهنمای نحوه‌ی نصب و مراحل آن در آدرس زیر موجود است:

CDROM:\Resources\Others\InstallDevcpp.pdf

البته IDE های مشابه دیگری نظیر Code Blocks نیز وجود دارد که تفاوت‌هایی جزئی با Dev C++ دارد و معمولاً انتخاب بین IDE هایی نظیر Dev C++ و Code Blocks بیشتر سلیقه‌ای است و تفاوت‌های بنیادینی بین آن‌ها وجود ندارد. در این کتاب مبنا Dev C++ است، اما با فراگیری Dev C++ و با کمی تلاش بیشتر، می‌توان به راحتی با Code Blocks نیز کار کرد.

وب

آدرس وب‌سایت اصلی نرم‌افزار Code Blocks :

http://www.codeblocks.org/downloads/26

لوح

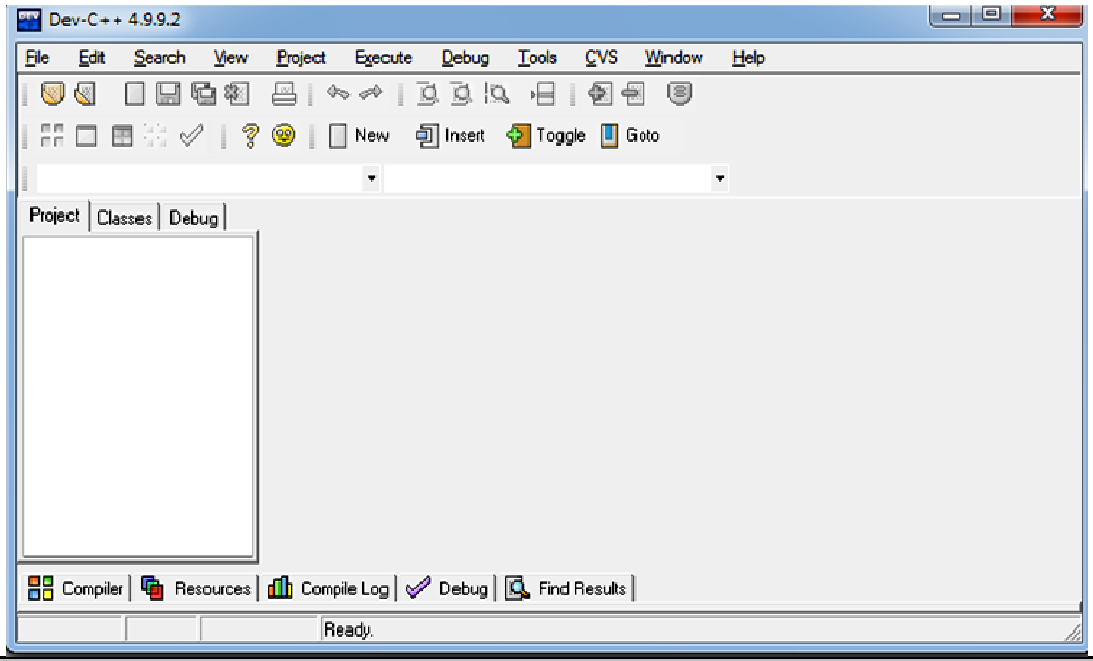
یک نسخه از فایل قابل نصـب نـرم‌افـزار Code blocks در لـوح فشـرده و در آدرس زیر قرار گرفته است.

CDROM:\IDE\Code Blocks

در ادامه‌ی بحث و در بخش بعدی، به مرور امکانات محیط Dev C++، پرداخته خواهد شد.

۶. آشنایی با محیط برنامه‌نویسی Dev C++

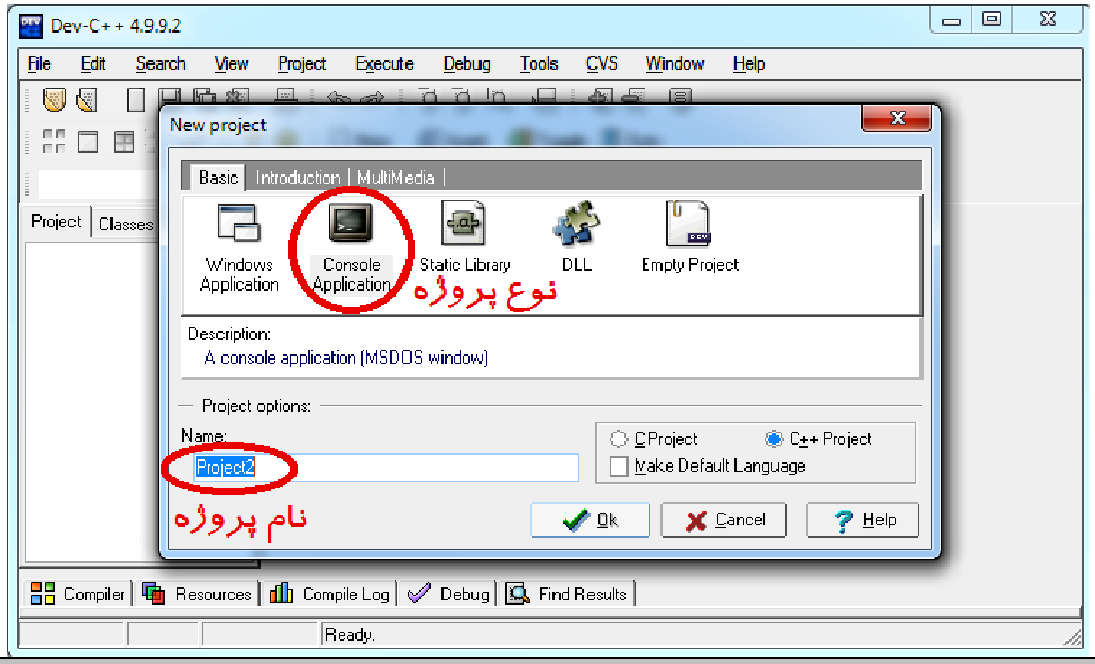
پس از نصب و اجرای برنامه (که راهنمای مرحله به مرحله‌ی آن در لوح فشرده‌ی همراه کتاب آمده) به محیط Dev C++ برخورد خواهید کرد (شکل ۱-۷).



شکل ۱-۷: محیط Dev C++

در همین محیط است که باید برنامه‌ها نوشته‌شده، اجرا شوند و در صورت لزوم مورد اشکال‌زدایی قرار بگیرند. برای ایجاد یک برنامه‌ی ساده، کافی است از منوها File→New→Source File انتخاب شود و یا از کلیدهای ترکیبی Ctrl+N استفاده شود. سپس می‌توان در فایل ایجاد شده برنامه‌ی مورد نظر را تایپ کرده و اجرا نمود. می‌توان در یک زمان چند فایل برنامه‌ی باز در Dev C++ داشت و هر کدام را که مورد نظر است

انتخاب و سپس ویرایش یا اجرا نمود. همچنین می‌توان از منوها File→New→Project را انتخاب کرد که در این صورت پنجره‌ای نمایش داده می‌شود (شکل ۱-۸) که از بین انواع پروژه‌ها باید یک کدام انتخاب شود و اسم آن نیز در قسمت مشخص شده وارد شود.

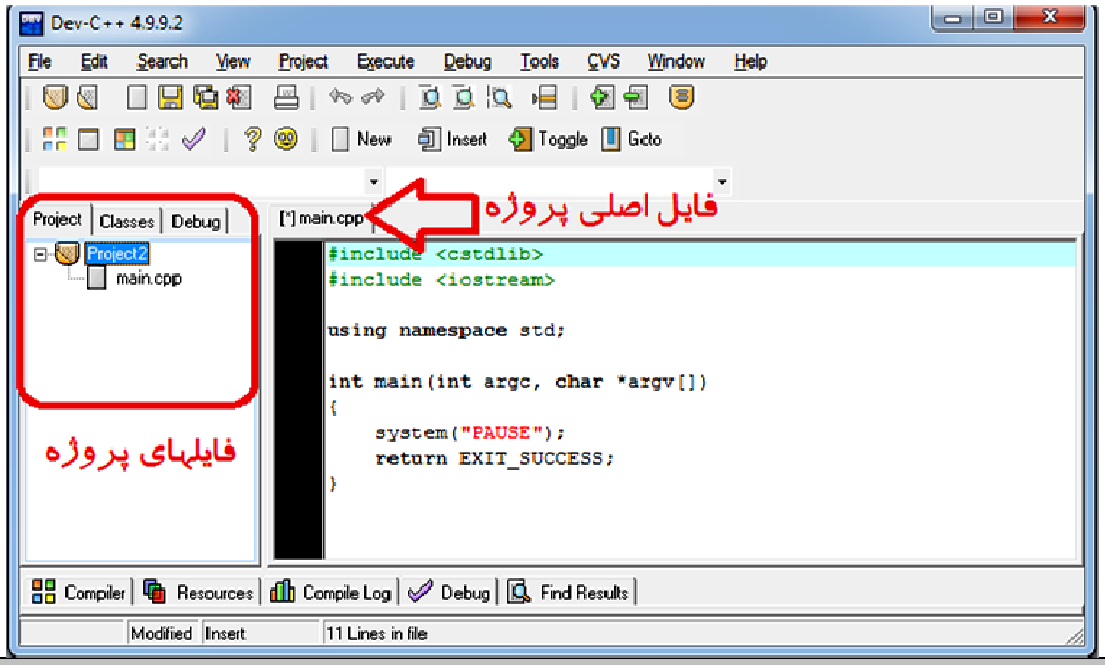


شکل ۱-۸: ایجاد یک پروژه‌ی جدید در Dev C++

سپس پروژه‌ی مورد نظر ایجاد شده و باید آن را در جایی ذخیره کرد. توصیه می‌شود همواره برای هر پروژه یک فولدر جدا ساخته و همه‌ی فایل‌های مربوط به آن پروژه را در آن فولدر ذخیره کنید.

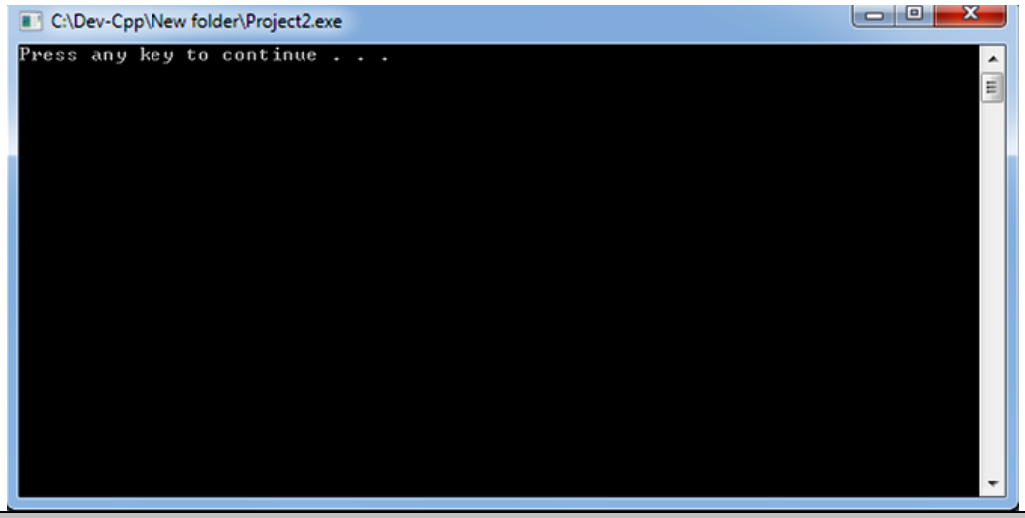
پس از اتمام ایجاد پروژه، نمودار درختی پروژه و فایل‌های مربوط به آن در قسمت سمت چپ برنامه‌ی Dev C++ نمایش داده می‌شود (شکل ۱-۹) و فایل اصلی پروژه که معمولاً با نام main.cpp ساخته می‌شود، در بخش نمایش فایل‌ها نمایش داده می‌شود. چنانچه بخواهید این

پروژه اجرا شود باید کلید F9 را زده و یا از منوها Execute→Compile & Run را انتخاب کنید و پس از انجام این کار پنجره‌ای باز می‌شود که از شما می‌خواهد برنامه‌ی اصلی یا همان main.cpp را ذخیره کنید.



شکل ۱-۹: نمودار درختی فایل‌های پروژه و فایل اصلی پروژه

چنانچه main.cpp را ذخیره کرده و اگر در ابتدا console application را به عنوان نوع پروژه انتخاب کرده باشید (در پنجره‌ای که در شکل ۱-۸ آمده) پروژه اجرا شده و خروجی آن نمایش داده می‌شود (شکل ۱-۱۰).



شکل ۱-۱۰: خروجی یک پروژه‌ی Console Application ساده

توجه!

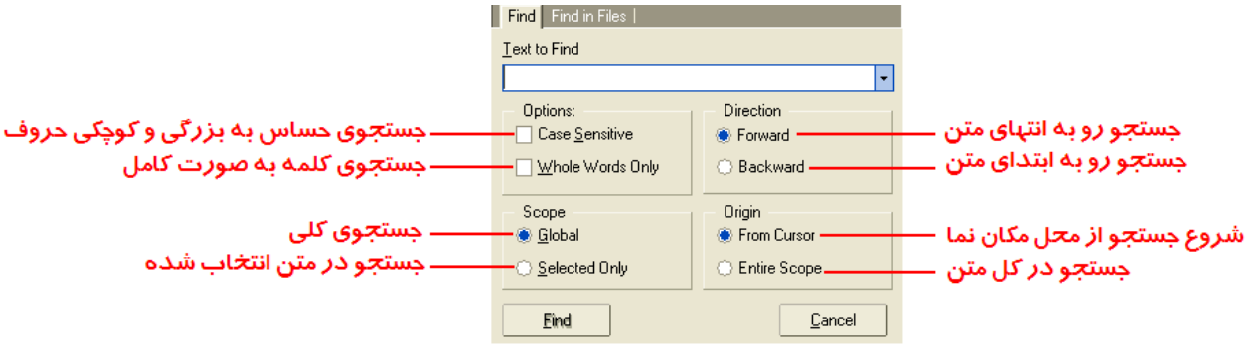
چنانچه پروژه‌ای باز باشد، تا زمانی که آن پروژه باز باشد، هر فایل جدیدی که ایجاد شود و یا هر فایلی که باز است انتخاب شود و سپس برای اجرای آن دکمه‌ی F9 زده شود، تنها همان پروژه اجرا می‌شود. برای جلوگیری از این کار باید از منوها File→Close Project انتخاب شود تا پروژه‌ی فعلی بسته شود. این اتفاق به خاطر آن است که پروژه نسبت به سایر فایل‌ها برای اجرا اولویت دارد. این قضیه در رابطه با فایل‌های ساده‌ی برنامه برقرار نیست، یعنی اگر پروژه‌ای باز نباشد، هر فایلی که انتخاب شده و در حال نمایش است، اجرا می‌شود.

لازم به تذکر است برای عملیاتی که زیاد تکرار می‌شود، نظیر اجرا کردن برنامه و یا ایجاد برنامه‌های جدید، دکمه‌هایی در نوار ابزار برنامه گنجانده شده. در شکل ۱-۱۱ برخی از این دکمه‌ها نمایش داده شده‌اند.

شکل ۱-۱۱: برخی دکمه‌های پرکاربرد در نوار ابزار برنامه‌ی Dev C++

از دیگر امکانات بسیار مفید در برنامه‌نویسی، جستجو و جستجو و جایگزینی هستند. فرض کنید در فایلی به دنبال کلمه‌ی int هستید، در این صورت کافی است تا از منوها Search→Find را انتخاب کرده و یا از کلیدهای ترکیبی Ctrl+F استفاده کنید تا کادر شکل ۱-۱۲ ظاهر شود. در این شکل توضیحات کافی در مورد گزینه‌های پر کاربرد داده شده است. برای جستجوی مورد بعدی، کافیست دکمه‌ی F3 زده شود و یا از منوها Search→Search Again و یا از نوار ابزار دکمه مربوطه انتخاب شود. عملیات جستجو و جایگزینی نیز مشابه جستجو است با این تفاوت که پس از یافتن عبارتی، آن را با عبارت دیگری جایگزین می‌کند.

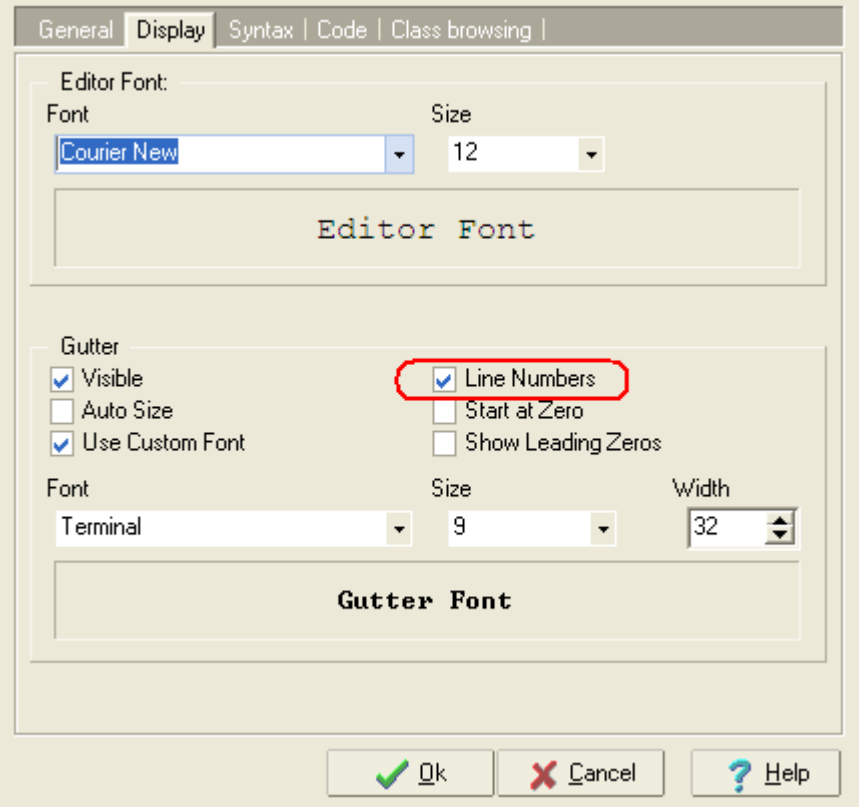
از امکانات مفید دیگر منوی Search، Goto line است که با Ctrl+G نیز در دسترس است و زمانی که خطوط برنامه زیاد باشد مفید است و می‌توان با استفاده از آن به خطی خاص از برنامه رفت.



شکل ۱-۱۲: کادر تبادلی جستجو در Dev C++

چنانچه بخواهید شماره‌ی خطوط برنامه نمایش داده شود، باید از منوها Tools→Editor Options را انتخاب کرده و از پنجره‌ای که ظاهر می‌شود به قسمت Display رفته و جلوی گزینه‌ی Line Numbers، تیک بزنید (شکل ۱-۱۳). همچنین در همین قسمت می‌توانید نوع قلم و سایز آن را که برای نمایش و ویرایش برنامه‌ها استفاده می‌شود، تعیین کنید.

برای مشخص کردن نوع رنگ و حالتِ قلمِ نوشتارهایِ متفاوتِ موجود در برنامه نیز می‌توانید پس از انتخاب Tools→Editor Options، به قسمت Syntax رفته و هرکدام از رنگ‌ها را که می‌خواهید عوض کنید و یا از تنظیمات از پیش آماده شده استفاده کنید (شکل ۱-۱۴). برای آن که آنچه در کتاب می‌بینید با محیط Dev C++ که در آن تایپ می‌کنید هماهنگ باشد، تنها تغییری که نسبت به حالت پیش‌فرض باید بدهید تغییر Forground مربوط به کاراکتر به قرمز است (شکل ۱-۱۴).



شکل ۱-۱۳: برخی تنظیمات ویرایشگر متن Dev C++ و نحوه نمایش شماره خطوط

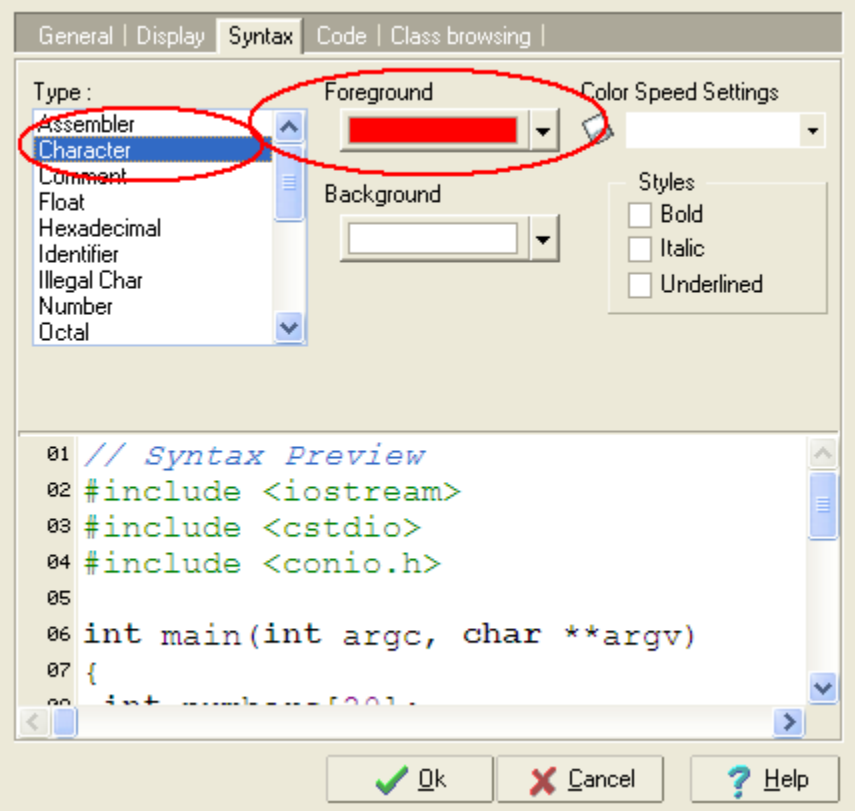
آن‌چه در این بخش گفته شد جهت آشنایی مختصر و سریع با محیط Dev C++ بود و برای مطالعه‌ی بیشتر می‌توانید به منابع دیگر رجوع کنید.

وب

در آدرس‌های زیر، نکاتی پیرامون Dev C++ آمده است:

http://www.bloodshed.net/dev/doc/index.html

http://sourceforge.net/projects/dev-cpp/forums/forum/128327



شکل ۱-۱۴: برخی تنظیمات ویرایشگر متن Dev C++ و نحوه تغییر رنگ متن نمایش داده

شاید بهتر باشد در آغاز فصل ۳ که برنامه‌نویسی به صورت جدی آغاز می‌شود، این بخش را برای آمادگی هر چه بیشتر در برنامه‌نویسی با Dev C++، مجدداً مرور نمایید.

فصل دوم

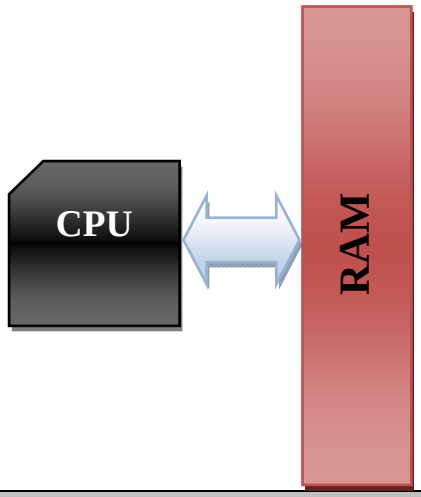
بررسی مسأله و مشخص کردن ورودی و خروجی‌ها

۱. مدل پایه‌ی کامپیوتر و مفهوم متغیر

در فصل اول راجع به اهمیت تجزیه‌ی صحیح مسأله و مشخص کردن ورودی‌ها و خروجی‌ها بحث شد. در این مورد مثالی نیز از حل معادله‌ی درجه اول زده شد و نمودار گردشی آن نیز ترسیم شد. در این فصل قرار است دقیق‌تر به این مسأله پرداخته شود و با مثال‌های متعدد، نحوه‌ی حل مسأله و بیان الگوریتمِ تولیدِ خروجی‌هایِ مورد نیاز تشریح شود، زیرا اگر تسلط به دستور زبان یک زبان برنامه‌نویسی مثل C بالا باشد ولی قدرت حل مسأله پایین باشد، فرد نمی‌تواند برنامه‌ی مورد نظر خود را بنویسد. زیرا اگرچه نحوه‌ی برنامه‌نویسی را می‌داند، اما چگونه حل کردن خود مسأله را نمی‌داند و این ضعف بزرگی است. در این فصل، هدف آن است که با تمرین، بتوانید پس از دیدن صورت مسأله، راه حل و الگوریتم آن را بیان کنید.

برای رسیدن به این هدف، لازم است کمی در مورد ساختار کامپیوتر بحث شود. کامپیوتر یک تعریف عوامانه و ظاهری دارد که معمولاً اغلب مردم با آن آشنا هستند: یک صفحه‌ی نمایش، صفحه کلید، موس و محفظه‌ی اصلی یا Case کامپیوتر. اما این تعریف، تعریف علمی کامپیوتر نیست، تعریف علمی کامپیوتر در ساده‌ترین حالت عبارت است از یک پردازنده که به یک حافظه متصل است. پردازنده می‌تواند از روی حافظه اعدادی را خوانده یا بر روی آن اعدادی را بنویسد، همچنین پردازنده می‌تواند عملیات ریاضی نظیر جمع و ضرب و یا اعمال مقایسه‌ای نظیر بزرگ‌تر، کوچک‌تر یا مساوی بودن را انجام دهد (شکل۲-۱). حافظه‌ی کامپیوتر را می‌توانیم در تئوری نامحدود فرض کنیم، به این ترتیب می‌توانیم هر چقدر که خواستیم در خانه‌های حافظه عدد ذخیره کرده و بعداً از آن بخوانیم یا بر روی آن عملیات خاصی انجام دهیم. معمولاً خانه‌های حافظه را نام‌گذاری کرده و به آن‌ها متغیر می‌گوییم، زیرا مقدار آن‌ها قابل تغییر است. نام‌گذاری آن‌ها هم همان‌طور که در فصل قبل دیدیم به سادگی صورت

می‌گیرد، مثلاً می‌توانیم سه متغیر با نام‌های a,b,c داشته باشیم یا سه متغیر با نام‌های x,y,z یا هر نام دیگری.



شکل ۲-۱: طرحواره‌ی یک کامپیوتر ساده که از یک پردازنده و یک حافظه تشکیل شده است

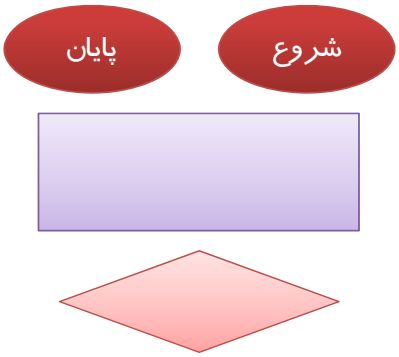
لزومی ندارد اسامی متغیرها یک حرفی باشد، می‌توانیم متغیرهایی با نام *input* یا *index* نیز داشته باشیم، حتی می‌توان اسم متغیرها را فارسی انتخاب کرد، اما چون در ریاضیات و نیز زبان‌های برنامه‌نویسی، چنین چیزی مرسوم نیست، ما نیز از انجام چنین کاری پرهیز می‌کنیم.

۲. نمودار گردشی

اکنون پس از مرور این مقدمات، می‌توانیم یک بار دیگر عمیق‌تر به مسأله‌ی مطرح‌شده در فصل قبل بپردازیم: مسأله‌ی معادله‌ی درجه اول. این مسأله دو ورودی دارد که باید کاربر مشخص کند، ورودی‌هایی که از این به بعد به آن‌ها متغیر می‌گوییم. پس مسأله دو متغیر

ورودی دارد و نتیجه باید در یک متغیر خروجی ذخیره شود. متغیرهای ورودی را a,b و متغیر خروجی را x می‌نامیم. به این ترتیب الگوریتم یا همان روش حل مسأله این طور خواهد بود که ابتدا متغیرهای a,b از کاربر گرفته شود[1]، سپس اگر $a \neq 0$ بود، آنگاه نتیجه‌ی نهایی که همان $-\frac{b}{a}$ است در x ذخیره شود. x نیز متغیر دیگری در حافظه است که می‌توان آن را نمایش داد. البته معمولاً در طراحی الگوریتم یا ترسیم نمودار گردشی، نمایش مد نظر نیست و همین قدر که نتیجه‌ی نهایی محاسبه شود کافیست، زیرا نمایش متغیرها از حافظه یک روال روشن و سر راست است که پیچیدگی خاصی ندارد و بنابراین خیلی مورد بحث نیست. برای ترسیم نمودار گردشی، قواعد خاصی وجود دارد. هر نمودار گردشی معمولاً بخش‌های شروع و پایان دارد که در یک کادر دایروی یا بیضی قرار می‌گیرند. انجام عملیات‌های مختلف، نظیر ضرب و جمع و... در کادرهای مستطیلی قرار می‌گیرند و شرط‌ها نیز داخل کادرهای لوزی شکل. همچنین معمولاً برای گرفتن ورودی و یا تولید خروجی، از کادرهای خاصی نظیر متوازی‌الاضلاع و یا اشکال خاص دیگر استفاده می‌شود، اما ما در این کتاب برای سادگی، عملیات ورودی گرفتن و یا تولید خروجی را نیز درون کادرهای مستطیلی قرار می‌دهیم. انواع این کادرها در شکل ۲-۲ نمایش داده شده است. پس از این توضیحات، باید مفهوم نمودار گردشی ترسیم شده در شکل ۱-۴ فصل قبل واضح‌تر شده باشد. اکنون می‌پردازیم به حل چند مثال دیگر با استفاده از نمودار گردشی تا هم مفهوم طراحی الگوریتم برای حل مسأله توسط کامپیوتر روشن شود و هم بازنمایی الگوریتم طراحی شده با استفاده از نمودار گردشی.

[1] اینکه این متغیرها چطور از کاربر گرفته شود، فعلاً موضوع بحث ما نیست و در فصل بعد به آن پرداخته خواهد شد. فعلاً هدف مشخص کردن روال و ترتیب کارهایی است که باید انجام بگیرد تا مسأله توسط کامپیوتر کامپیوتر حل شود.



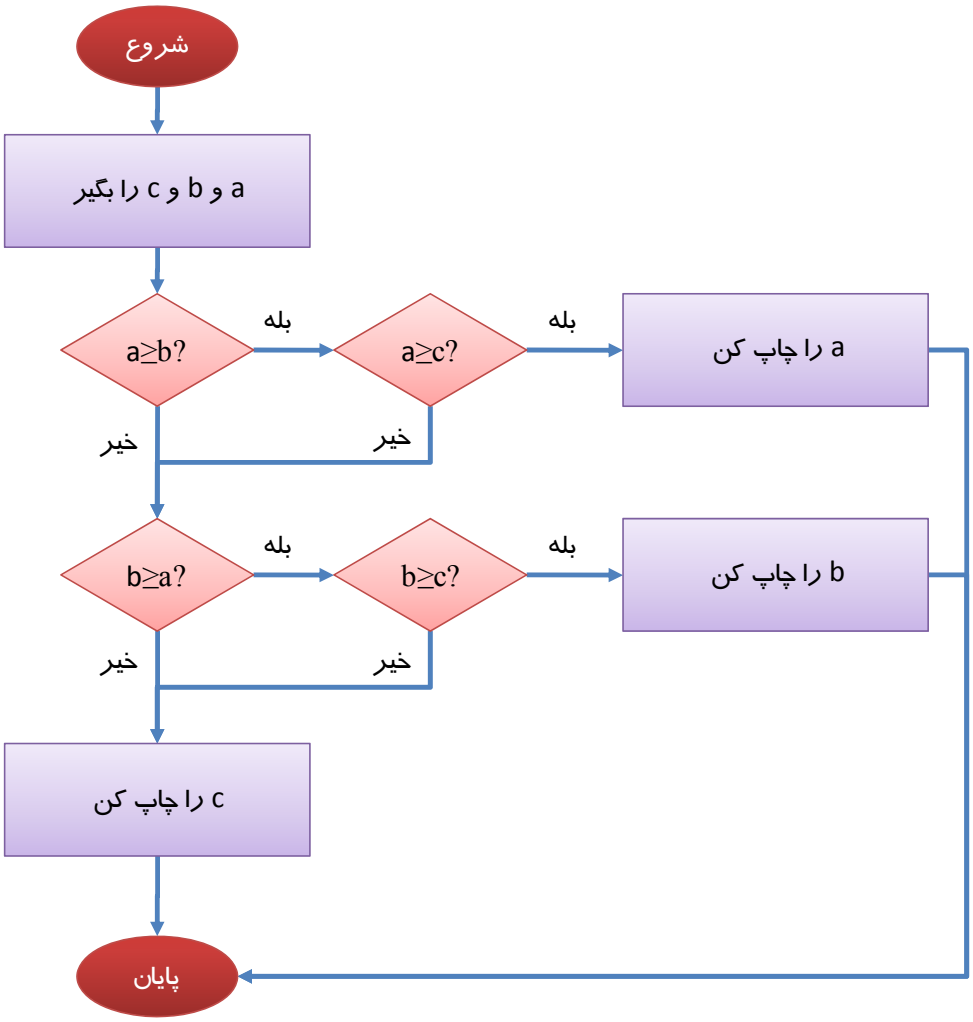
شکل ۲-۲: نمایش انواع کادرهای نمودار گردشی

۳. حل مسأله‌ی یافتن بیشینه بین سه عدد

فرض کنید می‌خواهیم از بین سه عدد، حداکثر یا بیشینه[۲]ی آن‌ها را انتخاب کنیم. این سه عدد را که در حافظه‌ی کامپیوتر ذخیره خواهند شد، a و b و c می‌نامیم. برای این که این اعداد معنا داشته باشند، باید از ورودی گرفته شوند، زیرا تا ما ورودی از کاربرِ برنامه نگیریم، نمی‌توانیم خروجی تولید کنیم و تولید خروجی این مثال، یعنی بیشینه‌ی سه عدد نیز وابسته به ورودی‌های برنامه است. به عنوان مثال اگر کاربر سه عدد a و b و c را به ترتیب ۷ و ۲۰ و ۱۳ وارد کند، خروجی ۲۰ خواهد بود و چنانچه کاربر سه عدد را ۵- و ۲- و ۱- وارد کند، خروجی ۱- خواهد بود. پس خروجی تولید شده توسط کامپیوتر کاملاً وابسته به ورودی‌هاست. اما تولید خروجی، با عملیات ضرب و جمع و... ارتباطی ندارد و تنها باید از مقایسه استفاده کنیم. در شکل ۲-۳ نمودار گردشی مربوط به حل این مسأله را مشاهده می‌کنید. با کمی تأمل در نمودار گردشی شکل ۲-۳ در می‌یابید که الگوریتم ارائه شده درست است. برای اطمینان بیشتر می‌توانید به جای a و b و c اعداد مختلفی گذاشته و در پایان ببینید که خروجی به درستی تولید می‌شود. در این نمودار، برای اعلام کردن بیشینه، از دستور چاپ استفاده شده که معمولاً منظور از

[2] Maximum

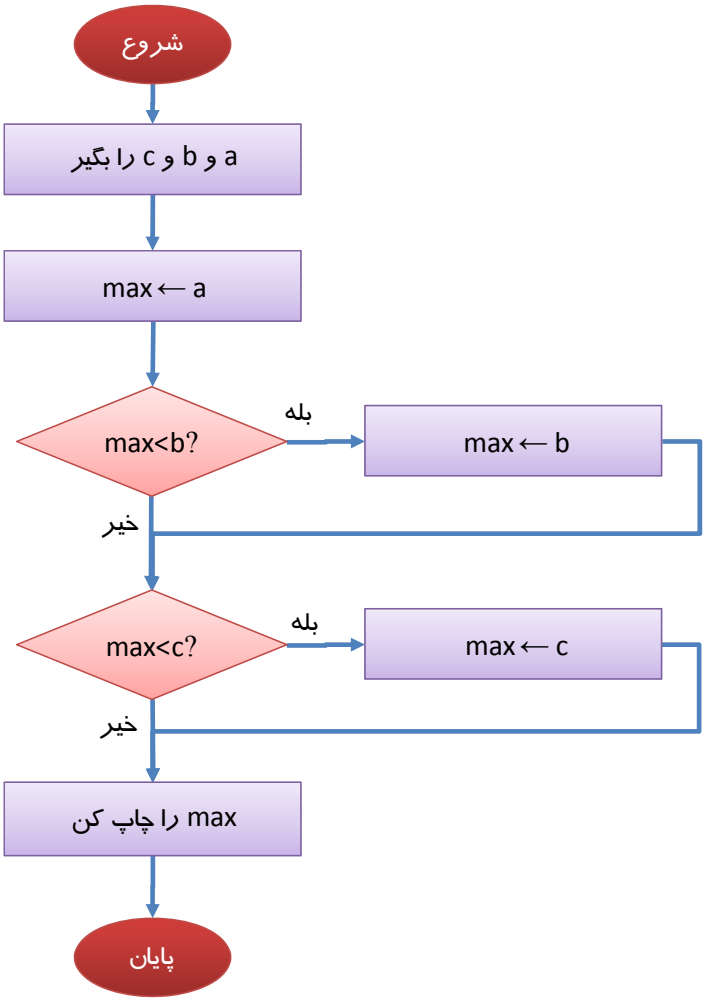
چاپ، نمایش دادن مقدار آن متغیر بر روی صفحه‌ی نمایش است. می‌توان غیر از چاپ، کار دیگری کرد و آن این که مقدار بیشینه را در متغیر چهارمی ذخیره کنیم، درست مثل مسأله‌ی معادله‌ی درجه اول که جواب آن را در متغیر x ذخیره کردیم.



شکل ۲-۳: نمودار گردشی حل مسأله‌ی یافتن بیشینه از بین سه عدد بدون استفاده از متغیر چهارم

به این ترتیب نام آن متغیر را max می‌گذاریم و در ابتدا مقدار متغیر a را در آن قرار می‌دهیم (معمولاً برای نمایش انتقال مقدار یک متغیر به متغیر دیگر از علامت ← استفاده می‌شود و این فلش همواره از سمت راست به چپ است و در سمت چپ آن متغیری قرار

دارد که قرار است مقدار متغیر سمت راست در آن قرار بگیرد)، سپس مقدار b را با مقدار max مقایسه می‌کنیم، اگر b از max بزرگ‌تر بود، مقدار b را در max قرار می‌دهیم و اگر نه ادامه می‌دهیم، سپس مقدار c را با max مقایسه می‌کنیم و اگر c از max بزرگ‌تر بود، مقدار c را در max قرار می‌دهیم، به این ترتیب در پایان اجرای این الگوریتم، max حاوی مقدار بیشینه‌ی a و b و c است که می‌توانیم آن را چاپ کرده و یا جهت کاری دیگر از آن استفاده نماییم. نمودار گردشی این الگوریتم در شکل ۲-۴ نمایش داده شده است.



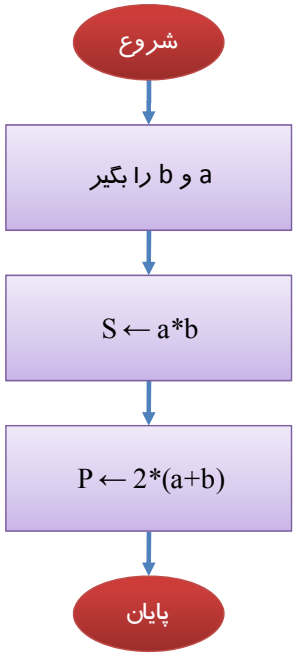
شکل ۲-۴: نمودار گردشی حل مسأله‌ی یافتن بیشینه از بین سه عدد با استفاده از متغیر چهارم

پس از حل این مثال، باید کاملاً شیوه‌ی برخورد با مسایل کامپیوتری را درک کرده و تفاوت‌هایی را که با مسایل عادی روزمره و یا مسایل ریاضی دارند، فهمیده باشید.

۴. حل چند مسأله و بازنمایی الگوریتم با نمودار گردشی

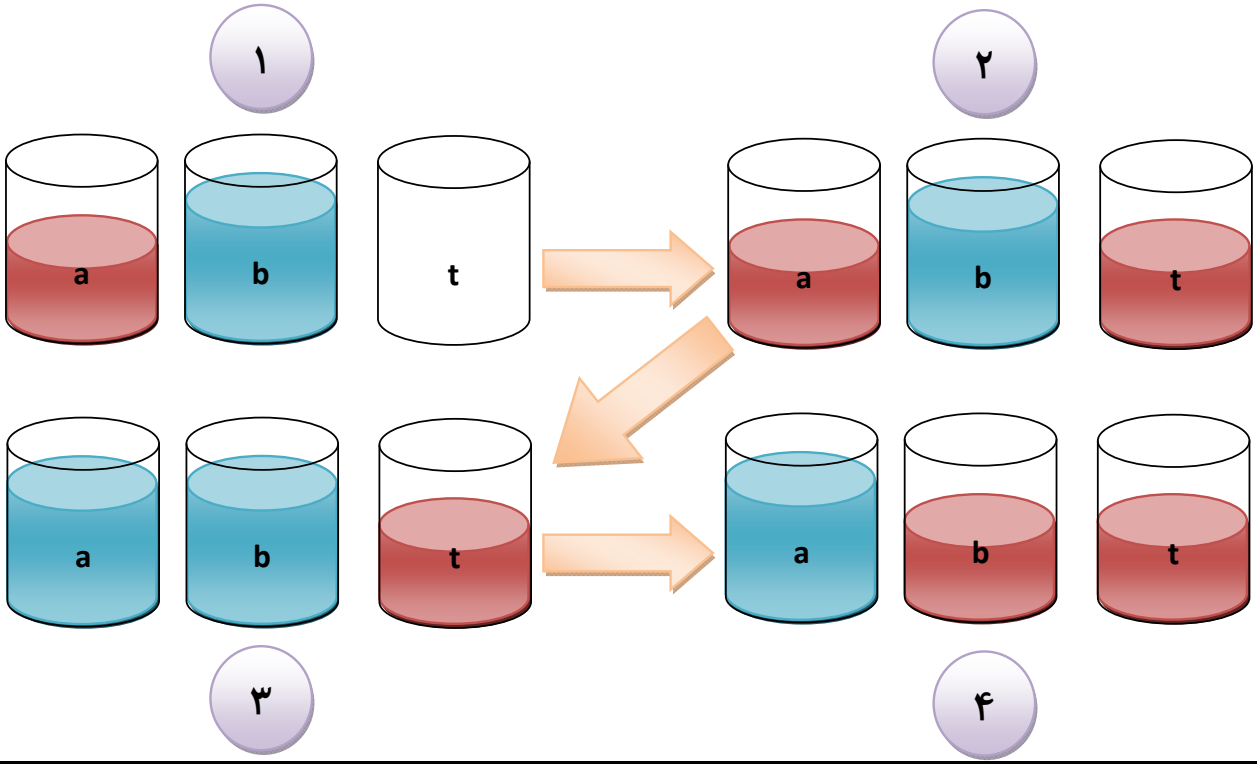
اکنون می‌پردازیم به حل یک مثال دیگر، فرض کنید می‌خواهیم با گرفتن طول و عرض یک مستطیل، محیط و مساحت آن را محاسبه کنیم. ورودی‌ها، طول و عرض مستطیل هستند که آن‌ها را a و b نام‌گذاری می‌کنیم و خروجی‌ها نیز محیط و مساحت هستند که آن‌ها را به ترتیب P و S می‌نامیم. مطابق آن چه که تا کنون گفته شده، نمودار گردشی الگوریتم حل این مسأله به سادگی به دست می‌آید. که در شکل ۲-۵ مشاهده می‌شود.

همان‌طور که در شکل ۲-۵ مشاهده می‌شود، برای قرار دادن نتیجه‌ی محاسبه نیز از علامت ← استفاده می‌شود. همچنین در نوشتار کامپیوتری مرسوم است که همواره عملگر * به نشانه‌ی ضرب بین دو متغیری که قرار است ضرب شوند قرار می‌گیرد و برای نمایش a ضرب در b، آن را به صورت a*b می‌نویسند، نه ab، زیرا ممکن است با متغیری به نام ab اشتباه شود. حال که به اینجا رسیدیم، می‌خواهیم یک مسأله‌ی جالب و مهم را با هم حل کنیم.



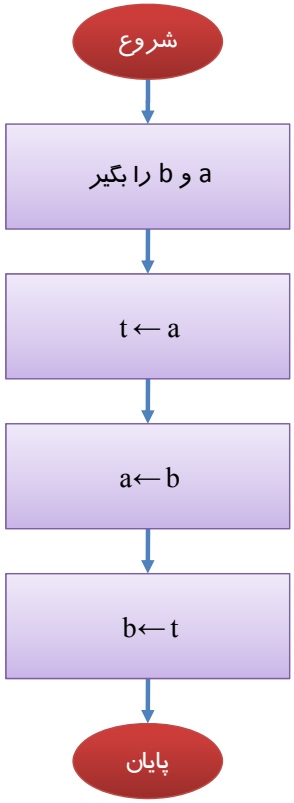
شکل ۲-۵: نمودار گردشی حل مسأله‌ی محاسبه‌ی محیط و مساحت یک مستطیل

فرض کنید دو متغیر به نام‌های a و b داریم و می‌خواهیم محتوای آن‌ها را با هم جابه‌جا کنیم، یعنی مثلاً چنانچه a برابر ۱۷ و b برابر ۶ است، پس از اجرای الگوریتم، a برابر ۶ و b برابر ۱۷ شود. برای انجام این کار احتیاج به یک متغیر سومی هست که به عنوان متغیر کمکی ایفای نقش کند، زیرا اگر مقدار a را در b یا b را در a قرار دهیم، بلافاصله مقدار قبلی آن‌ها از بین رفته و دیگر در دسترس نیست. بنابراین از متغیر سومی استفاده می‌کنیم که موقتاً مقدار یکی از متغیرها را برای ما ذخیره کند. این عمل درست شبیه تعویض محتوای دو ظرف است که برای انجام آن احتیاج به یک ظرف سوم موقتی برای تعویض بدون تداخل محتوای دو ظرف است (شکل ۲-۶).



شکل ۲-۶: نمایش مفهومی تعویض مقادیر دو متغیر با یکدیگر (به ترتیب مراحل دقت کنید)

به این ترتیب و با توجه به توضیحات داده شده، نمودار گردشی تعویض محتوای دو متغیر با یکدیگر در شکل ۲-۷ آمده است.



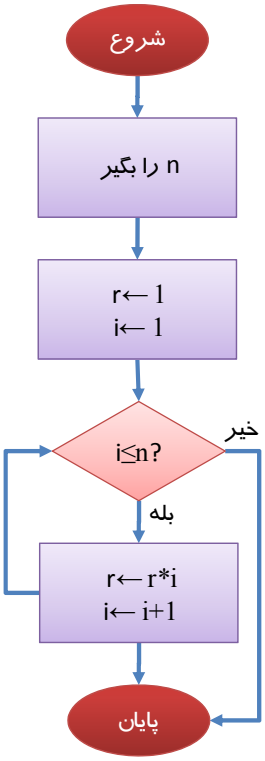
شکل ۲-۷: نمودار گردشی تعویض محتوای دو متغیر با یکدیگر

به این ترتیب اگر ابتدا مقدار a برابر ۱۷ و b برابر ۶ باشد، در قدم اول t برابر مقدار a یعنی ۱۷ می‌شود، سپس a برابر مقدار b یعنی ۶ شده و در انتها b برابر مقدار t یعنی ۱۷ می‌شود و مقدار t همان مقدار اولیه‌ی a است.

مسأله‌ی اخیر که حل شد، مسأله‌ی مهمی است، از این بابت که شهود خاصی نسبت به حافظه‌ی کامپیوتر و متغیرهای آن و نحوه‌ی ذخیره‌سازی آن‌ها به دست می‌دهد. حال می‌پردازیم به حل یک مسأله‌ی دیگر که اندکی بار محاسباتی بیشتری داشته باشد: مسأله‌ی محاسبه‌ی n! با گرفتن n از ورودی. برای حل چنین مسأله‌ای، چه باید بکنیم؟

ابتدا باید بدانیم که $n! = 1\times2\times3\times\ldots\times n$، پس محاسبه‌ی n! به n-1 عمل ضرب احتیاج دارد، پس تعداد عملیات آن ثابت نیست و وابسته به ورودی کاربر است. بنابراین باید در جایی (یعنی در یک متغیر) تعداد عملیات‌ها را ذخیره کنیم تا تعداد عملیات‌ها را در کنترل داشته باشیم و هرگاه تعداد عملیات‌ها کافی شد، الگوریتم خاتمه پیدا کند. به این ترتیب نمودار

گردشی حل چنین مسأله‌ای مانند شکل ۲-۸ خواهد بود. جالب است بدانید به متغیرهایی که به این ترتیب مسئولیت شمارش تعداد عملیات را بر عهده می‌گیرند، شمارنده[۳]، می‌گویند.



شکل ۲-۸: نمودار گردشی حل مسأله‌ی محاسبه‌ی n!

ممکن است از خود مقدار شمارنده استفاده بشود (نظیر همین الگوریتم که از مقدار i استفاده شده است) و یا از مقدار آن استفاده نشود و متغیر شمارنده، صرفاً مسئولیت نگهداری تعداد عملیات‌های انجام شده را داشته باشد. در نمودار گردشی شکل ۲-۸، r متغیری است که قرار است نتیجه‌ی عملیات‌های انجام شده را نگه دارد. در نمودار گردشی شکل ۲-۸، r متغیری است که قرار است نتیجه‌ی نهایی، یعنی n! را در خودش نگه دارد، بنابراین ابتدا مقدار آن را برابر ۱ قرار می‌دهیم، سپس هر بار که یکی به i اضافه می‌شود، اگر i≤n باشد، r فعلی، در i ضرب می‌شود. به این ترتیب، مثلاً اگر n برابر ۵ باشد، مقادیر r با توجه به i این‌طور عوض می‌شود:

r(فعلی)	i	r(قبلی)
۱	۱	۱
۲	۲	۱

[3] Counter

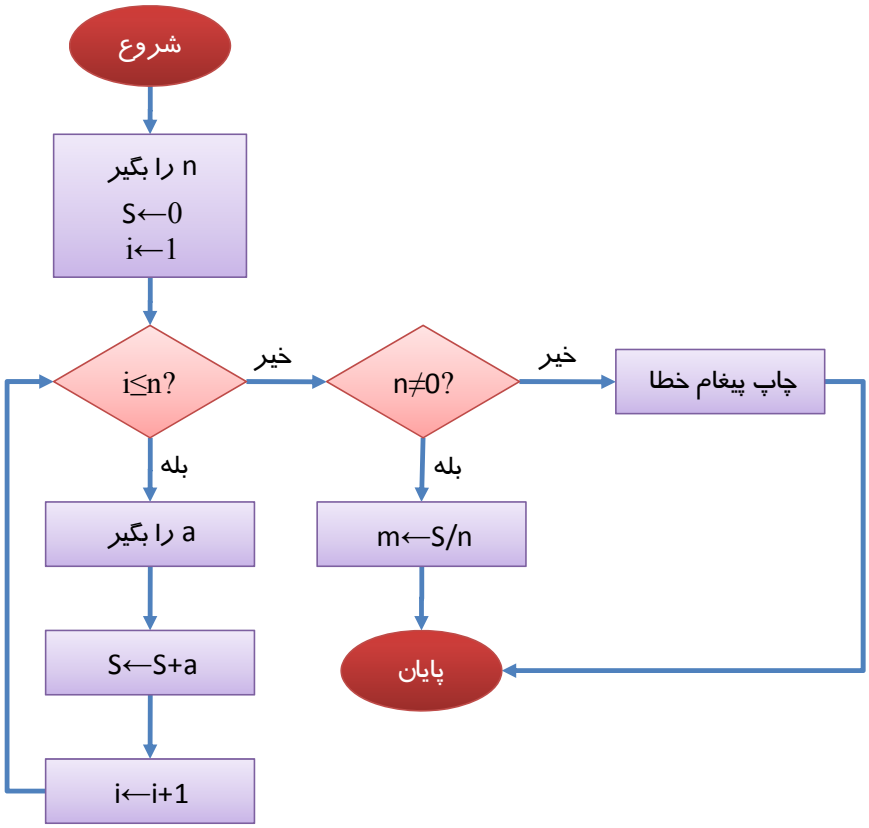
۶	۳	۲
۲۴	۴	۶
۱۲۰	۵	۲۴

منظور از r(قبلی) و r(فعلی) نیز، r قرار گرفته در سمت راست و چپ ← است: زمانی که عبارتی را بر حسب مقدار قبلی یک متغیر محاسبه می‌کنیم و مجدداً حاصل آن را در آن متغیر قرار می‌دهیم، **محاسبه‌ی اولیه‌ی سمت راست با مقدار قبلی انجام می‌شود** و پس از آن که محاسبه انجام شد، **مقدار فعلی در متغیر قرار می‌گیرد**، مثل i←i+۱ یا r←r×i که در هر مورد، هر مقداری که در متغیر بوده در نظر گرفته می‌شود، محاسبات سمت راست انجام می‌شود و در نهایت مقدار نهاییِ محاسبه، هر چه که باشد، درون متغیر سمت چپ قرار می‌گیرد.

پس از متغیرهای شمارنده، می‌خواهیم با یک مفهوم دیگر آشنا شویم: فرض کنید می‌خواهیم n متغیر را از ورودی خوانده و میانگین آن‌ها را محاسبه کنیم، آیا برای این کار احتیاج به n متغیر داریم تا مقادیر را تک تک در آن‌ها ذخیره کنیم؟ چنانچه n را کاربر وارد کند، یعنی ما از قبل ندانیم که n چند است چطور؟ برای پاسخ به این سؤال‌ها، ابتدا به نمودار گردشی الگوریتم حل این مسأله توجه نمایید. همان‌طور که در نمودار گردشی شکل ۲-۹ دیده می‌شود، در حل این مسأله به هیچ وجه از n متغیر استفاده نشده، بلکه از یک متغیر a برای خواندن ورودی استفاده شده و سپس، هر بار که کاربر عدد جدیدی را در متغیر a وارد کرده، آن را با مقدار قبلی S که حاصل جمع کل اعداد را نگه می‌دارد، جمع کرده‌ایم. به این عمل، یعنی جمع زدن تعدادی متغیر در یک متغیر (مثل S) جمع انباره‌ای[۴] و به خود متغیر (S) متغیر بانک می‌گوییم. دقت

[4] Accumulated sum

دارید، در ابتدای کار S را صفر کرده‌ایم تا مقدار اولیه‌ای برای S متصور نباشد تا سایر مقادیر با آن جمع شوند و در واقع با این کار، خیالمان راحت خواهد بود که در انتها، S تنها حاوی مجموع متغیرهای وارد شده در a توسط کاربر است نه مقداری اضافه‌تر یا کمتر. به این کار، یعنی دادن مقدار اولیه‌ی صحیح به متغیرها (مثل صفر کردن S یا ۱ کردن i) مقداردهی اولیه[5] می‌گویند که در درست کار کردن برنامه‌ها نقش به سزایی داشته و بسیار حایز اهمیت است.

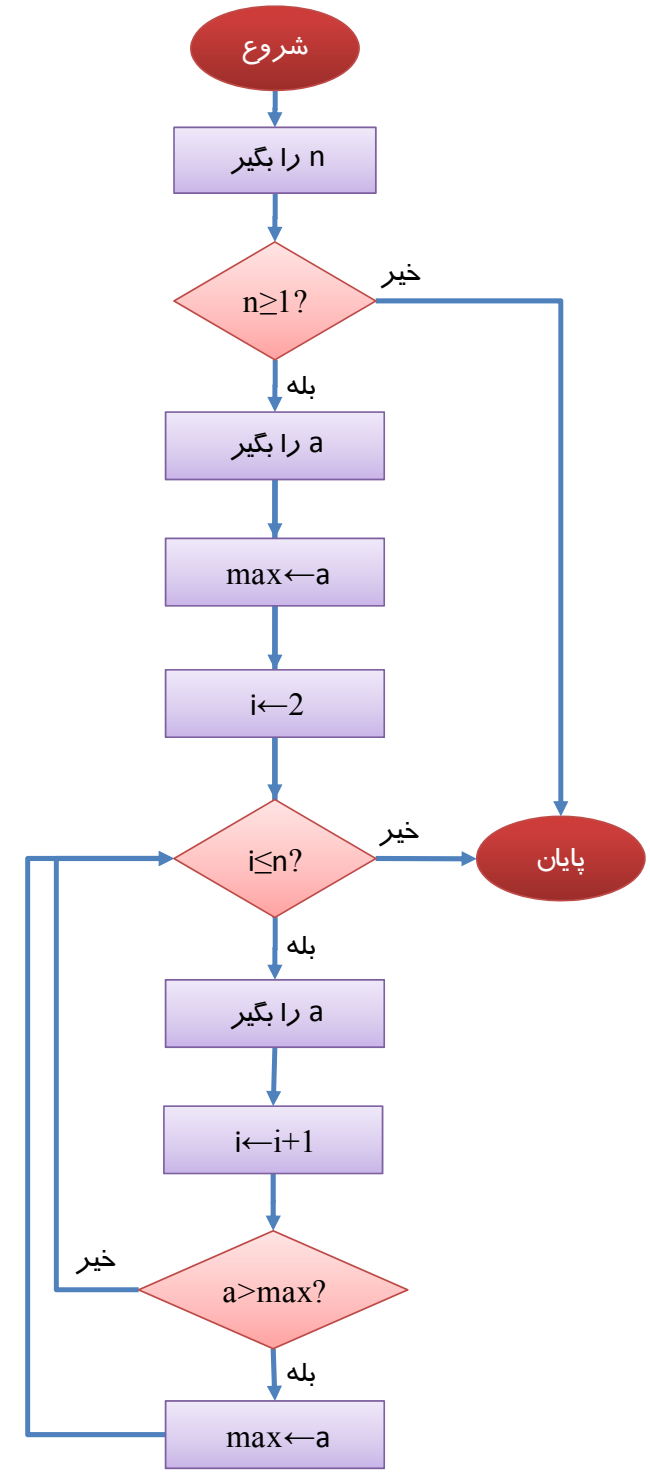


شکل ۲-۹: نمودار گردشی حل مسأله‌ی محاسبه‌ی میانگین n عدد

مثال بعدی که قصد مطرح کردن آن را داریم کمی شبیه مسأله‌ی قبلی است. فرض کنید کاربر قرار است n عدد (که n را هم خود کاربر مشخص می‌کند) وارد کند و ما از بین آن‌ها عدد بیشینه را به دست آورده و در متغیر دیگری به نام max قرار دهیم.

[5] Initialization

ممکن است با دیدن نمودار گردشی شکل۲-۱۰ کمی گیج شده باشید، پس اندکی آن را با هم مرور می‌کنیم: ابتدا عدد n از کاربر گرفته می‌شود تا مشخص شود کاربر چند عدد می‌خواهد وارد کند؟ سپس چک می‌کنیم که آیا $n\geq 1$ یا نه؟ هر چند بدیهی به نظر می‌رسد که کاربر عددی بزرگ‌تر یا مساوی یک وارد کند، اما همواره در نظر داشته باشید که ممکن است ورودی‌هایی، غیر از آنچه ما فرض کرده‌ایم باید وارد شود، توسط کاربر وارد شود! پس همواره باید آمادگی لازم را داشته باشیم و الگوریتم ارائه شده توسط ما جامعیت داشته باشد، یعنی شامل تمامی حالات ممکن بشود. پس از آنکه مطمئن شدیم $n\geq 1$ است، اولین عدد را می‌خوانیم و آن را در max قرار می‌دهیم. سپس i را برابر ۲ قرار می‌دهیم، دلیل آن این است که ما در حال حاضر یک عدد را از ورودی گرفته‌ایم، پس مقدار اولیه‌ی i باید برابر ۲ باشد و بلافاصله چک شود که آیا این مقدار، از n کوچک‌تر یا مساوی هست یا خیر؟



شکل ۲-۱۰: نمودار گردشی حل مسأله‌ی یافتن بیشینه از بین n عدد

اگر نبود (یعنی n=۱ بوده است) برنامه پایان یافته و max همان اولین متغیری است که خوانده شده، در غیر این صورت، هر بار یک متغیر جدید از کاربر گرفته و در a ذخیره می‌کنیم، سپس شمارنده را یکی زیاد می‌کنیم که نشان دهنده‌ی خواندن یک عدد جدید است، سپس چک می‌کنیم که آیا این متغیر جدید از max بزرگ‌تر است یا خیر؟ چنانچه این متغیر از max بزرگ‌تر بود فرض قبلی نقض می‌شود و عددی بزرگ‌تر از max پیدا می‌شود، که آن را به عنوان مقدار بیشینه‌ی جدید در max قرار می‌دهیم و برمی‌گردیم به چک کردن شرط شمارنده که بیش از n بار از کاربر عدد نگیریم. در پایان اجرای الگوریتم، عددی که در max قرار دارد، مقدار بیشینه‌ی اعدادی است که کاربر وارد کرده است.

برای درک بهتر کارکرد الگوریتم فوق، فرض می‌کنیم n=۷ بوده و اعداد وارد شده توسط کاربر به ترتیب برابر ۶،۱۳،۲۰،۱۹،۵،۴۰،۲۵ باشند. به این ترتیب مقدار max مطابق زیر عوض می‌شود:

max(فعلی)	a	max(قبلی)
۶	۶	-
۱۳	۱۳	۶
۲۰	۲۰	۱۳
۲۰	۱۹	۲۰
۲۰	۵	۲۰
۴۰	۴۰	۲۰
۴۰	۲۵	۴۰

به این ترتیب پس از اجرای الگوریتم، مقدار max برابر ۴۰ خواهد بود که بیشینه‌ی مقدارهایی است که توسط کاربر وارد شده. دقت دارید که در سطر اول مقدار قبلی max در دسترس نیست و البته مهم هم نیست، زیرا مقدار آن هرچه که باشد، برابر اولین عددی که کاربر وارد می‌کند خواهد شد.

۵. بازنمایی متنی حل مسأله

اکنون پس از دیدن مثال‌های متعدد باید با روش طراحی الگوریتم، برای حل یک مسأله توسط کامپیوتر و بازنمایی آن به روش نمودار گردشی آشنا شده باشید. این روش حل مسأله، به صورت قدم به قدم حل مسأله را ترسیم می‌کند و البته تنها مختص علم برنامه نویسی نیست، بلکه در بسیاری زمینه‌های دیگر، نظیر علوم مدیریتی استفاده می‌شود. البته بازنمایی مسأله با نمودار گردشی مشکلات خاصی نیز دارد، مثلاً چنانچه گام‌های مورد نیاز برنامه زیاد باشد و مسأله اندکی مفصل باشد، نمایش الگوریتم با نمودار گردشی کمی دشوار می‌شود و ممکن است دنبال کردن آن آسان نباشد و یا اصلاً نمودار در یک صفحه جا نشود، در چنین شرایطی بهتر است از بازنمایی متنی الگوریتم بهره برد. بازنمایی متنی شبیه بازنمایی به روش نمودار گردشی است، با این تفاوت که به جای نمایش مراحل مختلف الگوریتم به صورت گرافیکی، کارها به ترتیب و با شماره‌ی مرحله مشخص می‌شوند. به عنوان مثال بازنمایی متنی مسأله‌ی آخر را در شکل ۲-۱۱ مشاهده می‌کنید. همان‌طور که مشاهده می‌شود، این روش با شماره‌گذاری مراحل مختلف، از پیچیدگی‌های گرافیکی پرهیز می‌کند و متن الگوریتم در این روش می‌تواند طولانی‌تر باشد و مشکلات مربوط به نمودار گردشی پیش نخواهد آمد. پس از اجرای هر قدم از اجرای الگوریتم به صورت پیش فرض به مرحله‌ی بعدی می‌رویم مگر آنکه شرطی ما را محدود کند

تا به مرحله‌ای دیگر برویم. با این حال و با تمام این توضیحات، بازنمایی الگوریتم به روش نمودار گردشی در بسیاری از موارد روش مفید و کارآمدی است و دنبال کردن آن به خاطر گرافیکی بودن آن آسان‌تر است. مطالبی که در این فصل آمده، در فصول آینده نیز مورد استفاده قرار می‌گیرد، خصوصاً از مفهوم نمودار گردشی در بیان الگوریتم مسایل مختلف استفاده خواهد شد.

(۱) n را بگیر

(۲) اگر $n \geq 1$ است به (۳) برو در غیر این صورت برو به (۱۰)

(۳) a را بگیر

(۴) مقدار max را برابر a قرار بده

(۵) i را برابر ۲ قرار بده

(۶) اگر $i \leq n$ برو به (۷)، در غیر این صورت برو به (۱۰)

(۷) a را بگیر

(۸) i را یکی زیاد کن

(۹) اگر $a > max$ است مقدار max را برابر a قرار بده و برو به (۶) در غیر این صورت بدون انجام کاری برو به (۶)

(۱۰) پایان

شکل ۲-۱۱: بازنمایی متنی حل مسأله‌ی یافتن بیشینه از بین n عدد

۳

فصل سوم
همه با هم

ورودی-خروجی، متغیرها و محاسبات

۱. آشنایی با تولید ورودی و خروجی و انواع متغیرها در برنامه‌نویسی

تا کنون در فصل‌های قبل با مفاهیم پایه‌ای برنامه‌نویسی و حل مسایل توسط کامپیوتر آشنا شده‌اید، در این فصل قرار است به صورت عملی برنامه‌نویسی در محیط Dev C++ را آغاز کنیم. همان‌طور که قبلاً هم گفته شد، یکی از اصلی‌ترین بخش‌های حل یک مسأله گرفتن ورودی‌ها از کاربر و تولید خروجی مناسب است. در این بخش ابتدا با نحوه‌ی تولید خروجی آشنا خواهیم شد. می‌خواهیم برنامه‌ای بنویسیم که عبارت Hello World! را بر روی صفحه چاپ کند. برای نوشتن برنامه کافی است تا در Dev C++ یک فایل جدید باز و سپس برنامه‌ی مورد نظر را در آن تایپ کنیم. برنامه‌ای که عمل خواسته شده را انجام می‌دهد در کد ۳-۱ آمده است:

کد ۳-۱

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     cout<<"Hello World!";
7     getch();
8     return 0;
9  }
```

لوح

برنامه‌ی مربوط به کد ۳-۱ بر روی لوح فشرده و در آدرس زیر قرار دارد:

CDROM:\Codes\Chapter 03\P03_01\P03_01.cpp

حال که اولین برنامه را در C نوشتیم، به توضیحاتی راجع به آن می‌پردازیم. در اغلب برنامه‌های C، یک قالب تکرار شونده وجود دارد که ثابت است. برخی از بخش‌های این قالب را باید فعلاً به خاطر سپرد، زیرا توضیح آن و این که دقیقاً هر یک چه کار می‌کنند نیازمند مقدماتی است که برخی از آن‌ها تا چندین فصل جلوتر مطرح نخواهند شد. البته سعی می‌شود تا حدودی راجع به آن صحبت شود تا شهوداً مشخص باشد هر خط از برنامه چه کاری انجام می‌دهد. این قالب ثابت، در شکل ۳-۱ مشاهده می‌شود.

```
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{
    متن برنامه
    getch();
    return 0;
}
```

اضافه کردن لیست دستورات به برنامه

فعال کردن دستورات ورودی-خروجی

معرفی بدنه‌ی اصلی

شروع بلوک اصلی برنامه

انتظار برای فشردن یک کلید

دستور خاتمه برنامه و بازگشت

انتهای بلوک اصلی برنامه

شکل ۳-۱: معرفی قالب کلی برنامه‌های C نوشته شده در Dev C++

البته لازم به ذکر است این قالب مربوط به فرم کلی اکثر برنامه‌هایی است که در C نوشته می‌شود. و ممکن است برنامه‌ای وجود داشته باشد که از این قالب کلی تبعیت نکند، اما ملاک در معرفی این قالب، اکثر برنامه‌هایی است که از این به بعد با هم خواهیم دید. بخش‌های #include مربوط به شامل شدن دستوراتی است که در فایل‌های مختلف وجود دارد، به عنوان مثال با نوشتن #include<iostream> به برنامه امکان استفاده از دستورات ورودی-خروجی را می‌دهیم. به عنوان مثال دستور cout که در کد ۳-۱ آمده است، در این فایل قرار دارد و یا دستور ;getch() در فایل conio.h قرار دارد. معمولاً کلیه‌ی این فایل‌ها (به جز iostream) پسوند h دارند که حرف اول header به معنای سر آیند است، یعنی بخشی از برنامه که در ابتدای آن می‌آید.

پس این بخش اولیه، یک جمله قرار دارد: ;using namespace std، این جمله فعال کننده‌ی دستورات iostream است، یعنی باید حتماً این خط وجود داشته باشد، تا بتوانیم از دستوراتی نظیر cout استفاده کنیم. پس از این خطوط که به نوعی آماده‌سازی برای برنامه‌نویسی هستند، به بدنه‌ی اصلی برنامه می‌رسیم. بدنه‌ی اصلی برنامه با int main() شروع می‌شود و بین دو علامت {} قرار می‌گیرد، یعنی کلیه‌ی دستوراتی که به مجموعه‌ی آن‌ها برنامه می‌گوییم، یا همان الگوریتم حل مسأله‌ی ما، داخل {} خواهد بود. در انتهای بخشی که به صورت کلی متن برنامه نام گذاری شده، دو خط دیگر دیده می‌شود. خط آخر که return 0 است ثابت است و خط قبل از آن یعنی ;getch()، در اغلب برنامه‌ها حضور دارد، زیرا این دستور به معنای آن است که منتظر زدن کلیدی از طرف کاربر باشیم و یا به بیان دیگر صبر کنیم تا کاربر کلیدی را بزند. مفهوم این عبارت این است که صبر کنیم تا کاربر خروجی تولید شده را ببیند، به طور مثال اگر این خط را از کد ۳-۱ حذف کنید، پس از اجرای برنامه (توسط کلید F9) یک صفحه‌ی مشکی باز شده و بلافاصله بسته می‌شود، در واقع صفحه‌ی مربوط به نمایش نتایج

برنامه باز می‌شود، عبارت Hello World! روی آن نوشته می‌شود و سپس بلافاصله صفحه بسته می‌شود، زیرا برنامه خاتمه یافته است! در صورتی که اگر این خط وجود داشته باشد، صفحه‌ی نمایش اجرای برنامه باز می‌ماند تا کاربر کلیدی را بزند و در این فاصله، کاربر فرصت دارد خروجی برنامه را که همان متن Hello World! است ببیند. البته getch() کاربردهای دیگری هم دارد که بعداً به آن خواهیم پرداخت. نمونه‌ی اجرای کد ۳-۱ در شکل ۳-۲ آمده است.

شکل ۳-۲: نمونه‌ی خروجی برنامه‌ی کد ۳-۱

در کد ۳-۱ برخی کلمات به صورت پر رنگ نوشته شده‌اند (نظیر int یا return) که به آن‌ها کلمات کلیدی[1] یا رزرو شده[2] می‌گویند، این کلمات قابل استفاده به عنوان نام متغیرها یا توابعی که در زبان C تعریف می‌کنیم نیستند. به این ترتیب اولین برنامه‌ی خود در Dev C++ را نوشته و آن را اجرا کردیم! خروجی برنامه‌های Dev C++ نیز، همان‌طور که در شکل ۳-۲ مشاهده می‌شود، در پنجره‌ای جدید و مشکی رنگ نظیر پنجره‌های DOS نمایش داده می‌شود. قبل از نوشتن برنامه‌ای جدید، ذکر دو نکته‌ی اساسی راجع به زبان C ضروری است:

[1] Keyword
[2] Reserved

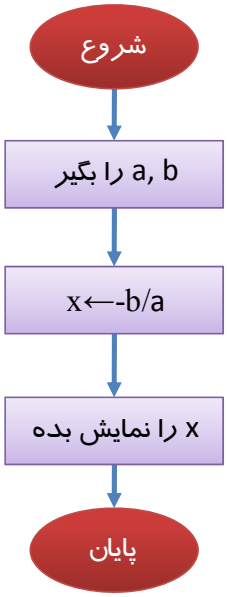
(۱) در همین تعداد کم دستور متوجه شده‌اید که در پایان دستورات در زبان C، نقطه ویرگول[۳] یا همان ; قرار می‌گیرد. این قاعده به صورت کلی برقرار است و تنها چند استثناء دارد، از جمله بعد از خطوط #include که ; قرار نمی‌گیرد، پس از ()int main و نیز پس از { و } هم ; قرار نمی‌گیرد. چند نمونه‌ی دیگر نیز وجود دارد که در جای خود تذکر داده خواهد شد.

(۲) زبان C، برخلاف برخی زبان‌های دیگر مثل BASIC و PASCAL، به بزرگی و کوچکی حروف حساس است و یا اصطلاحاً Case Sensitive است، بنابراین همواره باید بزرگی و کوچکی حروف را رعایت کرد. برای رعایت کردن بزرگی و کوچکی حروف، این قاعده‌ی کلی را مد نظر داشته باشید که کلیه‌ی دستورات با حروف کوچک است، مگر در موارد خاص که به موقع خود معرفی خواهند شد.

پس از ذکر نکاتی که تاکنون رفت، می‌خواهیم یک برنامه‌ی دیگر در زبان C بنویسیم: برنامه‌ی حل معادله‌ی درجه اول. در فصل اول نمودار گردشی حل این مسأله را در شکل ۱-۴ با هم دیدیم، حال می‌خواهیم با کمک این نمودار گردشی به نوشتن برنامه‌ی مربوط به آن بپردازیم.

البته چون در این فصل راجع به ساختارهای تصمیم و چک کردن شرط‌ها مطلبی گفته نمی‌شود و این مطالب مربوط به فصل آینده است، فرض می‌کنیم کاربر در معادله‌ی $ax+b=0$ ، a را مخالف صفر وارد می‌کند و ما دیگر آن را چک نمی‌کنیم. بنابراین نمودار گردشی الگوریتم مسأله نظیر شکل ۳-۳ خواهد شد. برنامه‌ی مربوط به نمودار گردشی شکل ۳-۳ نیز در کد ۳-۲ آمده است.

[3] Semicolon



شکل ۳-۳: نمودار گردشی حل معادله‌ی درجه اول بدون چک کردن شرط $a\neq0$

کد ۳-۲

```
1   #include <iostream>
2   #include <conio.h>
3   using namespace std;
4   int main()
5   {
6      double a,b,x;
7      cout<<"Enter a,b to solve ax+b=0:";
8      cin>>a>>b;
9      x=-b/a;
10     cout<<"x="<<x;
11     getch();
12     return 0;
13  }
```

در کد ۳-۲ چند بخش جدید وجود دارد که باید راجع به آن‌ها توضیحاتی داده شود. اول از همه در خط ۶ برنامه یک عبارت جدید دیده می‌شود، این عبارت مربوط به تعریف متغیر است. در زبان C چنانچه بخواهیم متغیرهایی با اسامی خاصی داشته باشیم، باید آن‌ها را معرفی

کنیم و از آنجا که برنامه‌ی ما توسط مترجم به صورت خط به خط خوانده می‌شود، همیشه باید قبل از استفاده از متغیرها آن‌ها را تعریف کنیم. تعریف متغیر در زبان C در هر جایی از بدنه‌ی اصلی (بین {}) مجاز است، اما توصیه می‌شود کلیه‌ی متغیرها را در همان خطوط اولیه‌ی برنامه تعریف کنید تا هرگونه تغییر در تعداد یا نوع یا نام آن‌ها سریعاً قابل اعمال باشد. دیگر این که همان‌طور که در خط ۶ دیده می‌شود، برای تعریف متغیر باید ابتدا نوع آن (در اینجا double که به معنای عدد اعشاری است) و سپس نام متغیر یا متغیرهایی که می‌خواهیم تعریف کنیم آورده شود. چنانچه بخواهیم در یک خط بیش از یک متغیر تعریف کنیم باید آن‌ها را با ویرگول از هم جدا کنیم. نوع متغیرهایی که در زبان C می‌توان تعریف کرد محدود است و معروف‌ترین آن‌ها int (عدد صحیح) و double (عدد اعشاری) هستند. البته انواع دیگری هم هستند که همگی در شکل ۳-۴ آمده‌اند.

نوع متغیر	اندازه (بایت)	بازه‌ی تعریف
char	1	-128...127
short int	2	-32768...32767
unsigned short int	2	0...65535
int	4	−2,147,483,648...2,147,483,647
long int	4	−2,147,483,648...2,147,483,647
unsigned int	4	0…4294967295
long long int	8	−9,223,372,036,854,775,808… 9,223,372,036,854,775,807
float	4	1.2E-38...3.4E38
double	8	2.2E-308...1.8E308

شکل ۳-۴: انواع متغیرها و بازه‌ی تعریف آن‌ها و سایز آن‌ها به بایت

همان‌طور که دیده می‌شود، نوع اعشاری دیگری با نام float وجود دارد که دقت کمتری نسبت به double دارد و بنابراین معمولاً از double استفاده می‌شود. دامنه‌ی تغییرات انواع متغیرها نیز نشان می‌دهد حداکثر ظرفیت یک نوع برای نگهداری متغیرها چقدر است و این که با توجه به محدودیت‌های مسأله نوع متغیر را انتخاب کنیم. مثلاً اگر قرار است در یک متغیر شماره تلفن ۸ رقمی ذخیره شود، کافی است آن عدد از نوع unsigned long باشد که عددی صحیح و با طول کافیست. متغیرهای unsigned فرقشان با متغیرهای عادی این است که بازه‌ی مثبت آن‌ها تقریباً دو برابر است، ولی در عوض مقدار منفی نمی‌گیرند.

در خط ۷ کد ۳-۲ مشابه آن‌چه که قبلاً هم دیده بودیم، یک پیغام چاپ شده، مبنی بر این که کاربر b و a را وارد کند تا معادله‌ی درجه‌ی اول ax+b=0 حل شود. در خط ۸، دستور جدیدی با نام cin دیده می‌شود که علامت‌های به کار رفته در نوشتن آن (<<) برعکس cout است که بیانگر این است که بر عکس cout که خروجی تولید می‌کند، یعنی بر روی صفحه‌ی نمایش چاپ می‌کند، این دستور عددی از صفحه کلید خوانده و در متغیرهایی که نامشان را روبروی آن نوشته‌ایم قرار می‌دهد. برای cin می‌توان یک متغیر یا بیشتر داشت و چنانچه تعداد متغیرها بیشتر از یکی باشد، باید آن‌ها را با علامت << از هم جدا کرد، درست همان‌طور که در خط ۸ کد ۳-۲ آمده است.

یکی از اشتباهات رایج در برنامه‌نویسی، گذاشتن ویرگول (,) بین متغیرهایی است که در cin آمده‌اند، مثلاً نوشتن cin>>a,b; به جای cin>>a>>b; است که منجر به اجرای غلط برنامه می‌شود و باید به شدت از این کار پرهیز کرد.

ترتیب گرفتن متغیرها از ورودی، به ترتیب از چپ به راست همانی است که در cin آمده، مثلاً اگر بنویسیم cin>>a>>b;، پس از اجرای برنامه اولین عددی که وارد می‌کنیم در a و دومی در b قرار می‌گیرد. به این ترتیب روش گرفتن ورودی از کاربر و قرار دادن آن در یک متغیر از حافظه با یک نام مشخص را یاد گرفتیم.

۲. انجام محاسبات

در خط ۹ کد ۳-۲، یک محاسبه انجام شده است، و حاصل آن در یک متغیر دیگر قرار داده شده است. انجام محاسبات در کامپیوتر بسیار شبیه محاسباتی است که بر روی کاغذ انجام می‌دهیم، با این تفاوت که در کامپیوتر کلیه‌ی عبارات ریاضی باید در یک سطر نوشته شوند و هر عملگری بین اعداد و متغیرهای آن است دقیقاً مشخص شود. مثلاً **نمی‌توان** برای کامپیوتر نوشت x=2b;، زیرا 2b برای کامپیوتر مفهومی ندارد، بلکه برای این که کامپیوتر متوجه منظور ما یعنی عمل ضرب بشود، باید عملگر * را بین 2 و b قرار داد و نوشت x=2*b;. نکته‌ی بسیار مهم بعدی، اولویت عملگرهاست، همان‌طور که می‌دانید عملگرهای * و / (ضرب و تقسیم) بر + و – اولویت دارند، یعنی به طور مثال حاصل عبارت 1+2*5-3 برابر 1+10-3 یعنی 8 خواهد بود، نه برابر 3*5-3 یا 12، یعنی ابتدا حاصل عملگرهای * و / محاسبه شده و سپس + و– محاسبه خواهد شد. البته در اینجا نیز همانند ریاضیات، پرانتز بالاترین اولویت را دارد، پس اگر عبارت قبلی را به (1+2)*5-3 تغییر دهیم، جواب آن برابر 12 خواهد بود، نه 8، زیرا پرانتز اولویت بیشتری نسبت به / و * و + و- دارد. در شکل ۳-۵، مثال‌هایی از عبارات ریاضی مختلف و تبدیل آن به شکل کامپیوتری آمده است.

معادل کامپیوتری	عبارت ریاضی
(a+b)/2-c*d	$\frac{(a+b)}{2}-cd$
(a-b*c)/(5*k)	$\frac{(a-bc)}{5k}$
(a*b-c*d)/(f*g)-5/k	$\frac{ab-cd}{fg}-\frac{5}{k}$

شکل ۳-۵: مثال‌هایی از عبارات ریاضی مختلف و تبدیل آن به شکل کامپیوتری

برای تبدیل عبارات ریاضی به عبارات کامپیوتری باید در نظر داشت، چون * و / نسبت به هم اولویت ندارند، به ترتیب از سمت چپ انجام می‌شوند، به عنوان مثال اگر بنویسیم a/b*c، ابتدا a/b محاسبه شده و سپس کل عبارت ضرب در c می‌شود، یعنی از لحاظ ریاضیاتی $\frac{ac}{b}$ محاسبه می‌شود. برای اعداد صحیح (int، long int و کلیه‌ی انواع صحیح یا غیر اعشاری) یک عملگر دیگر با علامت % وجود دارد که باقی‌مانده را محاسبه می‌کند، به عنوان مثال چنانچه x برابر 17 و y برابر 5 باشد، x%y یا باقی مانده‌ی 17 بر 5، برابر 2 خواهد بود. البته این عملگر برای انواع غیر صحیح وجود ندارد و در صورت استفاده‌ی آن برای متغیرهای اعشاری، با پیغام خطا مواجه می‌شویم.

در اینجا باید به یک نکته توجه کرد و آن این که عملگر / در مورد نوع اعشاری و غیر اعشاری دو کار متفاوت انجام می‌دهد. اگر x و y دو عدد اعشاری باشند، مثلاً x=5.0 و y=2.0 باشد، حاصل x/y برابر 2.5 می‌شود، زیرا هر دو x و y عدد اعشاری هستند، پس حاصل تقسیم آن‌ها نیز اعشاری خواهد شد. اما اگر هر دو x و y صحیح باشند، حاصل این تقسیم برابر ۲ می‌شود. در واقع عملگر / برای حالتی که هر دو عملوند آن عدد صحیح باشد،

نقش محاسبه کننده‌ی خارج قسمت را دارد. دقت کنید حتی اگر بنویسید z=x/y; و z عددی اعشاری باشد، اما x و y هر دو صحیح باشند، عددی که در z قرار می‌گیرد ۲ است، زیرا ابتدا سمت راست تساوی محاسبه می‌شود و بعد در z قرار می‌گیرد. البته اگر یکی از x و y نیز غیر صحیح یا اعشاری باشد، حاصل عملگر تقسیم دیگر خارج قسمت نبوده و یک عدد اعشاری است.

کلیه‌ی عملگرهایی که تاکنون گفتیم دو عملوندی بودند، مثل * و +، یعنی برای ضرب احتیاج به دو عملوند هست، مثلاً می‌نویسیم a*b یا x+y و نمی‌توان نوشت *b یا x+، زیرا به تنهایی معنا ندارند. اما دو عملگر در زبان C هست که عمل افزایش واحد و کاهش واحد را انجام می‌دهند: ++[4] و --[5] که تک عملوندی هستند، مثلاً برای یک واحد افزایش x کافی است بنویسیم x++; و یا ++x;، همین‌طور برای کاهش واحد نیز می‌توان نوشت x--; و یا --x; . این که ++ یا -- قبل یا بعد متغیر بگذاریم در نهایت بر روی خود متغیر اثری ندارد، یعنی نتیجه‌ی نهایی یک واحد افزایش و یا یک واحد کاهش متغیر خواهد بود، اما به طور مثال اگر بنویسیم y=x++; و مقدار x قبل از اجرای این خط ۷ باشد، ابتدا مقدار ۷ وارد y شده و سپس x یکی اضافه می‌شود، یعنی پس از اجرای این خط y برابر ۷ و x برابر ۸ خواهد شد، اما اگر بنویسیم y=++x;، ابتدا x یک واحد اضافه شده و سپس درون y قرار می‌گیرد، یعنی پس از اجرای این خط x برابر ۸ و y نیز برابر ۸ خواهد بود. مشابه این قاعده برای -- نیز برقرار است.

[4] Plus Plus
[5] Minus Minus

هنگام استفاده از ++ یا --، باید یا آن را قبل و یا بعد از متغیر نوشت و نمی‌توان هم‌زمان هر دو را استفاده کرد و دو واحد افزایش داشت، یعنی ;++x++ یک دستور غلط و تعریف نشده است و نوشتن چنین دستوری باعث بروز پیغام خطا خواهد شد.

از دو عملگر ++ و-- در آینده استفاده‌ی فراوانی خواهیم کرد.

۳. خلاصه نویسی در عبارات محاسباتی

تا کنون با انواع محاسبات و متغیرها و نوشتن عبارات ریاضی به زبان C در کامپیوتر آشنا شده‌اید، اکنون می‌خواهیم با خلاصه نویسی در زبان C آشنا شویم. در بسیاری از عبارات ریاضی که در برنامه‌ها می‌نویسیم، می‌خواهیم مقدار یک متغیر را با توجه به مقدار قبلی خود آن متغیر عوض کنیم، مثلاً می‌خواهیم x هر چه که بوده، ۵ واحد به آن اضافه شود. مطابق آن‌چه که تاکنون یاد گرفته‌ایم باید بنویسیم:

```
x=x+5;
```

یا اگر بخواهیم مقدار x را دو برابر کنیم، باید بنویسیم:

```
x=x*2;
```

اما چنین عبارت‌هایی را در زبان C می‌توان خلاصه‌تر نوشت، مثلاً دو عبارت اخیر را می‌توان به ترتیب به صورت x+=5; و x*=2; نوشت. به طور کلی x□=y; در زبان C یعنی x=x□y; که □ می‌تواند هر کدام از عملگرهای * و / و + و – و % باشد. البته y لزوماً یک

متغیر یا عدد نیست و می‌تواند یک عبارت باشد، مثلاً می‌توانیم بنویسیم x*=(2*i+1)*k; که به این معناست که محتوای قبلی x هر چه که هست در (2*i+1)*k ضرب شود.

۴. مربع مجموع سه رقم

اکنون می‌خواهیم برنامه‌ای بنویسیم که اعداد صحیح a و b و c را از ورودی خوانده و حاصل عبارت $(a+b+c)^2$ را محاسبه و چاپ کند. از آنجا که $(a+b+c)^2=(a+b+c)*(a+b+c)$ می‌توانیم برنامه را به صورت کد ۳-۳ بنویسیم (دقت کنید در زبان C عملگر توان وجود ندارد و باید توان را با استفاده از ضرب ساخت. البته C دستور pow(x,y) را در <math.h> برای محاسبه‌ی x به توان y دارد که مربوط به اعداد اعشاری است و برای محاسبات اعداد صحیح توصیه نمی‌شود).

کد ۳-۳

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     int a,b,c,m;
7     cout<<"Enter a,b,c:";
8     cin>>a>>b>>c;
9     m=(a+b+c)*(a+b+c);
10    cout<<"("<<a<<"+"<<b<<"+"<<c<<")^2="<<m;
11    getch();
12    return 0;
13 }
```

با توضیحاتی که تا کنون داده شده، نباید ابهامی در کد ۳-۳ وجود داشته باشد، فقط در مورد خط ۱۰ و cout که خروجی تولید می‌کند باید این نکته ذکر شود که اگر بخواهیم عبارتی عیناً به خروجی منتقل شود، آن را بین " " قرار می‌دهیم، مثل Hello World! که وقتی بین " " قرار می‌گرفت (کد ۳-۱) عیناً به خروجی منتقل شده و روی صفحه نمایش چاپ می‌شد. اما اگر بخواهیم مقدار یک متغیر در خروجی چاپ شود، نباید آن را بین " " قرار دهیم، زیرا مثلاً اگر بنویسیم cout<<"a"، همواره a در خروجی چاپ می‌شود و نه مقدار a، بنابراین چنانچه بخواهیم یک خروجی تولید کنیم که هم بخش‌های ثابت داشته باشد و هم بخش‌های متغیر که باید مقدارشان چاپ شود، باید بخش‌های مختلف را با استفاده از عملگر << از یکدیگر جدا کنیم.

به برعکس بودن علامت در cin و cout دقت کنید! cout با << و cin با >> می‌آید. برعکس نوشتن این علایم باعث بروز خطا می‌شود. همچنین یک خطای رایج در بین برنامه‌نویسان تازه کار نوشتن دستور cout به صورت cout<<"Hello World!"<< است، زیرا تصور می‌کنند به خاطر تقارن باید << را با >> ببندند که صحیح نیست. cout همواره با << و cin همواره با >> همراه می‌شود.

۵. نکات تکمیلی در مورد متغیرها

تا کنون تقریباً همه‌ی نکات لازم در مورد متغیرها و محاسبات را در زبان C آموخته‌اید. در این بخش می‌پردازیم به چند نکته‌ی پایانی و این فصل را پایان می‌دهیم. اول این که تا کنون

هرچه محاسبه کردیم و نتیجه‌ی محاسبه را در یک متغیر دیگر قرار دادیم، همه‌ی متغیرها از یک نوع بودند، مثلاً همه double یا همه int بودند، حال می‌خواهیم ببینیم چنانچه یک متغیر از نوع متفاوت را مساوی نوع دیگر قرار دهیم چه اتفاقی می‌افتد؟ آن‌چه که در ابتدا به ذهن می‌رسد این است که چون عدد اعشاری حالت کلی‌تر اعداد صحیح است پس بدون هیچ مشکلی عدد صحیح داخل متغیر int قرار می‌گیرد. البته این تصور تا حدود زیادی درست است، مگر در مورد اعداد صحیح خیلی بزرگ که دقت double توانایی نگهداری همه‌ی ارقام آن را ندارد و تخمینی از آن را نگه می‌دارد، مثلاً زمانی که یک long long int تعریف کرده و یک عدد ۱۵ رقمی در آن قرار دهیم (مثلاً 123456789012345) و سپس یک عدد double را مساوی آن قرار دهیم و هر دو را چاپ کنیم، عدد double به صورت 1.23457e04 چاپ می‌شود که همان $1.23457*10^{14}$ است.

در برنامه نویسی به زبان C استفاده از نماد علمی آزاد است، مثلاً می‌توان یک متغیر مثل x را برابر 2.15e3 قرار داد که همان 2150 است. همچنین توان‌های منفی 10 را نیز می‌توان استفاده کرد، مثلاً اگر بنویسیم x=1e-3، x برابر 0.001 خواهد شد و البته در این حالت x باید اعشاری باشد.

حال چنانچه یک متغیر اعشاری را در یک متغیر غیر اعشاری قرار دهیم، اولاً بخش اعشاری آن حذف می‌شود، مثلاً اگر بنویسیم x=2.7; و x متغیری از نوع int باشد، مقدار 2 وارد x می‌شود. اگر هم مقدار متغیر اعشاری خیلی بزرگ باشد یک عدد وارد x می‌شود که لزوماً ربطی به عدد نوشته شده ندارد. مثلاً چنانچه بنویسیم x=1e103 که یک عدد بسیار بزرگ

است (۱۰۴ رقم دارد!)، و سپس x را چاپ کنیم در خروجی ممکن است عدد 2147483647 چاپ شود که هیچ ربطی به 1e103 ندارد! پس در نوشتن عملگر = باید دقت لازم را داشت.

یک نکته‌ی بسیار مهم در زبان C که فرق اساسی آن با ریاضیات است، این است که x=y; به هیچ وجه به معنای x برابر y است نیست، یعنی ما همانند ریاضیات معادله نمی‌نویسیم، بلکه این عبارت یک جمله‌ی دستوری است: مقدار y را در x قرار بده. یعنی این عمل کاملاً یک‌طرفه است (از راست به چپ)، که یک طرفه بودن آن را در فصل دوم و نمودار گردشی دیدیم، که قرار دادن یک مقدار یا نتیجه‌ی یک محاسبه را در یک متغیر دیگر با ← نمایش می‌دادیم، یعنی هر آنچه در سمت راست وجود دارد (خواه یک عدد ثابت است، خواه یک متغیر و یا یک عبارت ریاضی) را در متغیری که در سمت چپ وجود دارد قرار بده. بنابراین علیرغم مفهومی که در ریاضیات وجود دارد و در آن x=y با y=x یک مفهوم دارند، در برنامه‌نویسی x=y; با y=x; کاملاً متفاوت هستند. همچنین برخی عبارات که در ریاضیات نوشتن آن‌ها بلا مانع است، نظیر (i+j)=k، در زبان C منجر به بروز خطا می‌شود، زیرا در سمت چپ عملگر = باید همواره، تنها و تنها یک متغیر وجود داشته باشد و نه هیچ چیز دیگری، زیرا قرار است نتیجه‌ی عبارت سمت راست = داخل آن قرار بگیرد.

نکته‌ی آخر در مورد نام‌گذاری متغیرهاست، نام‌گذاری متغیرها در برنامه‌نویسی نیز تابع قوانین خاصی است و نمی‌توان هر اسمی را برای هر متغیری انتخاب کرد. اسامی متغیرها تنها می‌تواند با حروف بزرگ یا کوچک (a تا z یا A تا Z) و یا _(زیرخط[۶]) شروع شود، حروف دوم به بعد متغیرها، علاوه بر کاراکترهای گفته شده، می‌تواند شامل اعداد 0 تا 9 نیز باشد. در شکل ۳-۶ نمونه‌هایی از اسامی مجاز و غیرمجاز در زبان C دیده می‌شود.

[6] Underline

اسامی مجاز	اسامی غیرمجاز
x01_02 _x_1 width coordinate_x	0y x#1 0c$1 (x_2)

شکل ۳-۶: نمونه‌هایی از اسامی مجاز و غیرمجاز در زبان C

در پایان این فصل ذکر این نکته ضروری است که نوشتن برنامه‌ها معمولاً با خطا همراه است و خیلی کم پیش می‌آید که با همان نوشتن بار اول، برنامه بدون خطا اجرا شود، و یا نتیجه‌ی درستی در بر داشته باشد. توصیه می‌شود برای آشنایی با نحوه‌ی اشکال‌زدایی برنامه‌های خود به ضمیمه‌ی اول کتاب مراجعه کرده و در حد مطالب تدریس شده تا اینجا با نحوه‌ی اشکال‌زدایی برنامه آشنا شوید و مطالبی از این ضمیمه که مربوط به دروس فصول آینده است را نیز پس از فراگیری آن، مطالعه نمایید.

فصل چهارم
مالکیت صفحه

مدیریت صفحه کلید، صفحه نمایش و آشنایی با ساختارهای تصمیم و تکرار

۱. نوع کاراکتری

تا کنون در مورد اصول برنامه‌نویسی در C و راه‌های گرفتن ورودی و تولید خروجی صحبت شد، همچنین انواع متغیرها و عملگرهای آن‌ها در فصل پیش مورد بررسی قرار گرفت. در این بخش قصد داریم یکی از انواع متغیرهایی که در فصل قبل مطرح شده را بیشتر معرفی کنیم: نوع کاراکتر که با char مشخص می‌شود. اصولاً نوع کاراکتر، همان نوع int است و تقریباً می‌توانند به جای هم استفاده شوند، فقط کاراکتر دو تفاوت با int دارد:

(۱) نوع کاراکتر ظرفیت کمتری دارد و تنها یک بایت اندازه دارد.

(۲) در موقع چاپ نوع کاراکتر، به جای چاپ عدد، معادل اَسکی[1] آن چاپ می‌شود.

تفاوت اول که مشخص است، یعنی حداکثر تعداد اعدادی که نوع کاراکتر می‌تواند نگه دارد ۲۵۶ عدد است. در مورد تفاوت دوم، کمی به توضیح احتیاج است. در کامپیوترهای اولیه که مفهوم قلم[2] های مختلف به وسعت امروز وجود نداشت و در محیط متنی، تنها یک قلم برای چاپ بر روی صفحه‌ی نمایش وجود داشت، برای نگه داشتن متن یا نمایش آن از یک جدول کد بندی استفاده می‌شد که به هر حرف یا علامت، یک عدد نسبت می‌داد، مثلاً به A، عدد 65، به B عدد 66 و... بنابراین اگر کسی می‌خواست روی صفحه‌ی نمایش رشته‌ی ABC456: را بنویسد، باید اعداد

A	B	C	4	5	6

65	66	67	52	53	54

[1] ASCII
[2] Font

را در حافظه‌ی نمایشی کامپیوتر می‌نوشت تا کامپیوتر این اعداد را بر روی مانیتور، بر اساس کد اَسکی آن‌ها نمایش دهد. این جدول از زمان‌های قدیم تا به امروز در حافظه‌ی کامپیوترها وجود داشته است. حال اگر شما یک متغیر از نوع کاراکتر را، برابر عددی مثل 65 قرار دهید و سپس بخواهید با cout آن را چاپ کنید، به جای چاپ عدد 65، معادل حرفی آن را از جدول اَسکی پیدا کرده و آن را چاپ می‌کند (یعنی حرف A). چنانچه بخواهیم عدد واقعی داخل کاراکتر چاپ شود، باید هنگام cout آن را به صورت موقت به int تبدیل کنیم، یعنی به مترجم زبان C بگوییم موقتاً با این متغیر کاراکتری، مثل int رفتار کند. برای درک بهتر مطالبی که تا کنون گفته شد، یک برنامه می‌نویسیم. ابتدا به برنامه‌ی نوشته شده در کد ۴-۱ دقت کنید:

کد ۴-۱

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     char ch=65;
7     cout<<ch<<endl;
8     ch++;
9     cout<<"Code of character "<<ch<<"="<<int(ch);
10    getch();
11    return 0;
12 }
```

در خط ۶ کد ۴-۱، ابتدا یک کاراکتر به نام ch تعریف شده و مقدار آن برابر 65 قرار گرفته که در نتیجه‌ی چاپ آن، حرف A در خروجی نمایش داده می‌شود. سپس ch با عملگر ++

یک واحد اضافه شده تا نشان داده شود این نوع به لحاظ محاسباتی، فرقی با int نداشته و کلیه‌ی عملیات ریاضی از جمله ضرب، جمع، تقسیم و تفریق (و البته باقی‌مانده) نیز بر روی آن قابل انجام است. سپس در خط ۹ با نوشتن int(ch) به جای خود ch، محتوای عددی ch چاپ می‌شود.

۲. مدیریت صفحه کلید

یکی از ابزارهای ورودی کامپیوتر صفحه کلید است. تا کنون برای گرفتن ورودی از کاربر و قرار دادن آن در یک متغیر، از cin استفاده می‌کردیم و کاربر پس از وارد کردن یک عدد و سپس زدن دکمه‌ی Enter، ورودی مورد نظر خود را وارد می‌کرد. حال اگر بخواهیم کاربر بدون زدن دکمه‌ی Enter و تنها فشردن کلیدی از صفحه کلید، به برنامه ورودی بدهد، دیگر نمی‌توانیم از cin استفاده کنیم. دستوری که باید استفاده کنیم دستور جدیدی نیست و قبلاً هم از آن استفاده کرده‌ایم: دستور getch() . تا به حال از getch() برای انتظار برنامه برای زدن یک کلید توسط کاربر استفاده می‌کردیم، اما به این نکته اشاره نکردیم که getch() علاوه بر منتظر شدن برای زدن یک کلید توسط کامپیوتر، می‌تواند کد آن را نیز برگرداند. به عنوان مثال به کد ۴-۲ دقت کنید:

کد ۴-۲

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
```

```
6       char ch;
7       ch=getch();
8       cout<<"The pressed key is:"<<ch<<endl;
9       cout<<"and its code is:"<<int(ch)<<endl;
10      getch();
11      return 0;
12  }
```

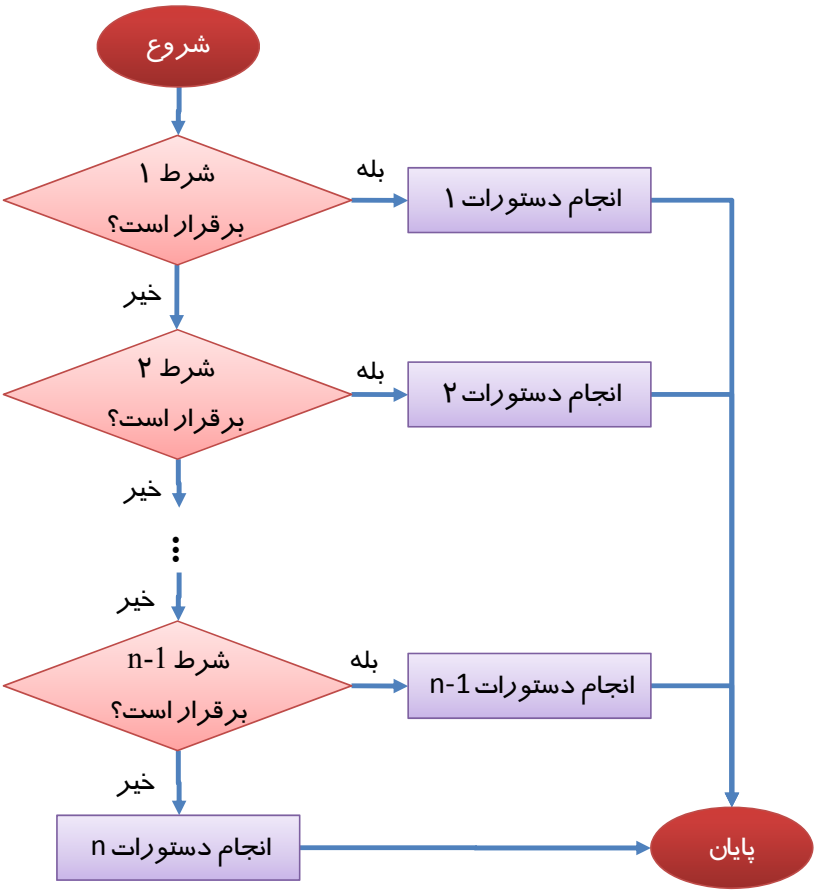
در getch() اول در خط ۷، کد کلید فشرده شده بلافاصله در ch قرار گرفته و ادامه‌ی برنامه اجرا می‌شود و دیگر نیازی به زدن کلید Enter نیست و می‌توان یک کنترل پیوسته بر روی برنامه داشت. کلیه‌ی برنامه‌هایی که نظیر بازی‌ها از صفحه کلید به طور مداوم استفاده می‌کنند و احتیاجی هم به فشردن کلید Enter ندارند، از فرایندی مشابه استفاده می‌کنند.

حال فرض کنید می‌خواهیم با توجه به کلیدی که کاربر فشار داده، تصمیم خاصی بگیریم، مثلاً اگر کاربر کلید g را فشرد، دو عدد از کاربر گرفته و جمع آن‌ها را چاپ کنیم و اگر کاربر کلید h را فشرد، سه عدد از کاربر گرفته و آن‌ها را ضرب کنیم و اگر کاربر کلید دیگری را زد پیغامی مبنی بر معتبر نبودن کلید فشرده شده چاپ شود و برنامه به اتمام برسد. اما تا کنون ساختاری که بر اساس شرطی، کاری را انجام بدهد یا ندهد، معرفی نکرده‌ایم. بنابراین باید ابتدا با نحوه‌ی ایجاد ساختارهای تصمیم در زبان C آشنا شویم.

۳. ساختار تصمیم if

ساختارهای تصمیم در C دو نوعند: if و switch . ابتدا مفهوم if را بررسی می‌کنیم، پس از آن فهم دستور switch بسیار آسان خواهد شد. ساختار کلی if به صورت نمودار گردشی در شکل ۴-۱ نمایش داده شده است.

البته در شکل ۴-۱، شروع و پایان مربوط به ساختار if است و می‌تواند قسمت قبل و بعد if در برنامه‌ی اصلی باشد. نحوه‌ی نوشتار if نیز بسیار ساده است. برای بیان نحوه‌ی نوشتار if در برنامه، فرض کنید می‌خواهیم عددی از کاربر بگیریم، اگر کوچک‌تر از 0 بود عبارت Negative، اگر بزرگ‌تر از 0 بود عبارت Positive و اگر صفر بود عبارت Zero بر روی صفحه چاپ شود.



شکل ۴-۱: نمودار گردشی ساختار کلی if

کد ۴-۳

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     int x;
```

```
7     cout<<"Enter a number:";
8     cin>>x;
9     if(x<0)
10       cout<<"Negative";
11    else if(x>0)
12       cout<<"Positive";
13    else
14       cout<<"Zero";
15    getch();
16    return 0;
17 }
```

همان‌طور که در کد ۴-۳ دیده می‌شود، (شرط)if، نحوه‌ی نوشتن شرط و ایجاد ساختار تصمیم در برنامه است.

چنانچه به ازای برقراری یک شرط، تعداد دستوراتی که باید انجام بدهیم بیش از یک دستور است، باید دستورها را در یک بلوک قرار دهیم، یعنی بین{} تا کلیه‌ی دستورات اجرا شوند. توصیه می‌شود در زمانی که حتی یک دستور هم در ساختار if می‌نویسید آن را در بلوک قرار دهید تا اگر بعداً خواستید دستوری به آن اضافه کنید، راحت باشید.

حال می‌توانیم برنامه‌ای را که در انتهای بخش قبل قصد نوشتن آن را داشتیم بنویسیم. متن این برنامه در کد ۴-۴ آمده است.

کد ۴-۴

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     int a,b,c;
7     char ch;
8     ch=getch();
9     if(ch=='g')
10    {
11       cout<<"Enter 2 numbers:";
12       cin>>a>>b;
13       cout<<a<<"+"<<b<<"="<<a+b;
14    }
15    else if(ch=='h')
16    {
17       cout<<"Enter 3 numbers:";
18       cin>>a>>b>>c;
19       cout<<a<<"*"<<b<<"*"<<c<<"="<<a*b*c;
20    }
21    else
22    {
23       cout<<"The pressed key was invalid...";
24    }
25    getch();
26    return 0;
27 }
```

همان‌طور که در کد ۴-۴ در خطوط ۹ و ۱۵ دیده می‌شود، برای چک کردن تساوی، از دو علامت مساوی (==) استفاده می‌شود و استفاده از = منجر به اجرای غلط برنامه خواهد شد و نتایج اشتباه حاصل می‌شود. همچنین در این خطوط برای چک کردن تساوی کاراکتر با g و h، به جای کد آن‌ها، از خود g و h استفاده کرده‌ایم، منتها آن‌ها را بین ' ' قرار داده‌ایم. دقت کنید نباید g و h را برای نوع کاراکتری بین " " قرار داد، زیرا منجر به ایجاد پیغام خطا شده و برنامه اجرا نخواهد شد.

نکته‌ی دیگری که شاید در نوشتن این برنامه متوجه آن شده باشید این است که چون ما کاراکتر ورودی از صفحه کلید را با g و h که حروف کوچک هستند مقایسه کردیم، اگر به دلیلی G و H توسط کاربر وارد شدند (مثلاً Caps Lock روشن باشد)، دیگر شرط if درست نخواهد بود، چون G با g فرق دارد (کد اَسکی آن‌ها یکسان نیست). برای این که این مشکل را نیز حل کنیم، با استفاده از دستور chartoupper و یاchartolower که در <nodet.h> قرار دارد، کاراکتر ورودی را به حرف بزرگ و یا حرف کوچک تبدیل می‌کنیم.

در هنگام استفاده از دستورهای chartoupper(ch) و chartolower(ch)، چنانچه ch یک کاراکتر حرفی باشد، آن را در صورت لزوم به بزرگ یا کوچک تغییر می‌دهد و در غیر این صورت کاراکتر ch را بدون تغییر برمی‌گرداند.

برنامه‌ی تغییر یافته‌ی کد ۴-۴ که به بزرگی و کوچکی حروف حساس نباشد در کد ۴-۵ آمده است.

کد ۴-۵

```
1  #include <iostream>
2  #include <conio.h>
3  #include <nodet.h>
4  using namespace std;
5  int main()
6  {
7     int a,b,c;
```

```
8     char ch;
9     ch=getch();
10    if(chartolower(ch)=='g')
11    {
12       cout<<"Enter 2 numbers:";
13       cin>>a>>b;
14       cout<<a<<"+"<<b<<"="<<a+b;
15    }
16    else if(chartolower(ch)=='h')
17    {
18       cout<<"Enter 3 numbers:";
19       cin>>a>>b>>c;
20       cout<<a<<"*"<<b<<"*"<<c<<"="<<a*b*c;
21    }
22    else
23    {
24       cout<<"The pressed key was invalid...";
25    }
26    getch();
27    return 0;
28 }
```

اکنون در این قسمت، می‌پردازیم به تکمیل مبحث ساختارهای تصمیم. نخست باید لیست کاملی از عملگرهای مقایسه‌ای ریاضی را که در شرط به کار می‌روند مرور کنیم. این لیست در زیر آمده است:

عملگر	مفهوم	مثال
==	تساوی	x==y
!=	عدم تساوی	x!=0
<	کوچک‌تر بودن	a<b+c

>	بزرگ‌تر بودن	z>x*y
<=	کوچک‌تر یا مساوی بودن	x<=5
>=	بزرگ‌تر یا مساوی بودن	ch>="A"

همچنین می‌توان شروط مختلف را با هم ترکیب کرد. مثلاً اگر بخواهیم چک کنیم که $0 \leq x \leq 5$ است یا خیر، نباید بنویسیم if(0<=x<=5)، بلکه باید از شرط ترکیبی استفاده کنیم، یعنی بنویسیم اگر x>=0 وَ x<=5 است، دستورات مربوطه اجرا شوند. به لحاظ منطقی، سه نوع ترکیب اساسی داریم: وَ – یا – نقیض، چنانچه دو دستور با وَ (AND) با هم ترکیب شوند، باید هر دو درست باشند تا نتیجه‌ی نهایی درست باشد و اگر دو دستور با یا (OR) ترکیب شوند، کافیست یکی از آن‌ها درست باشد تا نتیجه‌ی نهایی درست باشد. نقیض نیز یک عملگر تکی است و چنانچه عبارت درست باشد آن را نادرست می‌کند و بالعکس. در زیر جدول درستی[3] عملگرهای وَ و یا آمده است:

p	q	p و q
نادرست	نادرست	نادرست
درست	نادرست	نادرست
نادرست	درست	نادرست
درست	درست	درست

p	q	p یا q
نادرست	نادرست	نادرست
درست	نادرست	درست
نادرست	درست	درست
درست	درست	درست

[3] Truth table

در زبان C برای نوشتن وَ و یا از علامت‌های && و || استفاده می‌شود که اولی با shift+7 و دومی با shift+\ (البته هرکدام دو بار) نوشته می‌شود. به جدول زیر که مثال‌هایی از شروط ترکیبی و مفاهیم آن‌ها را نمایش می‌دهد دقت کنید:

شرط ترکیبی	مفهوم
`(x>=0) && (x<=5)`	x بزرگ‌تر مساوی 0 و کوچک‌تر مساوی 5 باشد
`! ( (x>=0) && (x<=5) )`	x بزرگ‌تر مساوی 0 و کوچک‌تر مساوی 5 نباشد
`(x==7) || ( (x>=0) && (x<=5) )`	x بزرگ‌تر مساوی صفر و کوچک‌تر مساوی 5 یا x برابر 7 باشد
`(x==2) || (y!=5)`	x برابر 2 باشد یا y مخالف 5 باشد

پرانتز گذاری در تعیین نتیجه‌ی نهایی شرط‌های ترکیبی بسیار اهمیت دارد، بنابراین در هنگام نوشتن پرانتزها در شرط‌های ترکیبی حداکثر دقت را داشته باشید تا نتیجه‌ی نهایی مطلوب نظر شما باشد.

این بخش را با یک مثال دیگر به پایان می‌بریم، فرض کنید می‌خواهیم سه عدد a,b,c را از ورودی گرفته و جواب‌های معادله‌ی درجه دوم $ax^2+bx+c=0$ را در صورت وجود محاسبه و چاپ کنیم. می‌دانیم معادله‌ی درجه دوم حداکثر دو جواب دارد که تعداد جواب‌ها بسته به علامت دلتا (Δ) یا مبین آن دارد: $\Delta=b^2-4ac$. اگر $\Delta>0$ باشد، دو جواب متفاوت داریم:

$$x_1 = \frac{-b + \sqrt{\Delta}}{2a} \qquad x_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

اگر $\Delta=0$ باشد، دو جواب با هم برابرند:

$$x_1 = x_2 = \frac{-b}{2a}$$

و اگر $\Delta<0$ باشد معادله جواب ندارد. برای این که نوشتن و فهم این برنامه کمی راحت‌تر شود، ابتدا نمودار گردشی الگوریتم حل این مسأله را در شکل ۴-۲ ببینید. البته در هنگام برنامه‌نویسی باید توجه کنید که از نمادهایی مثل Δ نمی‌توان استفاده کرد و به جای آن‌ها از حروف انگلیسی مثل D استفاده می‌کنیم. متن برنامه نیز در کد ۴-۶ نوشته شده است. در نوشتن کد ۴-۶، از تابع sqrt که رادیکال پارامتر ورودی خود را حساب می‌کند استفاده شده که در <math.h> قرار دارد.

کد ۴-۶

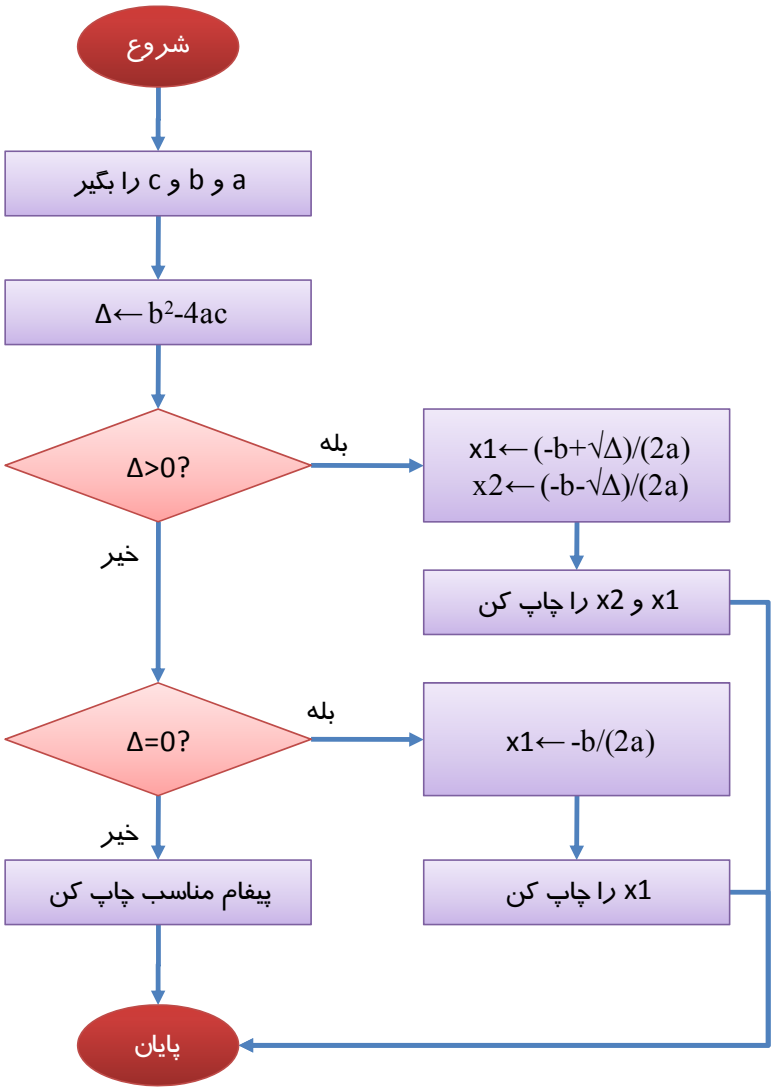
```
1  #include <iostream>
2  #include <conio.h>
3  #include <math.h>
4  using namespace std;
5  int main()
6  {
7     double a,b,c,x1,x2,D;
8     cout<<"Enter a,b,c to solve ax^2+bx+c=0 :";
```

```
9     cin>>a>>b>>c;
10    D=b*b-4*a*c;
11    if(D>0)
12    {
13        x1=(-b+sqrt(D))/(2*a);
14        x2=(-b-sqrt(D))/(2*a);
15        cout<<"Equation has two different
      answers:"<<endl;
16        cout<<"x1="<<x1<<", x2="<<x2<<endl;
17    }
18    else if(D==0)
19    {
20        x1=(-b+sqrt(D))/(2*a);
21        cout<<"Equation has two repeated answers:"<<endl;
22        cout<<"x1=x2="<<x1<<endl;
23    }
24    else
25    {
26        cout<<"Equation has no answer..."<<endl;
27    }
28    getch();
29    return 0;
30 }
```

وب

<math.h> توابع ریاضی زیادی دارد که برخی از آن‌ها را می‌توانیـد در آدرس زیر ببینید:

http://www.cplusplus.com/reference/clibrary/cmath

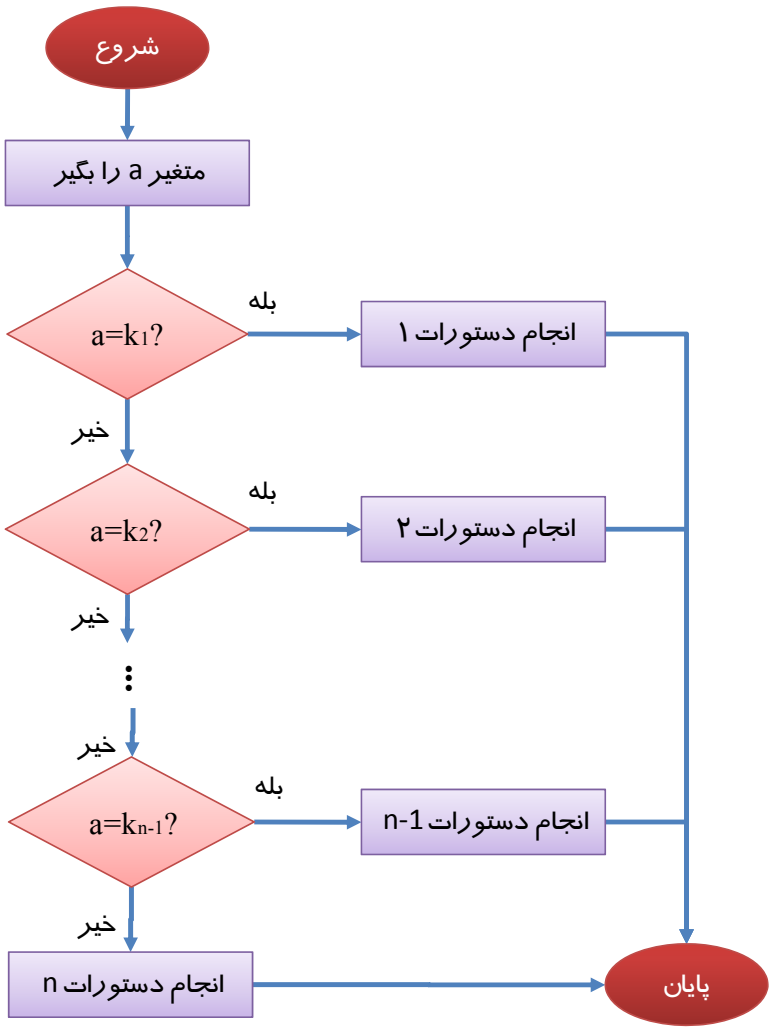


شکل ۴-۲: نمودار گردشی الگوریتم حل مسأله‌ی معادله‌ی درجه دو

۴. ساختار تصمیم switch

در کنار ساختار تصمیم if، ساختار تصمیم دیگری وجود دارد که در برخی مواقع برای ساده سازی در نوشتار به جای if به کار می‌رود. این ساختار تصمیم برای تصمیم‌گیری بر روی مقادیر مختلف یک متغیر طراحی شده است. در شکل ۴-۳ نمودار گردشی ساختار تصمیم switch ترسیم شده است.

مجدداً همان‌طور که در مورد if گفتیم، شروع و پایان در نمودار گردشی شکل ۴-۳ مربوط به شروع و پایان switch هستند و ممکن است برنامه‌ی مربوط به قبل و بعد switch به جای آن‌ها قرار بگیرد. در ضمن گرفتن متغیر a به منظور ورودی بودن آن برای دستور switch است. در واقع دستور switch همانند یک ساختار if عمل می‌کند که تنها شرط‌های برابری یک متغیر با مقادیر مختلف را بررسی می‌کند.



شکل ۴-۳: نمودار گردشی ساختار تصمیم switch

برای فهم هرچه بهتر موضوع یک مثال را در نظر بگیرید: فرض کنید می‌خواهیم برنامه‌ای بنویسیم که منتظر فشرده شدن یک کلید از صفحه کلید باشد، سپس، اگر کلید فشرده شده W بود، بر روی صفحه بنویسد Up، اگر کلید فشرده شده S بود، بر روی صفحه بنویسد Down و اگر کلیدهای D و A زده شدند، به ترتیب عبارات Right و Left را روی صفحه بنویسد و چنانچه کلید فشرده شده هیچ کدام از این چهار کلید نبود بر روی صفحه عبارت I don't know چاپ شود. برنامه‌ی مربوط به این صورت مسأله در کد ۴-۷ آمده است:

کد ۴-۷

```
1  #include <iostream>
2  #include <conio.h>
3  #include <nodet.h>
4  using namespace std;
5  int main()
6  {
7     char ch;
8     ch=getch();
9     switch(chartolower(ch))
10    {
11       case 'w':
12          cout<<"Up"<<endl;
13       break;
14       case 's':
15          cout<<"Down"<<endl;
16      break;
17      case 'a':
18          cout<<"Left"<<endl;
19      break;
20      case 'd':
21          cout<<"Right"<<endl;
22      break;
23      default:
24          cout<<"I don't know"<<endl;
25      break;
26    }
```

```
27      getch();
28      return 0;
29  }
```

همان‌طور که در کد ۴-۷ در خط ۲۳ می‌بینید، برای اجرای دستور در صورت برقرار نبودن هیچ‌کدام از حالت‌ها، از default استفاده می‌شود. دقت کنید که برای ارسال پارامتر switch، آن را تبدیل به حروف کوچک کرده‌ایم.

عبارت break; پس از پایان دستورات هر حالت از تساوی در case های یک switch باید نوشته شود، در غیر این صورت کلیه‌ی شرط‌ها با هم OR می‌شوند و برنامه غلط خواهد شد.

البته همان‌طور که قبلاً هم گفته شد، ساختار switch را با if نیز می‌توان پیاده‌سازی کرد. به عنوان مثال، برنامه‌ی مربوط به کد ۴-۷ که با if نوشته شده را در کد ۴-۸ مشاهده می‌کنید.

کد ۴-۸

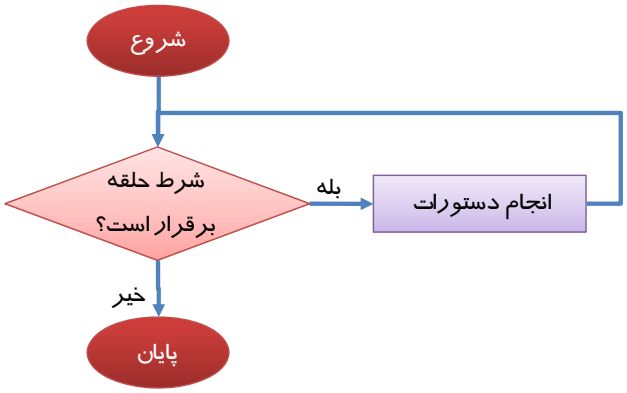
```
1   #include <iostream>
2   #include <conio.h>
3   #include <nodet.h>
4   using namespace std;
5   int main()
6   {
7      char ch;
8      ch=getch();
9      if(chartolower(ch)=='w')
10     cout<<"Up"<<endl;
```

```
11      else if(chartolower(ch)=='s')
12         cout<<"Down"<<endl;
13      else if(chartolower(ch)=='a')
14         cout<<"Left"<<endl;
15      else if(chartolower(ch)=='d')
16         cout<<"Right"<<endl;
17      else
18         cout<<"I don't know"<<endl;
19      getch();
20      return 0;
21  }
```

استفاده از if یا switch وابسته به موقعیت است، معمولاً زمانی که شرط ترکیبی داریم و یا شروط مربوط به حالت بزرگ‌تر یا کوچک‌تر بودن است، از if و زمانی که برابری یک متغیر با مقادیر مختلف را می‌خواهیم بیازماییم از switch استفاده می‌کنیم.

۵. ساختار تکرار while

در معرفی ساختار تصمیم if گفته شد که اگر شرط if درست باشد، کد مربوط به آن اجرا می‌شود و اگر آن شرط نادرست باشد، کدهای مربوط به if اجرا نخواهند شد. حال اگر بخواهیم تا زمانی که شرط درست است، یک سری از دستورات اجرا شوند، چه کار باید بکنیم؟ راه حل استفاده از حلقه‌ی while است. حلقه‌ی while درست مانند if است، else ندارد و تا زمانی که شرط درست است، دستورات را اجرا می‌کند. نمودار گردشی ساختار کلیwhile در شکل ۴-۴ آمده است:



شکل ۴-۴: نمودار گردشی ساختار کلی while

به عنوان مثال فرض کنید می‌خواهیم برنامه‌ی کد ۴-۲ را به گونه‌ای تغییر دهیم که تا زمانی که کلید فشرده شده برابر Esc نیست منتظر فشرده شدن یک کلید از صفحه کلید شده و کد آن را چاپ کند. متن مربوط به این برنامه در کد ۴-۹ آمده است:

کد ۴-۹

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     char ch=0;
7     while(ch!=27)
8     {
9        ch=getch();
10       cout<<"The pressed key is:"<<ch<<endl;
11       cout<<"and its code is:"<<int(ch)<<endl;
12    }
13    return 0;
14 }
```

در متن برنامه‌ی نوشته شده، در کد ۴-۹، چند نکته باید توضیح داده شود. اول این که زمان تعریف متغیر کاراکتری ch در خط ۶، بلافاصله مقدار آن برابر صفر قرار گرفته است. دلیل این امر آن است که چنانچه متغیرها را در زبان C مقداردهی اولیه نکنیم، مقدار آن‌ها یک عدد تصادفی است، از طرفی شرط while برای اجرا شدن در این برنامه این است که ch برابر 27 (27 کد کلید Esc است) نباشد، بنابراین، برای این که حتماً برای بار اول وارد while بشویم، یعنی بار اول حتماً شرط while درست باشد که حداقل یک بار اجرا شود، مقدار ch را برابر عددی مخالف 27 می‌گذاریم تا مطمئن باشیم ch به طور تصادفی نیز برابر 27 نخواهد بود.

این بخش و این فصل را با یک مثال که کمی از مثال‌های قبلی پیچیده‌تر است به پایان می‌بریم. فرض کنید می‌خواهیم مکان‌نما[4] را در محیط متنی کنترل کنیم، طوری که با کلیدهای w و s و a و d به ترتیب به جهت‌های بالا، پایین، چپ و راست برود و هر کجا کاربر کلید space را فشرد، یک * قرار بگیرد. برای این که مکان‌نما حرکت کند و به مختصاتی که ما می‌خواهیم برود، باید از تابع gotoxy(x,y) که در <nodet.h> قرار دارد استفاده کنیم. gotoxy(x,y) مکان‌نما را به مختصات (x,y) در صفحه می‌برد.

با ذکر این توضیحات، نوشتن برنامه چندان دشوار نخواهد بود، برنامه‌ی مربوط به این مثال در کد ۴-۱۰ آمده است.

[4] Cursor

توجه!

مختصات y در کامپیوتر برعکس مختصات دکارتی رو به پایین زیاد می‌شود. سیستم مختصات کامپیوتری از (0,0) آغاز می‌شود که (0,0) گوشه‌ی بالا و چپ صفحه خروجی است. در شکل زیر چند نقطه‌ی مختصاتی برای فهم بهتر نمایش داده شده است:

کد ۴-۱۰

```
#include <iostream>
#include <conio.h>
#include <nodet.h>
using namespace std;
int main()
{
   char ch=0;
   int x=0,y=0;
   while(ch!=27)
   {
       ch=getch();
       switch(chartolower(ch))
       {
           case 'w':
               y--;
           break;
           case 's':
               y++;
           break;
           case 'a':
               x--;
           break;
           case 'd':
               x++;
```

```
25              break;
26              case ' ':
27                  cout<<"*";
28              break;
29          }
30          gotoxy(x,y);
31      }
32      return 0;
33  }
```

چنانچه برنامه را مطابق کد ۴-۱۰ نوشته باشید، متوجه ایرادهایی در آن خواهید شد. اول آن که برخی مواقع که یکی از مؤلفه‌های x یا y صفر می‌شوند و می‌خواهیم آن‌ها را کاهش دهیم، مکان‌نما در محل خود گیر می‌کند (زیرا gotoxy(x,y)، با x یا y منفی فراخوانی شده است) و یا این که برای پیشروی مکان‌نما، هیچ محدودیتی وجود ندارد. برای برطرف کردن این ایرادات، فرض می‌کنیم که می‌خواهیم محدوده‌ی حرکت مکان‌نما، با دو عدد M و N مشخص شود و در ضمن چنانچه مکان‌نما در انتها یا ابتدای x و y قرار گرفت، دور بزند و به ابتدا یا انتهای x و y برود. به این ترتیب، تنها کافیست کد ۴-۱۰ را طوری تکمیل کنیم که پس از کاهش یا افزایش x و y، محدوده‌ی x و y را نسبت به 0 و اعداد از پیش تعیین شده‌ی M و N بسنجد (M و N را نیز دو متغیر int در نظر می‌گیریم). متن کامل شده‌ی این برنامه در کد ۴-۱۱ آمده است.

کد ۴-۱۱

```
1   #include <iostream>
2   #include <conio.h>
3   #include <nodet.h>
4   using namespace std;
5   int main()
6   {
```

```
7     char ch=0;
8     int x=0,y=0;
9     int M=7,N=12;
10    while(ch!=27)
11    {
12        ch=getch();
13        switch(chartolower(ch))
14        {
15            case 'w':
16                y--;
17            break;
18            case 's':
19                y++;
20            break;
21            case 'a':
22                x--;
23            break;
24            case 'd':
25                x++;
26            break;
27            case ' ':
28                cout<<"*";
29            break;
30        }
31        if(x<0)
32            x=M-1;
33         else if(x>=M)
34            x=0;
35        if(y<0)
36            y=N-1;
37         else if(y>=N)
38            y=0;
39        gotoxy(x,y);
40    }
41    return 0;
42 }
```

متن برنامه‌ای که در کد ۴-۱۱ نوشته شد را می‌توان برای اهداف مختلفی مثل ویرایش متن یا انجام بازی‌های مختلف به کاربرد.

فصل پنجم
هجنی سازی

برنامه‌نویسی گرافیکی

۱. ورود به محیط گرافیکی

تا کنون طی چهار فصل گذشته مطالب زیادی آموخته و برنامه‌های زیادی نوشتیم. تا اینجا، هر برنامه‌ای که نوشتیم در محیط متنی اجرا می‌شد، یعنی خروجی به صورت یک پنجره‌ی سیاه بود که یک مکان‌نما در آن وجود داشت و می‌شد مکان‌نما را به مختصات دلخواه برد یا کاراکتر یا رشته‌ای و یا محتوای متغیری را چاپ کرد. اما اگر بخواهیم یک خروجی گرافیکی داشته باشیم چطور؟ یعنی مثلاً بخواهیم یک خط یا یک دایره ترسیم کنیم؟ در این صورت دیگر در محیط متنی نمی‌توانیم این کار را انجام دهیم و احتیاج داریم که ابتدا به محیط گرافیکی وارد شویم، سپس برنامه‌های گرافیکی را بنویسیم. پنجره‌ی محیط گرافیک بسیار شبیه محیط متنی است و مختصات آن از $(0,0)$ و از گوشه‌ی بالا و چپ پنجره شروع می شود. برای آشنایی هرچه بیشتر با این محیط و نحوه‌ی ورود به آن کد ۵-۱ را ببینید.

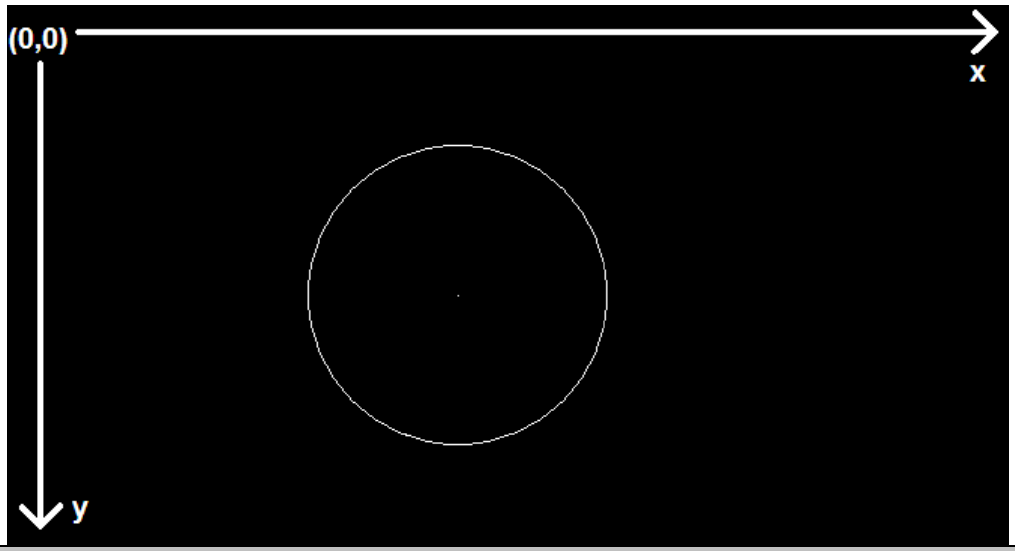
کد ۵-۱

```
1  #include <iostream>
2  #include <graphics.h>
3  using namespace std;
4  int main()
5  {
6     initwindow(800,600);
7     circle(300,300,100);
8     putpixel(300,300,15);
9     getch();
10    closegraph();
11    return 0;
12 }
```

توجه!

برای نوشتن برنامه‌های گرافیکی، علاوه بر استفاده از دستورات <graphics.h>، به جای فایل جدید، باید پروژه‌ی جدید ساخت و پارامترهای آن را به درستی تنظیم کرد. نحوه‌ی انجام این کار در ضمیمه‌ی دوم کتاب آمده است.

در کد ۵-۱، ابتدا در خط ۶، با دستورinitwindow وارد محیط گرافیکی می شویم، یعنی یک پنجره به ابعاد ۸۰۰ پیکسل در ۶۰۰ پیکسل باز می‌شود که می‌توان در آن ترسیمات گرافیکی انجام داد و مبدأ مختصات آن همانند محیط متن، در گوشه‌ی بالا و سمت چپ واقع است و محور y آن رو به پایین زیاد می‌شود (شکل ۵-۱).



شکل ۵-۱: خروجی برنامه نوشته شده در کد ۵-۱ به همراه نحوه قرارگیری محورهای مختصات در صفحه نمایش کامپیوتر در حالت گرافیکی

پس در تابع initwindow، پارامتر اول عرض پنجره و پارامتر دوم ارتفاع پنجره‌ی باز شده برای انجام ترسیمات گرافیکی است. عنوان پنجره نیز windows BGI است که می‌توانید به عنوان پارامتر سوم، عنوان دلخواه خودتان را وارد کنید، مثلاً اگر بنویسید:

```
initwindow (800,600,"ALI")
```

عنوان پنجره باز شده از windows BGI به ALI تغییر می‌یابد و اگر پارامتر سوم را رشته‌ی تهی یا "" وارد کنید، بخش نوار عنوان از پنجره‌ی گرافیک خروجی حذف می‌شود. Initwindow دو پارامتر دیگر دارد، x و y شروع، یعنی برنامه، پنجره‌ی گرافیک را از چه مختصاتی شروع به ترسیم کند؟ مثلاً اگر بنویسید:

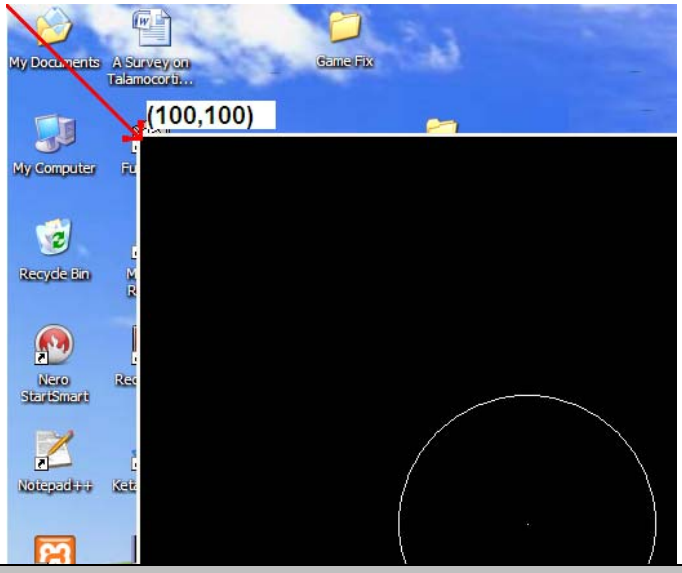
```
initwindow(800,600,"",100,100);
```

پنجره‌ی گرافیک به جای شروع شدن از مختصات (0,0) در صفحه‌ی نمایش (گوشه‌ی بالا سمت چپ)، از مختصات (100,100) در صفحه‌ی اصلی شروع می‌شود (شکل ۵-۲). برای داشتن یک صفحه‌ی گرافیکی به ابعاد صفحه‌ی نمایش، کافی است تا مختصات کل صفحه‌ی نمایش را به دست بیاورید (یا دو عدد بزرگ به جای عرض و ارتفاع صفحه وارد کنید) و سپس بنویسید:

```
initwindow(W,H, "",-3,-3);
```

که در آن W عرض صفحه و H ارتفاع آن هستند (یا دو عدد بزرگ مثل 2000 یا 3000 جای آن‌ها می‌گذاریم). شایان ذکر است که دستور closegraph() که در خط ۱۰ کد ۵-۱ آمده است، برعکس initwindow عمل کرده و منجر به بسته شدن محیط گرافیک می‌شود. حال که با ورود به محیط گرافیکی آشنا شدیم، می‌پردازیم به برخی دستورات پایه و اساسی ترسیم در محیط گرافیک. اصلی‌ترین اجزاء ترسیم، نقطه، خط و دایره هستند. الگوی دستور کشیدن این سه عنصر را در جدول زیر مشاهده می‌کنید:

دستور	توضیحات
`putpixel(x,y,c)`	نقطه‌ای در مختصات (x,y) به رنگ c ترسیم می‌کند.
`line(x1,y1,x2,y2)`	خطی بین نقاط (x1,y1) و (x2,y2) ترسیم می‌کند.
`circle(x,y,r)`	دایره‌ای به مرکز (x,y) و شعاع r ترسیم می‌کند.



شکل ۵-۲: شروع پنجره گرافیک از مختصات (100,100) در صفحه نمایش

در دستور putpixel، رنگ که پارامتر سوم است و در جدول بالا با c مشخص شده، یک عدد است. این عدد در محیط‌های قدیمی برنامه‌نویسی مثل Turbo C++ محدود به فاصله‌ی 0 تا 15 (یعنی شانزده رنگ) و تعدادی رنگ محدود بود، اما در محیط گرافیکی Dev C++، هم می‌توان از رنگ‌های از پیش تعریف شده که با 0 تا 15 مشخص شده‌اند استفاده کرد و هم می‌توان با استفاده از تنظیم میزان نور قرمز و سبز وآبی (R,G,B) یک رنگ دلخواه بر روی صفحه ایجاد کرد. برای این منظور باید از دستور COLOR(R,G,B) استفاده کرد (دقت کنید این تابع برخلاف بقیه‌ی توابع با حروف بزرگ نوشته می‌شود). به این ترتیب اگر بنویسیم

putpixel(100,100,COLOR(255,0,0)) یک نقطه‌ی کاملاً قرمز در مختصات (100,100) ترسیم می‌شود، زیرا پارامتر R تابع COLOR را برابر حداکثر مقدار آن یعنی 255 گذاشتیم، بنابراین این نکته را نیز به خاطر می‌سپارید که حداقل و حداکثر میزان هرکدام از R و G و B به ترتیب 0 و 255 است.

با استفاده از اصل ضرب، می‌توانید بگویید با توجه به محدوده‌ی R و G و B و با استفاده از تابع COLOR، چند رنگ متفاوت می‌توان ایجاد کرد؟

بقیه‌ی رنگ‌ها را نیز می‌توان با استفاده از مخلوطی از نورهای قرمز و سبز و آبی بر روی صفحه ایجاد کرد. اما برای line و circle چطور می‌توان تعیین رنگ نمود؟ برای تعیین رنگ سایر خطوط و منحنی‌هایی که ترسیم می‌شود، باید از دستور setcolor(c); استفاده کرد. به این منظور کافی است قبل از ترسیم، دستور setcolor(c); نوشته شود تا **کلیه‌ی ترسیم‌های بعد از آن** به رنگ c ترسیم شود. برای ترسیم بر روی صفحه می‌توان از کاربر نیز ورودی گرفت. فرض کنید می‌خواهیم مختصات دو سر یک پاره خط را از کاربر گرفته و در متغیرهای x1,y1,x2,y2 ذخیره کنیم و سپس دایره‌ای به شعاع 5 و به مرکز وسط پاره‌خط ترسیم کنیم. ابتدا برنامه‌ی مربوط به این مسأله را در کد ۵-۲ ببینید تا راجع به آن توضیح داده شود:

کد ۵-۲

```
1  #include <iostream>
2  #include <graphics.h>
3  using namespace std;
4  int main()
5  {
6     int x1,y1,x2,y2,xm,ym;
7     cout<<"Enter x1,y1,x2,y2:";
8     cin>>x1>>y1>>x2>>y2;
9     initwindow(1280,800,"",-3,-3);
10    line(x1,y1,x2,y2);
11    xm=(x1+x2)/2;
12    ym=(y1+y2)/2;
13    circle(xm,ym,5);
14    getch();
15    closegraph();
16    return 0;
17 }
```

اولاً در کد ۵-۲ دقت کنید که دستورات cin و cout قبل از initwindow نوشته شده تا از تداخل محیط متن و گرافیک جلوگیری شود و کاربر ابتدا محیط متنی را ببیند. همچنین در خطوط ۱۱ و ۱۲ کد ۵-۲، ممکن است این طور به نظر برسدکه چون x1,x2,y1,y2 همه عدد صحیح هستند، پس جمعشان نیز صحیح است و با تقسیم بر دو (که آن هم عددی صحیح است) دقت از دست می‌رود. یعنی ممکن است حاصل میانگین x2 و x1 برابر 20.5 شود، حال آن‌که چون تقسیم صحیح است، تنها بخش صحیح آن یعنی 20 در xm قرار می‌گیرد. اما باید در نظر داشت که مختصات صفحه‌ی نمایش، تنها اعداد صحیح را می‌پذیرد، یعنی مثلاً پیکسلی به مختصات (20.5,7) نداریم. طول پیکسل یا 20 است و یا 21 و بین این دو بی معنی خواهد بود. بنابراین نوع متغیرها و عملیاتی که انجام دادیم درست است.

در برخی موارد پس از گرفتن ورودی از کاربر و باز شدن پنجره، ممکن است پنجره‌ی گرافیک در پشت پنجره‌ی متنی باز شود. در این صورت باید با کلیک موس آن را انتخاب کنید تا به عنوان پنجره‌ی فعال برنامه باشد. به عنوان مثال در برنامه‌ی اخیر که در کد ۵-۲ نوشته شد، اگر پنجره‌ی گرافیک را به ترتیبی که گفته شد انتخاب نکنید، هرچقدر از روی صفحه کلید، کلید بزنید، از برنامه خارج نخواهید شد، زیرا getch(); بعد از دستور initwindow نوشته شده و در صفحه‌ی گرافیک عمل خواهد کرد.

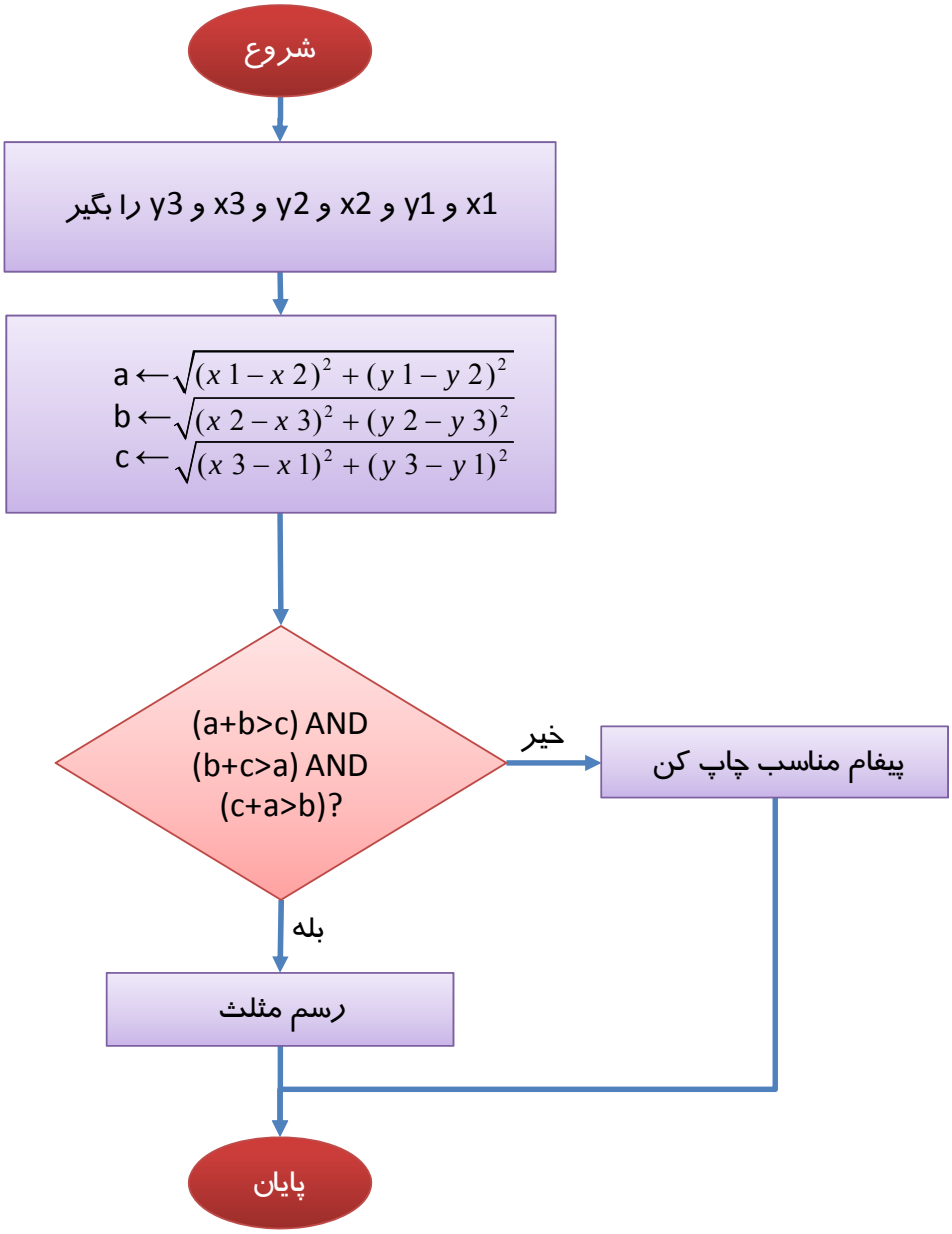
۲. محاسبات در گرافیک: گرفتن سه رأس و آزمودن نامساوی مثلث

در مثال بعدی می‌خواهیم کمی محاسبات نیز به گرافیک اضافه کنیم، به این ترتیب که ۶ عدد x1,y1,x2,y2,x3,y3 را از ورودی گرفته و اگر این سه نقطه (یعنی (x1,y1)، (x2,y2) و (x3,y3)) سه رأس یک مثلث بودند، آن را ترسیم و در غیر اینصورت یک پیغام خطای مناسب چاپ کنیم. گرفتن ورودی همانند قبل و فرایندی ساده و مشخص است. برای این‌که ببینیم آیا با اعداد وارد شده مثلث تشکیل می‌شود یا نه، ابتدا طول اضلاع را محاسبه کرده و سپس تست می‌کنیم که آیا این اضلاع در نامساوی مثلث صدق می‌کنند یا خیر؟ یعنی مجموع طول هر دو ضلع، از ضلع سوم بزرگتر است یا خیر؟ انجام مراحل این کار ساده است، اما برای یادآوری نحوه‌ی استفاده از نمودار گردشی، قبل از حل مسأله و نوشتن برنامه‌ی آن، نمودار گردشی این مسأله را در شکل ۵-۳ مرور کنید. همچنین متن این برنامه در کد ۵-۳ آمده است.

کد ۵-۳

```
1  #include <iostream>
2  #include <graphics.h>
3  #include <conio.h>
4  #include <math.h>
5  using namespace std;
6  int main()
7  {
8     int x1,y1,x2,y2,x3,y3;
9     double a,b,c;
10    cout<<"Enter x1,y1,x2,y2,x3,y3:";
11    cin>>x1>>y1>>x2>>y2>>x3>>y3;
12    a=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
13    b=sqrt((x2-x3)*(x2-x3)+(y2-y3)*(y2-y3));
14    c=sqrt((x3-x1)*(x3-x1)+(y3-y1)*(y3-y1));
15    initwindow(1280,800,"",-3,-3);
16    if((a+b>c)&&(b+c>a)&&(c+a>b))
17    {
18       line(x1,y1,x2,y2);
19       line(x2,y2,x3,y3);
20       line(x3,y3,x1,y1);
21       getch();
22    }
23    else
24    {
25       outtext("The entered points don't form a
   triangle...");
26       getch();
27    }
28    closegraph();
29    return 0;
30 }
```

اما در مورد دستور outtext که در خط ۲۵ آمده، این دستور برای نوشتن متن در محیط گرافیک است. البته می‌توان مختصات گوشه‌ی سمت چپ و بالای متن را نیز به عنوان پارامتر ورودی لحاظ کرده و از تابع outtextxy استفاده کرد که در این صورت باید مختصات نقطه‌ای که می‌خواهیم متن چاپ شود نیز در ابتدا به عنوان دو پارامتر x و y وارد کنیم، مثلاً اگر بخواهیم عبارت Hello در مختصات (320,240) چاپ شود کافی است بنویسیم outtextxy(320,240, "Hello");.



شکل ۵-۳: نمودار گردشی گرفتن مختصات سه نقطه و ترسیم مثلث یا چاپ پیغام مناسب در صورت عدم تشکیل مثلث

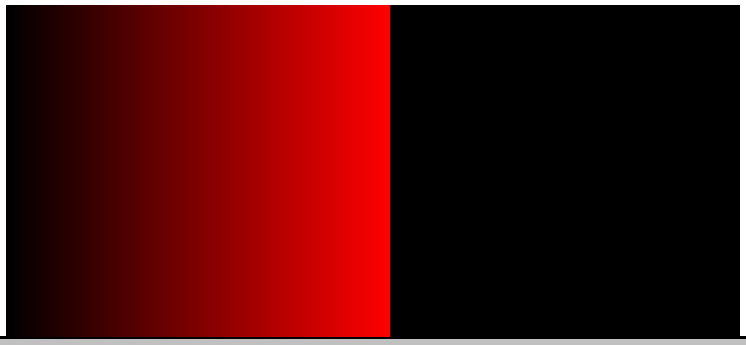
۳. ایجاد طیف رنگ با RGB

در مثال بعدی، می‌خواهیم کمی بیشتر در مورد رنگ‌ها صحبت کنیم. قبلاً گفتیم که تابع COLOR(R,G,B)، سه پارامتر R و G و B که میزان نور قرمز و سبز و آبی سازنده‌ی رنگ نهایی هستند و مقداری بین 0 تا 255 می‌گیرند را گرفته و رنگ مورد نظر را تولید می‌کند. اکنون می‌خواهیم با استفاده از این تابع و ساختار تکرار while که در فصل قبل آموختیم، یک طیف رنگی ایجاد کنیم، یعنی از یک رنگ خاص شروع کرده و به رنگ دیگر برسیم. به عنوان مثال کد ۵-۴ برنامه‌ای را نشان می‌دهد که از رنگ سیاه به رنگ قرمز می‌رسد. برای نمایش هر رنگ، خطی به عرض یک پیکسل و ارتفاع 800 رسم کرده‌ایم.

کد ۵-۴

```
1  #include <iostream>
2  #include <graphics.h>
3  using namespace std;
4  int main()
5  {
6     int i=0;
7     initwindow(1280,800,"",-3,-3);
8     while(i<256)
9     {
10       setcolor(COLOR(i,0,0));
11       line(i,0,i,800);
12       i++;
13    }
14    getch();
15    closegraph();
16    return 0;
17 }
```

حاصل اجرای این کد در شکل ۵-۴ آمده است:



شکل ۵-۴: نمایش طیف رنگی قرمز حاصل اجرای کد ۵-۴

در کد ۵-۴، i متغیری است که هم x خط مورد نظر و هم میزان شدت نور قرمز را تعیین می‌کند.

توجه!

چون صفحه‌ی ایجاد شده در کد مورد نظر ارتفاع 800 داشت، ارتفاع خطوط را نیز 800 گرفتیم، البته نیازی نیست این عدد به صورت دستی وارد شود و می‌توان ارتفاع پنجره‌ی گرافیک را با دستور getmaxy() به دست آورد و به جای عدد 800 در دستور line در خط ۱۱ کد ٥-٤ قرار داد. مشابه این دستور برای x نیز وجود دارد: getmaxx() که عرض صفحه را بر می‌گرداند.

این تنها یک مثال بود که چگونه می‌توان از ترکیب رنگ‌های مختلف در برنامه استفاده کرد.

۴. ترسیم میانه‌های مثلث

مثال بعدی که می‌خواهیم به حل آن بپردازیم، ترسیم میانه‌های مثلث علاوه بر ترسیم خود مثلث است. قبلاً یاد گرفتیم که برای به دست آوردن مختصات وسط یک پاره‌خط، باید

میانگین نقاط ابتدا و انتهای آن را حساب کنیم. به این ترتیب کافی است تا از هر رأس به وسط ضلع مقابل آن یک خط ترسیم کنیم. متن برنامه‌ی انجام این کار در کد ۵-۵ آمده است.

کد ۵-۵

```
1  #include <iostream>
2  #include <graphics.h>
3  #include <conio.h>
4  #include <math.h>
5  using namespace std;
6  int main()
7  {
8     int x1,y1,x2,y2,x3,y3;
9     double a,b,c;
10    cout<<"Enter x1,y1,x2,y2,x3,y3:";
11    cin>>x1>>y1>>x2>>y2>>x3>>y3;
12    a=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
13    b=sqrt((x2-x3)*(x2-x3)+(y2-y3)*(y2-y3));
14    c=sqrt((x3-x1)*(x3-x1)+(y3-y1)*(y3-y1));
15    initwindow(1280,800,"",-3,-3);
16    if((a+b>c)&&(b+c>a)&&(c+a>b))
17    {
18       line(x1,y1,x2,y2);
19       line(x2,y2,x3,y3);
20       line(x3,y3,x1,y1);
21       setcolor(12);
22       line(x1,y1,(x2+x3)/2,(y2+y3)/2);
23       line(x2,y2,(x1+x3)/2,(y1+y3)/2);
24       line(x3,y3,(x2+x1)/2,(y2+y1)/2);
25       getch();
26    }
27    else
28    {
29       outtext("The entered points don't form a
   triangle...");
30       getch();
31    }
32    closegraph();
33    return 0;
```

```
34 }
```

در کد ۵-۵ و در خطوط ۲۲ تا ۲۴، خطوط میانه به صورت مستقیم ترسیم شده‌اند و دیگر در متغیرها ذخیره نشده‌اند، همچنین در خط ۲۱ رنگ ترسیمات عوض شده تا خطوط میانه با رنگی متفاوت بر روی صفحه ترسیم شوند. ترسیم خطوط و دوایر مختلف محاطی و محیطی مثلث، از جمله تمرین‌های بسیار خوب در زمینه‌ی تقویت گرافیک کامپیوتری است و البته نیاز به مهارت‌های ریاضی و هندسه دارد.

۵. استفاده از موس در محیط گرافیکی

اکنون پس از مثال‌های مختلف ترسیمات گرافیکی، می‌پردازیم به یکی از مهم‌ترین امکاناتی که تحت ویندز و محیط Dev C++ در دسترس است: استفاده از موس در محیط گرافیکی. برای اولین مثال استفاده از موس، تنها یک کار ساده انجام می‌دهیم و آن ترسیم دایره‌ای به شعاع 10 پیکسل و به مرکز مختصات فعلی نشان‌گر موس است. برای گرفتن مختصات فعلی موس از دو تابع mousex() و mousey() باید استفاده کنیم. همچنین شرطی که برای پایان یافتن برنامه می‌گذاریم این است که یک کلید از روی صفحه‌کلید (هر کلیدی غیر از Shift, Alt, Ctrl) فشرده شود. برای چک کردن این شرط باید از تابع kbhit() استفاده کنیم، زمانی که کلیدی از روی صفحه‌کلید فشرده شود، تابع kbhit() مقدار درست و اگر کلیدی فشار داده نشده باشد، مقدار نادرست برمی‌گرداند، پس شرط while می‌شود نقیض مقدار kbhit() یعنی

!kbhit() که مفهوم آن همان کلیدی فشار داده نشده است می‌باشد. متن برنامه‌ی گفته شده در کد ۵-۶ قابل مشاهده است.

کد ۵-۶

```
1  #include <iostream>
2  #include <graphics.h>
3  using namespace std;
4  int main()
5  {
6     initwindow(1280,800,"",-3,-3);
7     while(!kbhit())
8     {
9        circle(mousex(),mousey(),10);
10    }
11    closegraph();
12    return 0;
13 }
```

همانطور که از نتیجه‌ی اجرا می‌بینید، دوایر قبلی پاک نمی‌شوند. برای پاک شدن اثر آن‌ها می‌توان ابتدا چک کرد که آیا مختصات موس عوض شده یا نه؟ اگر مختصات عوض شده بود و احتیاج به ترسیم مجدد بود دایره‌ی قبلی را با یک بار ترسیم مجدد با رنگ سیاه (کد 0) پاک کرده و دایره‌ای در مختصات جدید رسم می‌کنیم. برنامه کامل مربوط به این فرایند در کد ۵-۷ آمده است.

کد ۵-۷

```
1  #include <iostream>
2  #include <graphics.h>
3  using namespace std;
4  int main()
5  {
6     int x=-1,y=-1;
7     initwindow(1280,800,"",-3,-3);
8     while(!kbhit())
9     {
10       if((x!=mousex())||(y!=mousey()))
11       {
12          setcolor(0);
13          circle(x,y,10);
14          x=mousex();
15          y=mousey();
16          setcolor(15);
17          circle(x,y,10);
18       }
19    }
20    closegraph();
21    return 0;
22 }
```

نکته‌ی مهم در کد ۵-۷ این است که x و y را تعریف کردیم تا مختصات موس را در آن‌ها نگه داریم، زیرا در غیر این صورت با عوض شدن مختصات موس، اعدادی که mousex() و mousey() بر می‌گردانند عوض شده و قادر نخواهیم بود اثر دایره‌ی قبلی را پاک کنیم.

حال می خواهیم کمی بیشتر از امکانات موس استفاده کنیم و چنانچه دکمه‌ی چپ موس یک بار کلیک شد، در محل موس یک مربع و دو خط بر روی آن ترسیم (شکلی شبیه یک خانه!). برای این‌که بفهمیم کلید سمت چپ موس کلیک شده یا نه می‌توانیم از تابع ismouseclick استفاده کنیم. این تابع یک پارامتر ورودی می‌گیرد که نوع رویداد[1] مربوط به موس را مشخص

[1] Event

می‌کند و در صورت اتفاق افتادن آن رویداد مقدار درست و در غیر این صورت نادرست برمی‌گرداند.

مقدار	مفهوم
WM_MOUSEMOVE	حرکت موس
WM_LBUTTONDBLCLK	دوبار کلیک دکمه چپ موس
WM_LBUTTONDOWN	فشرده شدن دکمه چپ موس
WM_LBUTTONUP	رها شدن دکمه چپ موس
WM_MBUTTONDBLCLK	دوبار کلیک دکمه میانی موس
WM_MBUTTONDOWN	فشرده شدن دکمه میانی موس
WM_MBUTTONUP	رها شدن دکمه میانی موس
WM_RBUTTONDBLCLK	دوبار کلیک دکمه راست موس
WM_RBUTTONDOWN	فشرده شدن دکمه راست موس
WM_RBUTTONUP	رها شدن دکمه راست موس

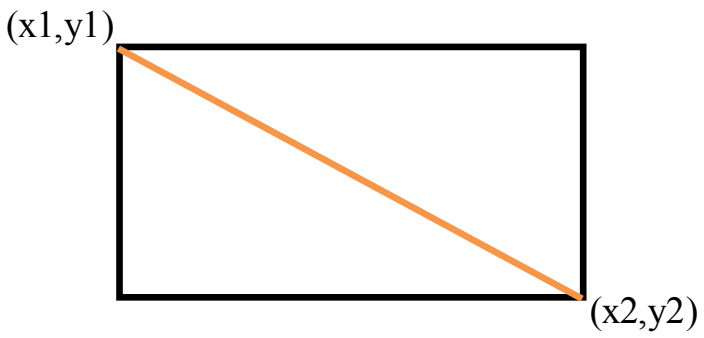
بنابراین با استفاده از تابع ismouseclick می‌توان برنامه‌ی مورد نظر را نوشت که متن آن در کد ۵-۸ آمده است.

کد ۵-۸

```
1  #include <iostream>
2  #include <graphics.h>
3  using namespace std;
4  int main()
5  {
6     int x=-1,y=-1;
7     initwindow(1280,800,"",-3,-3);
8     while(!kbhit())
9     {
10       if(ismouseclick(WM_LBUTTONDOWN))
11       {
```

```
12              x=mousex();
13              y=mousey();
14              rectangle(x,y,x+100,y+100);
15              line(x,y,x+50,y-50);
16              line(x+50,y-50,x+100,y);
17              clearmouseclick(WM_LBUTTONDOWN);
18          }
19      }
20      closegraph();
21      return 0;
22  }
```

در کد ۵-۸ یک دستور ترسیمی جدید دیگر می‌بینید و آن rectangle(x1,y1,x2,y2) است که یک مستطیل با قطر مشخص شده با x1,y1,x2,y2 رسم می‌کند (شکل ۵-۵).



شکل ۵-۵: نحوه عملکرد دستور rectangle(x1,y1,x2,y2) و ترسیم مستطیل در صفحه

همچنین در خط ۱۷ کد ۵-۸ دستور clearmouseclick آمده، زیرا اگر یک بار ismouseclick درست باشد، درست می‌ماند تا زمانی که دستور clearmouseclick مربوط به همان رویداد صدا زده شود. برای فهم بهتر وظیفه‌ی این دستور، می‌توانید خط ۱۷ کد ۵-۸ را حذف کرده و یک بار دیگر برنامه را اجرا کرده و نتیجه را ببینید.

اکنون فرض کنید می‌خواهیم دو مسأله‌ی آخر را با هم ادغام کنیم (یعنی کدهای ۵-۷ و۵-۸)، طوری که هنگام حرکت موس همواره دور آن یک دایره باشد و از طرفی هنگام کلیک دکمه‌ی چپ، یک ترسیم انجام شود. شاید اولین برنامه‌ای که به ذهن می‌رسد شبیه کد ۵-۹ باشد:

کد ۵-۹

```
#include <iostream>
#include <graphics.h>
using namespace std;
int main()
{
   int x=-1,y=-1;
   initwindow(1280,800,"",-3,-3);
   while(!kbhit())
   {
      if((x!=mousex())||(y!=mousey()))
      {
         setcolor(0);
         circle(x,y,10);
         x=mousex();
         y=mousey();
         setcolor(15);
         circle(x,y,10);
      }
      if(ismouseclick(WM_LBUTTONDOWN))
      {
         rectangle(x,y,x+100,y+100);
         line(x,y,x+50,y-50);
         line(x+50,y-50,x+100,y);
         clearmouseclick(WM_LBUTTONDOWN);
      }
   }
   closegraph();
   return 0;
}
```

اگر برنامه‌ی مربوط به کد ۵-۹ را اجرا کرده باشید، دیده‌اید که حرکت موس بر روی ترسیمات، باعث پاک شدن آن‌ها می‌گردد. دلیل این امر نیز بسیار واضح است، در خط ۱۲ و ۱۳ کد ۵-۹، دایره‌ای با رنگ مشکی پس‌زمینه ترسیم می‌شود، حال اگر این ترسیم بر روی قسمت‌هایی که رنگشان سفید است انجام شود، آن ترسیمات پاک می‌شوند.

راه حل استفاده از مُد ترسیمی XOR است. در حالت عادی، هر چه که ترسیم کنیم بر روی پیکسل‌های قبلی نوشته می‌شود، اما می‌توان کاری کرد که هر ترسیمی در هر نقطه‌ای انجام می‌شود با نقاط قبلی خود XOR شود. همچنین اگر یک عدد مثل a دو بار پشت سر هم با یک عدد دیگر مثل b، XOR شود، حاصل خود a خواهد بود. پس کافی است مُد ترسیم را از حالت کپی، به حالت XOR ببریم، دایره را ترسیم کنیم و سپس مجدداً مُد ترسیم را به حالت کپی برگردانیم. هنگام کلیک کردن کاربر نیز ابتدا باید مُد ترسیم را برابر XOR قرار داده و با یک بار ترسیم دایره، آن را از صفحه پاک کنیم، سپس مُد ترسیم را به کپی برگردانیم و ترسیمات را انجام دهیم، سپس مُد ترسیم را دوباره به XOR برگردانده و دایره را مجدداً ترسیم کنیم. برنامه‌ی مربوطه در کد ۵-۱۰ نوشته شده است.

کد ۵-۱۰

```
1  #include <iostream>
2  #include <graphics.h>
3  using namespace std;
4  int main()
5  {
6     int x=-100,y=-100;
7     initwindow(1280,800,"",-3,-3);
8     while(!kbhit())
9     {
10       if((x!=mousex())||(y!=mousey()))
11       {
```

```
12              setwritemode(XOR_PUT);
13              setcolor(15);
14              circle(x,y,10);
15              x=mousex();
16              y=mousey();
17              setcolor(15);
18              circle(x,y,10);
19              setwritemode(COPY_PUT);
20          }
21          if(ismouseclick(WM_LBUTTONDOWN))
22          {
23              setwritemode(XOR_PUT);
24              setcolor(15);
25              circle(x,y,10);
26              setwritemode(COPY_PUT);
27              rectangle(x,y,x+100,y+100);
28              line(x,y,x+50,y-50);
29              line(x+50,y-50,x+100,y);
30              clearmouseclick(WM_LBUTTONDOWN);
31              setwritemode(XOR_PUT);
32              setcolor(15);
33              circle(x,y,10);
34              setwritemode(COPY_PUT);
35          }
36      }
37      closegraph();
38      return 0;
39  }
```

در کد ۵-۱۰، XOR_PUT و COPY_PUT ثوابتی هستند که تعریف شده‌اند و به ترتیب برابر 1 و 0 هستند، یعنی به جای XOR_PUT و COPY_PUT می‌توان 1 و 0 قرار داد و برنامه درست کار خواهد کرد.

۶. استفاده از اعداد تصادفی[۲]

در بسیاری از موارد، احتیاج به این داریم که یک عدد به صورت تصادفی توسط کامپیوتر تولید شود و ما از آن استفاده کنیم. مثلاً فرض کنید می‌خواهیم تعدادی خطوط تصادفی با رنگ‌های تصادفی بر روی صفحه ترسیم کنیم تا زمانی که کاربر کلیدی را بزند. برای تولید اعداد تصادفی، ابتدا باید یک‌بار از دستور srand(time(NULL)); استفاده کنیم تا اعداد تولید شده واقعاً تصادفی باشند. اگر این دستور را نزنیم، اعداد تصادفی به درستی تولید نخواهند شد. اما برای تولید عدد تصادفی باید از دستور rand() استفاده کنیم. این دستور یک عدد صحیح تولید خواهد کرد که بین 0 تا 32767 است. چنانچه بخواهیم عدد تولید شده، بین 0 تا N-1 باشد، باید از rand()%N استفاده کنیم، زیرا باقی‌مانده‌ی هر عدد به N بین 0 تا N-1 خواهد بود. لازم به ذکر است برای کار کردن کلیه‌ی این دستورها، باید <cstdlib> را #include کنید. حال برای ترسیم خطوط تصادفی بر روی صفحه، کافی است مختصات ابتدا و انتها و همچنین رنگ را به صورت تصادفی تولید کنیم. برنامه‌ی مربوط به انجام چنین کاری در کد ۵-۱۱ آمده است.

کد ۵-۱۱

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <graphics.h>
4  using namespace std;
5  int main()
6  {
7     int x1,y1,x2,y2,C;
8     srand(time(NULL));
9     initwindow(1280,800,"",-3,-3);
```

[۲] Random Numbers

```
10      while(!kbhit())
11      {
12          x1=rand()%getmaxx();
13          y1=rand()%getmaxy();
14          x2=rand()%getmaxx();
15          y2=rand()%getmaxy();
16          C=rand()%15+1;
17          setcolor(C);
18          line(x1,y1,x2,y2);
19      }
20      closegraph();
21      return 0;
22  }
```

در کد ۵-۱۱ و در خط ۱۶، c که همان رنگی است که قرار است خط با آن رنگ ترسیم شود، به صورت rand()%15+1 نوشته شده، علت آن است که می‌خواهیم c بین ۱ تا ۱۵ باشد، بنابراین ابتدا با rand()%15 یک عدد تصادفی بین0 تا 14 می‌سازیم، سپس با اضافه کردن 1 به آن، عدد نهایی بین 1 تا 15 خواهد بود.

اگر برنامه‌ی کد ۵-۱۱ را اجرا کنید، دو نکته در آن وجود دارد :

(۱) خطوط، به سرعت ترسیم می‌شوند و صفحه به سرعت پرشده و روند کشیده شدن خط‌ها قابل پیگیری توسط چشم نیست.

(۲) تعداد رنگ‌ها محدود است (۱۵ رنگ) در حالی که می‌توانیم با استفاده از دستور COLOR رنگ‌های تصادفی خیلی بیشتری ایجاد کنیم.

برای حل مشکل اول، از دستور delay استفاده می‌کنیم، این دستور یک تأخیر بر حسب میلی ثانیه ایجاد می‌کند. مثلاً اگر بنویسیم ;delay(10) یک تأخیر ۱۰ میلی ثانیه‌ای یا یک صدم ثانیه‌ای ایجاد می‌شود. به این ترتیب چشم فرصت دنبال کردن تغییرات را خواهد یافت. برای

دومین مشکل نیز، به جای تولید یک عدد تصادفی بین ۱ تا ۱۵ به عنوان شماره‌ی رنگ سه عدد تصادفی بین 0 تا 255 تولید می‌کنیم و به عنوان مؤلفه‌های R و G و B دستور COLOR از آن‌ها استفاده می‌کنیم. به این ترتیب برنامه‌ی اصلاح شده در کد ۵-۱۲ نوشته شده است:

کد ۵-۱۲

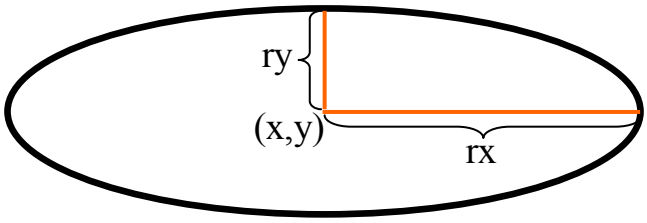
```
1  #include <cstdlib>
2  #include <iostream>
3  #include <graphics.h>
4  using namespace std;
5  int main()
6  {
7     int x1,y1,x2,y2,R,G,B;
8     srand(time(NULL));
9     initwindow(1280,800,"",-3,-3);
10    while(!kbhit())
11    {
12        x1=rand()%getmaxx();
13        y1=rand()%getmaxy();
14        x2=rand()%getmaxx();
15        y2=rand()%getmaxy();
16        R=rand()%256;
17        G=rand()%256;
18        B=rand()%256;
19        setcolor(COLOR(R,G,B));
20        line(x1,y1,x2,y2);
21        delay(40);
22    }
23    closegraph();
24    return 0;
25 }
```

۷. ترسیم اشیاء توپر

تا کنون هر ترسیمی که انجام دادیم توخالی بود! یعنی اگر دایره یا مستطیلی ترسیم کردیم، تنها دور آن شیء رسم شده و داخل آن رنگ نمی‌شد. در این بخش می‌خواهیم اشیاء توپر ترسیم کنیم. در ابتدا دستورهای مربوطه را معرفی می‌کنیم: دو دستور پرکاربرد برای ترسیم اشیاء توپر وجود دارد:

```
bar(x1, y1,x2,y2);
fillellipse(x,y,rx,ry);
```

دستورbar، مثل rectangle عمل می‌کند، با این فرق که مستطیل ترسیم شده توپر خواهد بود. دستور fillellipse نیز یک بیضی توپر به مرکز (x,y) و شعاع افقی rx و شعاع عمودی ry ترسیم می‌کند (شکل ۵-۶).

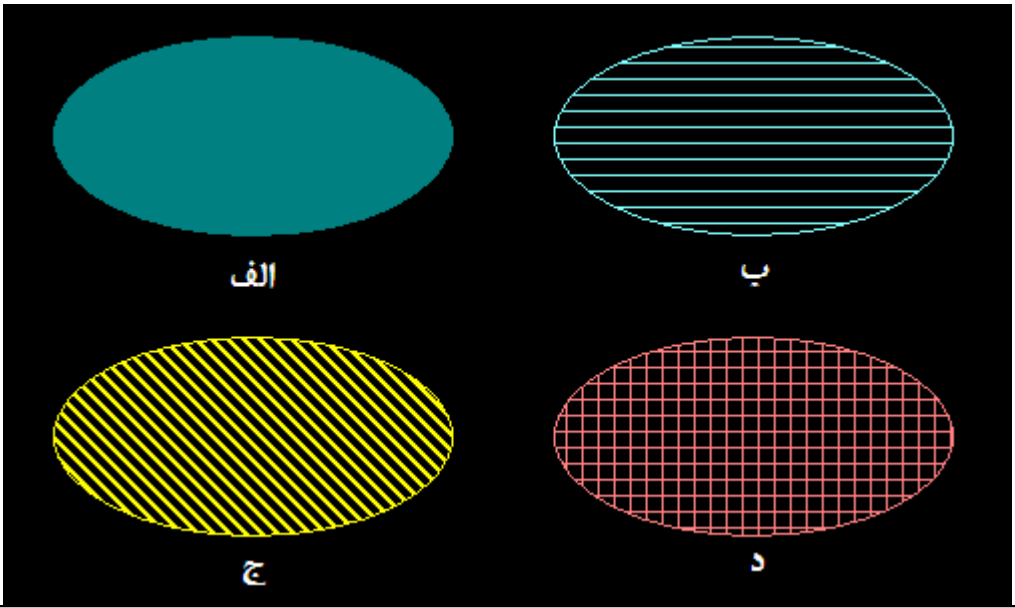


شکل ۵-۶: نحوه ترسیم یک بیضی توپر با دستور fillellipse و مشخص کردن پارامترهای ورودی آن بر روی شکل

برای اشیاء توپر، دو نوع رنگ می‌تواند تعیین شود. یکی رنگ پیرامون آن‌ها که با همان setcolor تعیین می‌شود، دیگری رنگ و مدل پرکردن آن‌ها که توسط تابع setfillstyle تعیین می‌شود. الگوی این تابع چنین است:

```
setfillstyle(PATTERN, COLOR);
```

که PATTERN عددی است که نوع پرکردن را نشان می‌دهد، مثلاً ۱ برای پرکردن ساده، ۲ برای هاشور افقی و... می‌باشند. به عنوان مثال چند ترکیب مختلف از رنگ پیرامون و رنگ پرشده‌ی داخل شئ را به همراه الگوهای مختلف در شکل ۵-۷ ملاحظه می‌کنید.



شکل ۵-۷: انواع مدل‌های پر کردن با دستور setfillstyle که به جای پارامتر اول آن اعداد (الف): ۱ (ب): ۲ (ج): ۵ (د): ۷ قرار گرفته است

حال برای استفاده از ترسیم اشیاء توپر، برنامه‌ی کد ۵-۱۲ را تغییر داده و مستطیل‌هایی توپر به جای خطوط ترسیم می‌کنیم، خواهید دید که با رسم مستطیل توپر، تفاوت رنگ‌های تولید شده بیشتر مشخص خواهد شد. متن این برنامه برای استفاده از اشیاء توپر در کد ۵-۱۳ آمده است.

کد ۵-۱۳

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <graphics.h>
4  using namespace std;
5  int main()
6  {
7     int x1,y1,x2,y2,R,G,B;
```

```
8     srand(time(NULL));
9     initwindow(1280,800,"",-3,-3);
10    while(!kbhit())
11    {
12        x1=rand()%getmaxx();
13        y1=rand()%getmaxy();
14        x2=rand()%getmaxx();
15        y2=rand()%getmaxy();
16        R=rand()%256;
17        G=rand()%256;
18        B=rand()%256;
19        setcolor(COLOR(R,G,B));
20        setfillstyle(1,COLOR(R,G,B));
21        bar(x1,y1,x2,y2);
22        delay(100);
23    }
24    closegraph();
25    return 0;
26 }
```

همچنین می‌توان غیر از رنگ مستطیل‌های توپر، الگوی پر شدن آن‌ها را نیز تصادفی در نظر گرفت که ایجاد این تغییر با توجه به مطالبی که گفته شده نباید کار سختی باشد. آخرین برنامه‌ای که در این فصل با هم مرور می‌کنیم، این است که به جای کشیدن دایره‌ای خالی در مختصات موس، آن را تبدیل به یک دایره‌ی توپر کنیم و با کلیک موس در هر جای صفحه، اثر آن دایره در آن مختصات باقی بماند. برای این کار کافیست تا تغییرات نه چندان زیادی در کد ۵-۱۰ داده شود. متن تغییر یافته‌ی برنامه در کد ۵-۱۴ آمده است.

کد ۵-۱۴

```
1  #include <iostream>
2  #include <graphics.h>
3  using namespace std;
4  int main()
5  {
6     int x=-100,y=-100;
7     initwindow(1280,800,"",-3,-3);
```

```
8       while(!kbhit())
9       {
10          if((x!=mousex())||(y!=mousey()))
11          {
12              setwritemode(XOR_PUT);
13              setcolor(4);
14              setfillstyle(1,4);
15              fillellipse(x,y,10,10);
16              x=mousex();
17              y=mousey();
18              setcolor(4);
19              setfillstyle(1,4);
20              fillellipse(x,y,10,10);
21              setwritemode(COPY_PUT);
22          }
23          if(ismouseclick(WM_LBUTTONDOWN))
24          {
25              setwritemode(XOR_PUT);
26              setcolor(4);
27              setfillstyle(1,4);
28              fillellipse(x,y,10,10);
29              setwritemode(COPY_PUT);
30              setcolor(1);
31              setfillstyle(1,1);
32              fillellipse(x,y,10,10);
33              clearmouseclick(WM_LBUTTONDOWN);
34              setwritemode(XOR_PUT);
35              setcolor(4);
36              setfillstyle(1,4);
37              fillellipse(x,y,10,10);
38              setwritemode(COPY_PUT);
39          }
40      }
41      closegraph();
42      return 0;
43  }
```

در این برنامه، هنگام عبور دایره‌ی قرمز که در کنار اشاره‌گر موس است از روی دایره آبی ترسیم شده، رنگ بنفش به وجود می‌آید که به خاطر XOR شدن رنگ قرمز با آبی است.

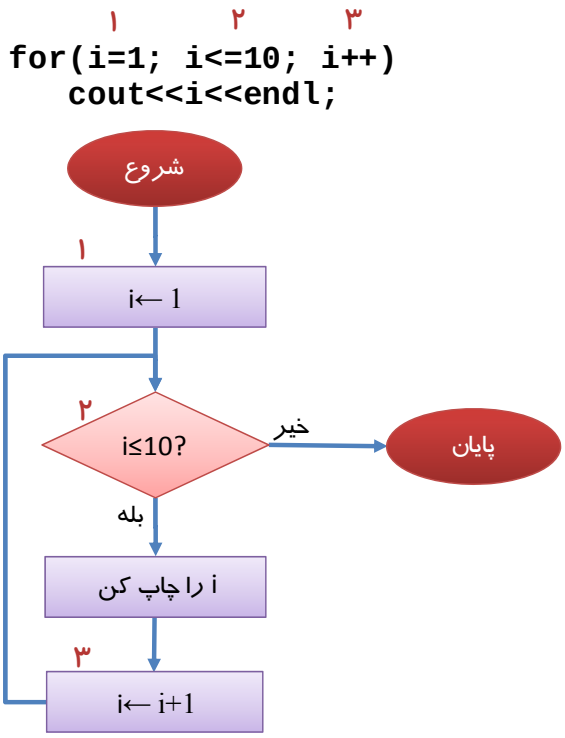
فصل ششم
هشت ها

مباحث تکمیلی ساختارهای تکرار

۱. ساختار تکرار for

در فصل‌های قبل با انواع ساختارهای تصمیم و تکرار آشنا شدیم، در این فصل می‌خواهیم بیشتر در مورد ساختارهای تکرار صحبت کرده و کاربردهای آن‌ها را بررسی کنیم. حلقه‌ی while که در فصل چهارم معرفی کردیم، تا زمانی که شرط آن درست بود اجرا می‌شد. در بسیاری از کاربردها نیاز است تا به مقدار مشخصی یک روند یا یک کار انجام شود. در این مواقع که تعداد دفعات تکرار از قبل مشخص است و نظم خاصی وجود دارد، معمولاً از حلقه‌ی for استفاده می‌شود. حلقه‌ی for سه بخش اساسی دارد:۱) مقداردهی اولیه ۲)شرط حلقه ۳) گام حلقه. قبل از توضیح این سه بخش لازم به ذکر است که حلقه‌ی for احتیاج به یک متغیر حلقه یا همان شمارنده دارد تا تعداد دفعات تکرار حلقه را کنترل کند. مقدار دهی اولیه، به این متغیر مقدار اولیه می‌دهد، مثلاً اگر متغیر حلقه i باشد و بخواهیم i از یک شروع شود می‌نویسیم ;i=1، مقدار دهی اولیه تنها یک بار و در اول حلقه انجام می‌شود. سپس در هر بار اجرای حلقه (از جمله بار اول اجرای حلقه، یعنی بعد از مقدار دهی اولیه) ابتدا درستی شرط حلقه چک می‌شود و اگر شرط حلقه درست بود، حلقه یک بار اجرا می‌شود، سپس، یک بار دستور گام حلقه اجرا شده و مجدداً شرط حلقه چک می‌شود و اگر درست بود دوباره همین فرآیند تکرار می‌شود (یعنی حلقه یک بار اجرا می‌شود و در انتها نیز یک بار گام حلقه اجرا می‌شود) تا شرط حلقه غلط شود. مثلاً گام حلقه می‌تواند این باشد که متغیر حلقه یک واحد افزایش بیابد، یعنی اگر در مثالی که گفتیم متغیر حلقه i باشد، دستور گام حلقه می‌شود ++i، شرط حلقه نیز می‌تواند هر شرطی باشد، مثلاً اگر بخواهیم i از 1 تا 10 برود، باید در مقداردهی اولیه i=1 باشد و شرط نیز i<=10 باشد، یعنی تا زمانی که i از 10 کوچک‌تر یا مساوی است، حلقه اجرا شود. در شکل ۶-۱ شکل کلی حلقه‌ی for برای مثال چاپ اعداد از ۱

تا ۱۰ به همراه نمودار گردشی آن آمده است. همچنین در برنامه‌ی کد ۶-۱ نیز برنامه‌ی مربوط به چاپ اعداد از ۱ تا ۱۰ نوشته شده است.



شکل ۶-۱: نمایش حلقه‌ی for و نمودار گردشی چاپ اعداد ۱ تا ۱۰ (بخش‌های مختلف حلقه for بر روی نمودار گردشی با شماره‌های ۱ تا ۳ مشخص شده‌اند)

کد ۶-۱

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     int i;
7     for(i=1;i<=10;i++)
8        cout<<i<<endl;
9     getch();
10    return 0;
11 }
```

البته حلقه‌ی for را با while نیز می‌توان ساخت، اما چون for ساختار مرتب‌تری برای حلقه‌هایی با تعداد تکرار مشخص دارد، معمولاً از آن برای این‌گونه حلقه‌ها استفاده می‌شود و while بیشتر در مواقعی استفاده می‌شود که پایان حلقه مشخص نیست و مربوط به تحقق شرط خاصی است که ممکن است در طول اجرای برنامه محقق شود یا وابسته به ورودی کاربر است، مثلاً هنگامی که می‌خواهیم تعدادی عدد مشخص چاپ کنیم از for استفاده می‌کنیم اما اگر بخواهیم تا زمانی که کاربر کلیدی را نزده یک ترسیم خاص بر روی صفحه انجام شود، از حلقه‌ی while استفاده می‌کنیم.

همانند if و while، در حلقه‌ی for نیز اگر بخواهیم بیش از یک دستور بنویسیم باید دستورات را در یک بلوک قرار دهیم (یعنی بین { }).

۲. محاسبه‌ی میانگین n عدد

حال فرض کنید می‌خواهیم برنامه‌ای بنویسیم که n را از کاربر گرفته و سپس n عدد از کاربر بپرسد و میانگین آن‌ها را چاپ کند. قبلاً در فصل دوم، نمودار گردشی این مسأله را با هم دیدیم (شکل ۲-۹)، بنابراین پس از آشنایی با ساختار for، نوشتن برنامه نباید سخت باشد. برنامه‌ی حل این مسأله در کد ۶-۲ آمده است.

کد ۶-۲

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     int S=0,a,i,n;
7     cout<<"Enter n:";
8     cin>>n;
9     for(i=1;i<=n;i++)
10    {
11       cout<<"Enter number #"<<i<<":";
12       cin>>a;
13       S=S+a;
14    }
15    cout<<"Average is: "<<double(S)/n;
16    getch();
17    return 0;
18 }
```

در خط ۱۵ کد ۶-۲، متغیر S موقتاً تبدیل به نوع double شده تا حاصل تقسیم S بر n، تبدیل به خارج قسمت نشود (اگر هر دو عملوند عملگر / صحیح باشد، / تبدیل به خارج قسمت می‌شود و قسمت اعشاری حاصل تقسیم، بریده می‌شود).

۳. استفاده از ساختار تصمیم در ساختار تکرار: یافتن بیشینه بین n عدد

حال می‌خواهیم اندکی برنامه را پیچیده‌تر کنیم و از if داخل حلقه‌ی for استفاده کنیم، برنامه‌ی بعد که می‌خواهیم بنویسیم، یافتن بیشینه بین n عدد است. الگوریتم این برنامه را نیز در نمودار گردشی شکل ۲-۱۰ قبلاً دیده‌ایم. برنامه‌ی مربوط به یافتن بیشینه، که ابتدا n را گرفته و سپس n عدد می‌پرسد و از بین آن‌ها بیشینه را پیدا می‌کند، در کد ۶-۳ آمده است. تنها

فرق برنامه‌ی نوشته شده در کد ۶-۳ با الگوریتم نمایش داده شده نمودار گردشی شکل ۲-۱۰ آن است که در انتهای برنامه، مقدار بیشینه چاپ می‌شود.

کد ۶-۳

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     int max=0,a,i,n;
7     cout<<"Enter n:";
8     cin>>n;
9     if(n>=1)
10    {
11        cout<<"Enter number #1:";
12        cin>>a;
13        max=a;
14        for(i=2;i<=n;i++)
15        {
16            cout<<"Enter number #"<<i<<":";
17            cin>>a;
18            if(a>max)
19                max=a;
20       }
21       cout<<"max value of numbers is:"<<max;
22    }
23    getch();
24    return 0;
25 }
```

۴. یک مثال گرافیکی از حلقه‌ی for: رسم n دایره‌ی متحدالمرکز

حال که به این‌جا رسیدیم، می‌پردازیم به حل یک مثال گرافیکی، می‌خواهیم برنامه‌ای بنویسیم که n و r را از کاربر پرسیده و سپس n دایره‌ی متحدالمرکز در وسط صفحه نمایش گرافیکی

ترسیم کند، به گونه‌ای که شعاع دایره‌ی اول r، شعاع دایره‌ی ۲r و ... شعاع دایره‌ی آخر nr باشد. برنامه‌ای که چنین کاری را انجام بدهد در کد ۶-۴ نوشته شده است.

کد ۶-۴

```
#include <iostream>
#include <graphics.h>
using namespace std;
int main()
{
   int x,y,r,n,i;
   cout<<"Enter n:";
   cin>>n;
   cout<<"Enter r:";
   cin>>r;
   initwindow(1280,800,"",-3,-3);
   x=getmaxx()/2;
   y=getmaxy()/2;
   for(i=1;i<=n;i++)
      circle(x,y,i*r);
   getch();
   closegraph();
   return 0;
}
```

۵. چند مثال ریاضیاتی

به این ترتیب با for می‌توان به ترسیم در محیط گرافیکی نیز پرداخت. تا این‌جا باید کاربرد حلقه‌ها یا همان ساختارهای تکرار در زبان C روشن شده باشد. بیشتر مثال‌ها را از مسایلی که در فصول قبل بررسی شده بودند انتخاب کردیم تا نحوه‌ی تبدیل یک نمودار گردشی به برنامه را نیز دیده باشید.

اکنون می‌پردازیم به برخی مثال‌های محاسباتی: می‌خواهیم برنامه‌ای بنویسیم که کلیه‌ی اعداد از یک تا n (n ورودی کاربر است) که مضرب ۳ یا ۵ هستند را با هم جمع کند. به عنوان مثال اگر n برابر ۲۰ وارد شود، باید عدد ۹۸ بر روی صفحه چاپ شود. برای این کار، به استفاده از if در for احتیاج داریم، زیرا باید ببینیم شرط ترکیبی (i%5==0)||(i%3==0) **درست** می‌شود یا خیر؟ اگر درست بود باید i را به مجموع اعداد اضافه کنیم و اگر نبود اضافه نکنیم. i، متغیر حلقه است که از ۱ تا n تغییر خواهد کرد. برنامه‌ی حل این مثال در کد ۶-۵ آمده است.

کد ۶-۵

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     int S=0,i,n;
7     cout<<"Enter n:";
8     cin>>n;
9     for(i=1;i<=n;i++)
10    {
11       if((i%3==0)||(i%5==0))
12          S+=i;
13    }
14    cout<<"S="<<S;
15    getch();
16    return 0;
17 }
```

عملگر % برای دو عملوند صحیح کاربرد دارد و باقی مانده‌ی اولی بـر دومـی را محاسبه می‌کند، یعنی برای بخش‌پذیری a بر b باید a%b برابر با صفر باشد.

یک اشتباه رایج که ممکن است در نوشتن برنامه‌ی مثال گفته شده، یعنی محاسبه‌ی مجموع مضارب ۳ یا ۵ در اعداد از ۱ تا n، استفاده از دو for مجزا و محاسبه‌ی if جداگانه در هر کدام است. اگر چنین اشتباهی بکنیم، مضارب مشترک ۳ و ۵، نظیر ۱۵،۴۵ و... دو بار محاسبه می‌شوند و باید یک بار آن‌ها را از حاصل جمع کل کم کرد که در کل زمان بیشتری می‌گیرد و همان الگوریتم اولیه بهتر است.

۶. محاسبه‌ی سری‌ها و تخمین عدد π

در این بخش می‌پردازیم به یکی از کاربردهای مهم حلقه‌ی for در محاسبات عددی توسط کامپیوتر: محاسبه‌ی سری‌ها. محاسبه‌ی سری‌ها، عبارتست از محاسبه‌ی حاصل جمع جملاتی که از یک نظم ریاضی خاص پیروی می‌کنند. به عنوان مثال، از ریاضیات می‌دانیم که رابطه‌ی زیر برقرار است:

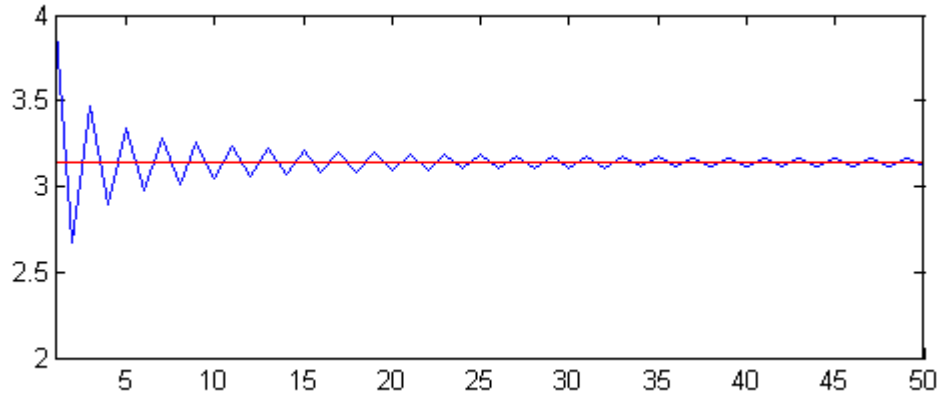
$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$$

یعنی اگر این جملات را که تا بی نهایت ادامه دارند، با هم جمع کنیم، عدد $\frac{\pi}{4}$ حاصل می‌شود که با ضرب آن در 4 عدد π حاصل می‌شود. می‌دانیم جمع بی نهایت جمله ممکن نیست،

بنابراین در عمل معمولاً تا n جمله از این عبارت را جمع می‌کنیم تا تخمینی از عدد π به دست بیاید. همان‌طور که در سری بالا می‌بینید، هرچه در سری پیش‌تر می‌رویم، جملات کوچک‌تر می‌شوند (قدر مطلق آن‌ها $\frac{1}{2i-1}$ است)، پس اگر دقت خیلی زیادی احتیاج نداشته باشیم، با جمع جملات محدودی نیز، می‌توانیم به تخمین خوبی از π دست پیدا کنیم. شکل ۶-۲ نمودار تغییرات این تخمین بر حسب تعداد جملات را تا ۵۰ جمله نشان می‌دهد. خط قرمز عدد π است و نمودار آبی مقدار محاسبه شده با رابطه‌ی گفته شده تا جمله‌ی nاُم است (محور افقی n را نمایش می‌دهد). برنامه‌ی گرفتن n و محاسبه‌ی تخمین π تا n جمله از رابطه‌ی گفته شده در کد ۶-۶ نوشته شده است.

کد ۶-۶

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     int i,n;
7     double S=0,a=1;
8     cout<<"Enter n:";
9     cin>>n;
10    for(i=1;i<=n;i++)
11    {
12       S+=a/(2*i-1);
13       a=-a;
14    }
15    S=4*S;
16    cout<<"Pi="<<S;
17    getch();
18    return 0;
19 }
```



شکل ۶-۲: محاسبه‌ی عدد π تا ۵۰ جمله از روی رابطه‌ی تخمینی گفته شده (به کاهش خطای عدد به دست آمده بر حسب تعداد جملات دقت کنید).

در کد ۶-۶ باید توجه داشته باشید که متغیر S و a باید double باشند، دلیل double بودن متغیر S که مشخص است، زیرا حاصل جمع یکسری اعداد کسری است که قرار است عدد π را تخمین بزند و نمی‌تواند عدد صحیح باشد. متغیر a نیز باید double باشد، زیرا i یک متغیر صحیح است و بنابراین 2*i-1 نیز صحیح است، حال اگر a نیز صحیح باشد، حاصل تقسیم a بر 2*i-1، تبدیل به خارج قسمت می‌شود، یعنی برای i>1، حاصل تقسیم a/(2*i-1)، صفر خواهد شد.

نقش متغیر a نیز این است که به صورت یکی در میان، ۱+ و ۱- شود، زیرا جملات سری تخمین زننده‌ی π، به صورت یکی در میان مثبت و منفی هستند.

۷. مثالی از کاربرد while: تجزیه‌ی ارقام یک عدد صحیح

تا اینجا در مورد حلقه‌ی for بحث زیادی شد و مثال‌های متعددی نیز از کاربرد آن حل شد. اکنون می‌خواهیم یادآور شویم، حلقه‌ی while نیز همواره کارایی خود را خواهد داشت و در مواقع لزوم باید از آن استفاده کرد. مثالی که در این‌جا می‌خواهیم با استفاده از حلقه‌ی while حل کنیم، تجزیه‌ی ارقام یک عدد صحیح است، یعنی یک عدد صحیح مثل ۷۴۱۹ وارد شود و سپس تک تک ارقام آن به صورت مجزا در یک خط چاپ شود. برای این کار، دو راه وجود دارد، راه ساده‌تر آن است که به صورت معکوس تجزیه را انجام دهیم، یعنی در مثال فوق ابتدا ۹، سپس۱، سپس ۴ و نهایتاً ۷ بر روی صفحه چاپ شود. برای انجام تجزیه‌ی عدد به این صورت، کافی است از رقم یکسان شروع کرده و هر بار، یک رقم جدا کنیم. سپس آن رقم را ببریم(با محاسبه‌ی خارج قسمت عدد بر ۱۰) و دوباره از یکان عدد جدید شروع کنیم. به عنوان مثال برای همین عدد ۷۴۱۹، ابتدا رقم یکان که ۹ است محاسبه و چاپ می‌شود (رقم یکان همان باقی‌مانده بر ۱۰ است)، سپس خارج قسمت ۷۴۱۹ بر ۱۰ محاسبه می‌شود، یعنی ۷۴۱، که خود جای عدد قبلی می‌نشیند و به این ترتیب، اگر دوباره رقم یکان را حساب کنیم، این بار عدد ۱ حاصل می‌شود و عدد قبلی تبدیل به ۷۴ می‌شود، به همین ترتیب اعداد ۴ و ۷ نیز بر روی صفحه چاپ می‌شوند و در مرحله‌ی آخر که عدد به ۷ تبدیل شده، با محاسبه‌ی خارج قسمت آن به ۱۰ عدد صفر به دست می‌آید (چون ۷ از ۱۰ کوچک‌تر است) و صفر شدن عدد، نشان‌دهنده‌ی پایان الگوریتم است. برای درک بهتر آن‌چه توضیح داده شد به برنامه‌ی نوشته شده در کد ۶-۷ دقت کنید.

اگر بخواهیم اعداد به صورت معکوس چاپ نشوند چه کار باید کرد؟ ابتدا باید تعداد ارقام عدد را به دست بیاوریم (فرض کنید k) و سپس 10^{k-1} را بسازیم و با استفاده از آن، از رقم آخر شروع به خارج کردن ارقام کنیم. مثلاً برای همان مثال ۷۴۱۹، چنانچه 10^{3} را داشته باشیم، خارج قسمت ۷۴۱۹ بر 10^{3} برابر ۷ است. اکنون برای آن که ۴۱۹ باقی بماند و ۷ از

بین برود، کافیست تا همان عدد ۷ را ضرب در $۱۰^{۳}$ کرده و از ۷۴۱۹ کم کنیم. سپس یکی از توان ۱۰ باید کم شود و همین عمل تکرار شود تا ۹ نیز چاپ شود. برنامه‌ی مربوط به انجام این الگوریتم در کد ۶-۸ نوشته شده است.

کد ۶-۷

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     unsigned n;
7     cout<<"Enter n:";
8     cin>>n;
9     while(n>0)
10    {
11       cout<<n%10<<endl;
12       n=n/10;
13    }
14    getch();
15    return 0;
16 }
```

کد ۶-۸

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     unsigned m,n,d,i=1;
7     cout<<"Enter n:";
8     cin>>n;
9     m=n;
10    while(m>0)
11    {
12       m=m/10;
```

```
13          i*=10;
14      }
15      while(i>1)
16      {
17          i=i/10;
18          d=n/i;
19          cout<<d<<endl;
20          n=n-d*i;
21      }
22      getch();
23      return 0;
24  }
```

در خط ۹ کد ۶-۸، متغیر m برابر n قرار گرفته تا با استفاده از آن تعداد ارقام n مشخص شود، زیرا در while اول، عدد مورد استفاده (در این‌جا m) از بین می‌رود. همچنین متغیر i نیز برای نگه داشتن توان‌های ۱۰ استفاده شده. برای درک بهتر هرچه بهتر این برنامه، جدول متغیرهای while اول و while دوم در زیر آمده است (منظور از i در while دوم، پس از انجام تقسیم در خط ۱۷ کد ۶-۸ است):

while دوم			while اول	
n	i	d	m	i
۷۴۱۹	۱۰۰۰	۷	۷۴۱۹	۱
۷۴۱	۱۰۰	۴	۷۴۱	۱۰
۷۴	۱۰	۱	۷۴	۱۰۰
۷	۱	۹	۷	۱۰۰۰
			۰	۱۰۰۰۰

همان‌طور که دیده می‌شود، برخی اوقات که تعداد دفعات اجرای حلقه مشخص نیست، باید از while استفاده کرد و زمانی که تعداد دفعات اجرا مشخص است باید از for استفاده کرد.

۸. تشخیص اول یا مرکب بودن عدد

مثال دیگری که در این‌جا می‌خواهیم بررسی کنیم، برنامه‌ای است که عدد n را گرفته و بررسی کند که آیا اول است یا مرکب؟ برای تشخیص اول یا مرکب بودن، کافیست ببینیم عدد n، به اعداد ۲ تا $\sqrt{n}$ بخش‌پذیر است یا خیر؟ (چرا؟) اگر چنین بود، یعنی به یکی از اعداد ۲ تا $\sqrt{n}$ بخش‌پذیر بود مرکب است و در غیر این صورت عدد اول است. چون تعداد تکرار حلقه مشخص است، می‌توان از حلقه‌ی for استفاده نمود. برنامه‌ی مربوط به یافتن اول یا مرکب بودن عدد در کد ۶-۹ آمده است.

کد ۶-۹

```
1  #include <iostream>
2  #include <conio.h>
3  #include <math.h>
4  using namespace std;
5  int main()
6  {
7     unsigned n,i,p=1;
8     cout<<"Enter n:";
9     cin>>n;
10    for(i=2;i<=sqrt(n);i++)
11    {
12       if(n%i==0)
13          p=0;
14    }
15    if(p==1)
```

```
16          cout<<n<<" is prime";
17      else
18          cout<<n<<" is not prime";
19      getch();
20      return 0;
21  }
```

۹. حلقه‌های تو در تو

در این بخش می‌پردازیم به استفاده از یک حلقه در حلقه‌ی دیگر، در بسیاری از موارد، نیاز داریم تا از یک حلقه درون حلقه‌ی دیگر استفاده کنیم، به عنوان مثال فرض کنید می‌خواهیم جدول ضرب ۱۰ در ۱۰ را بر روی صفحه چاپ کنیم. برای این کار احتیاج است یک متغیر شمارنده‌ی i داشته باشیم که از ۱ تا ۱۰ را طی کند و یک متغیر شمارنده‌ی دیگر مثل j که به ازای هر مقدار i از ۱ تا ۱۰، خود از ۱ تا ۱۰ مقدار بگیرد تا در کل ۱۰۰ عدد بتوانیم تولید کنیم. برای درک بهتر این موضوع ابتدا به برنامه‌ی چاپ جدول ضرب ۱۰ در ۱۰ در کد ۶-۱۰ دقت کنید:

کد ۶-۱۰

```
1   #include <iostream>
2   #include <conio.h>
3   #include <math.h>
4   using namespace std;
5   int main()
6   {
7       unsigned i,j;
8       for(i=1;i<=10;i++)
9       {
10          for(j=1;j<=10;j++)
11              cout<<i*j<<"\t";
12          cout<<endl;
13      }
```

```
14      getch();
15      return 0;
16  }
```

لازم به ذکر است "\t" که در خط ۱۱ کد ۶-۱۰ آمده است، یک کاراکتر TAB چاپ می‌کند تا برای هر عدد چاپ شده‌ای، تا ۸ خانه رزرو شود و به این ترتیب کل جدول ضرب مرتب چاپ شود (شکل ۶-۳).

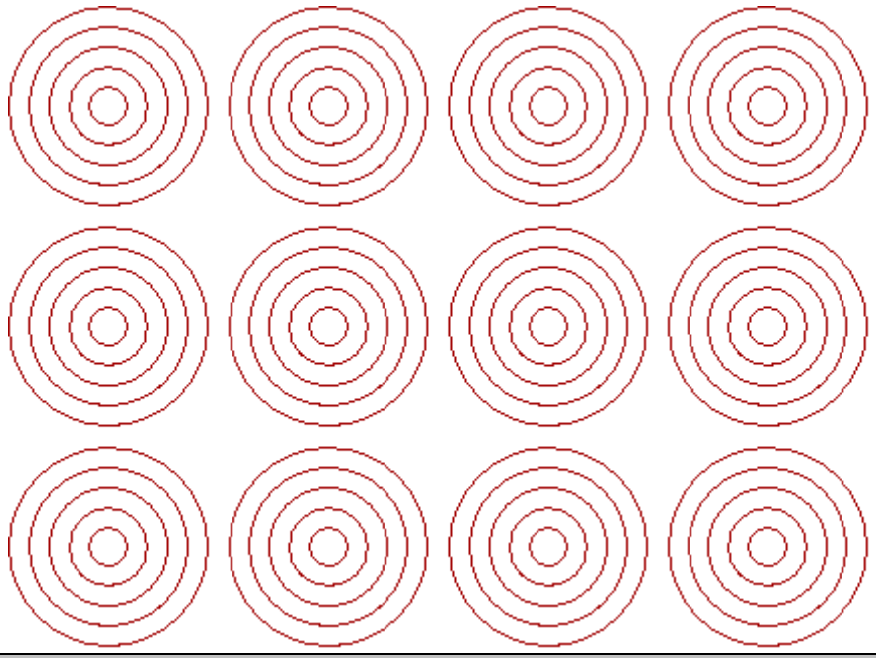
```
1   2   3   4   5   6   7   8   9   10
2   4   6   8   10  12  14  16  18  20
3   6   9   12  15  18  21  24  27  30
4   8   12  16  20  24  28  32  36  40
5   10  15  20  25  30  35  40  45  50
6   12  18  24  30  36  42  48  54  60
7   14  21  28  35  42  49  56  63  70
8   16  24  32  40  48 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

شکل ۶-۳: چاپ جدول ضرب به صورت مرتب و با فواصل ۸ تایی با استفاده از TAB

این مثال تنها یکی از ساده‌ترین مثال‌هایی است که می‌توان برای حلقه‌های تودرتو ذکر کرد. این مبحث بسیار پرکاربرد است و در حل مسایل بسیاری، نیاز داریم تا حلقه‌ها را با یکدیگر ترکیب کنیم. به عنوان مثال فرض کنید می‌خواهیم اعداد ورودی r و k و n و m را از کاربر گرفته و یک جدول m×n از k دایره‌ی متحدالمرکز را ترسیم کنیم که شعاع دایره‌ی اول r،

دایره‌ی دوم ۲r و... شعاع دایره‌ای kاُم kr است.به عنوان مثال در شکل ۶-۴ یک جدول ۳ در ۴ از ۵ دایره‌ی متحدالمرکز به شعاع اولیه‌ی ۱۰ پیکسل (یعنی r=10) را مشاهده می‌کنید.

برای ترسیم این شکل، باید در نظر داشت که اولاً باید دو حلقه‌ی for تو در تو داشته باشیم تا بتوانیم یک جدول m×n تشکیل بدهیم، ثانیاً در هر مرحله یک حلقه‌ی for دیگر برای ترسیم k دایره مورد نیاز است.

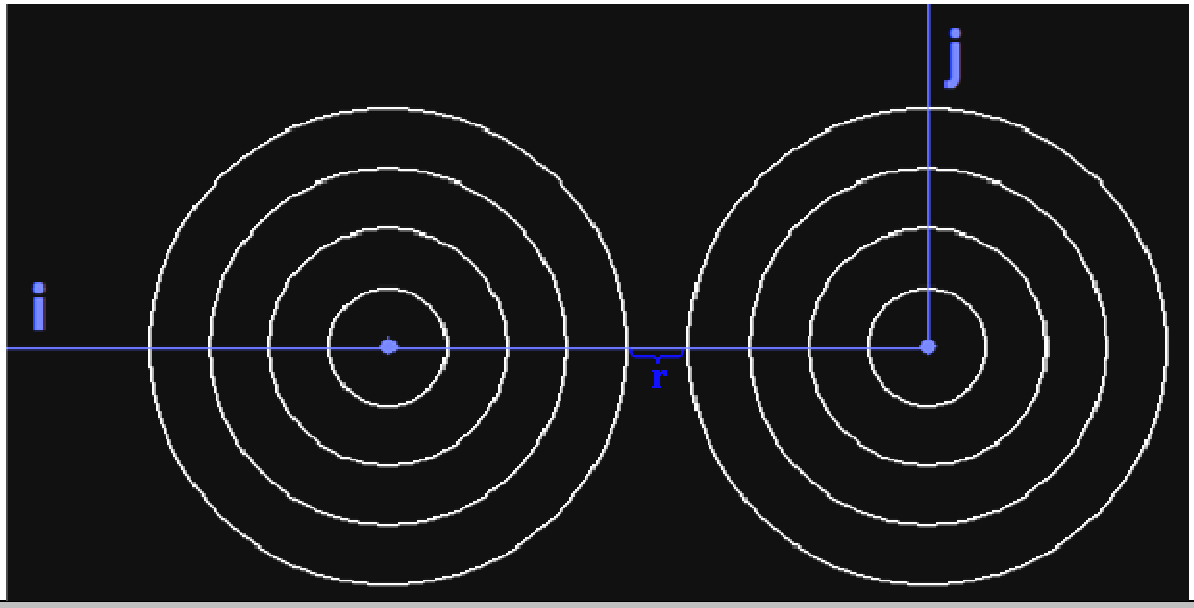


شکل ۶-۴: یک جدول ۳ در ۴ از ۵ دایره‌ی متحدالمرکز به شعاع اولیه‌ی ۱۰ پیکسل

مهم‌ترین چیزی که در این مرحله باید مشخص شود، مرکز دوایر متحدالمرکز است، فرض کنید می‌خواهیم بین هر مجموعه از این دوایر، به اندازه‌ی شعاع پایه‌ی دایره یعنی r نیز فاصله باشد (شکل ۶-۵)، به این ترتیب با اندکی محاسبات متوجه می‌شویم که مرکز دوایری که در سطر i اُم و ستون j اُم قرار دارند (با فرض این‌که 1≤i≤n , 1≤j≤n) از رابطه‌ی:

$$((2\times j-1)\times k\times r+(j-1)\times r\ ,\ (2\times i-1)\times k\times r+(i-1)\times r)$$

به دست می‌آید (کمی به شکل ۶-۵ دقت کنید). بنابراین، برنامه‌ی مورد نظر مطابق کد ۶-۱۱ خواهد بود.



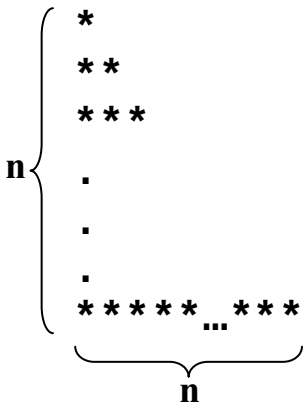
شکل ۶-۵: بررسی نحوه‌ی به دست آوردن مرکز دوایر متحدالمرکز واقع در سطر i و ستون j

کد ۶-۱۱

```
1   #include <iostream>
2   #include <graphics.h>
3   using namespace std;
4   int main()
5   {
6      int x,y,r,m,n,k,i,j,t;
7      cout<<"Enter m,n,k:";
8      cin>>m>>n>>k;
9      cout<<"Enter r:";
10     cin>>r;
11     initwindow(1280,800,"",-3,-3);
12     for(i=1;i<=m;i++)
13        for(j=1;j<=n;j++)
14        {
15           x=(2*j-1)*k*r+(j-1)*r;
```

```
16              y=(2*i-1)*k*r+(i-1)*r;
17              for(t=1;t<=k;t++)
18                  circle(x,y,t*r);
19          }
20      getch();
21      closegraph();
22      return 0;
23  }
```

حال فرض کنید می‌خواهیم با گرفتن n شکل زیر را چاپ کنیم:



شکل ۶-۶: شکلی که در محیط متنی ترسیم شده و حاوی n سطر است و در سطر اول ۱ ستاره، در سطر دوم ۲ ستاره و ... در سطر nاُم n ستاره دارد

باید حواسمان باشد که این شکل را می‌خواهیم در محیط متنی چاپ کنیم نه در محیط گرافیک. نوشتن این برنامه پس از دیدن برنامه‌های مختلف حلقه‌های تودرتو، نباید کار سختی باشد. برنامه‌ی تولید این شکل در کد ۶-۱۲ آمده است.

کد ۶-۱۲

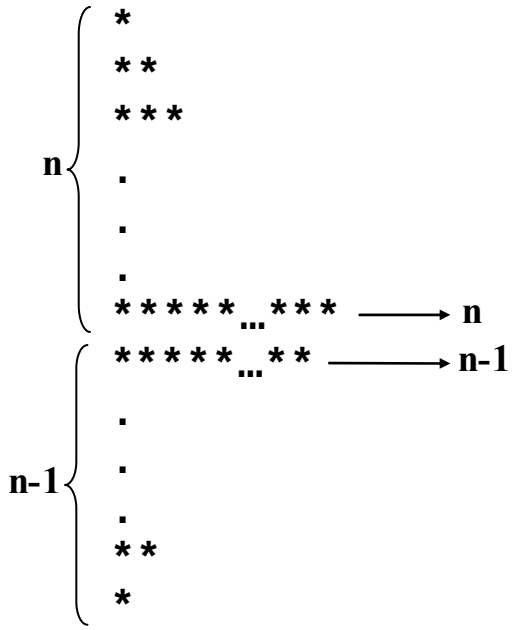
```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     unsigned n,i,j,p;
7     cout<<"Enter n:";
8     cin>>n;
9     for(i=1;i<=n;i++)
10    {
11       for(j=1;j<=i;j++)
12          cout<<"*";
13       cout<<endl;
14    }
15    getch();
16    return 0;
17 }
```

حال اگر برنامه را کمی سخت‌تر کرده و بخواهیم شکل ۶-۷ را ترسیم کنیم، احتیاج به یک حلقه‌ی تودرتوی دیگر در کنار حلقه‌ی قبلی داریم. البته اگر برنامه‌ی کد ۶-۱۲ را خوب فهمیده باشیم، نوشتن برنامه‌ی کد ۶-۱۳ از روی آن سخت نخواهد بود.

کد ۶-۱۳

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     unsigned n,i,j,p;
7     cout<<"Enter n:";
8     cin>>n;
9     for(i=1;i<=n;i++)
10    {
11       for(j=1;j<=i;j++)
12          cout<<"*";
```

```
13          cout<<endl;
14      }
15      for(i=n-1;i>=1;i--)
16      {
17          for(j=1;j<=i;j++)
18              cout<<"*";
19          cout<<endl;
20      }
21      getch();
22      return 0;
23  }
```



شکل ۶-۷: شکلی که در محیط متنی ترسیم شده و حاوی ۲n-۱ سطر است و در سطر اول ۱ ستاره، در سطر دوم ۲ ستاره و ... در سطر nاُم n ستاره دارد و در سطر n+۱اُم n-۱ ستاره و ... در سطر ۲n-۲ دو ستاره و در سطر ۲n-۱ یک ستاره دارد

مثال آخر این فصل، در مورد چاپ کلیه‌ی اعداد اول از ۲ تا n است (n ورودی کاربر است). قبلاً برنامه‌ی تشخیص اول یا مرکب بودن اعداد را نوشته‌ایم، حال کافیست تا آن را در یک

حلقه‌ی بیرونی دیگر قرار دهیم تا کلیه‌ی اعداد از ۲ تا n را چک کند. این برنامه در کد ۶-۱۴ نوشته شده است.

کد ۶-۱۴

```
1  #include <iostream>
2  #include <conio.h>
3  #include <math.h>
4  using namespace std;
5  int main()
6  {
7     unsigned n,i,j,p;
8     cout<<"Enter n:";
9     cin>>n;
10    for(i=2;i<=n;i++)
11    {
12        p=1;
13        for(j=2;j<=sqrt(i);j++)
14           if(i%j==0)
15               p=0;
16        if(p==1)
17            cout<<i<<" is prime"<<endl;
18    }
19    getch();
20    return 0;
21 }
```

دقت کنید، در این برنامه و در خط ۱۲، هر بار p برابر ۱ شده است، زیرا ممکن است به ازای یک i، p برابر 0 قرار گیرد، بنابراین برای i بعدی، مجدداً باید از p برابر ۱ آغاز کنیم.

به این ترتیب این فصل که یکی از مهم‌ترین فصول کتاب است به پایان می‌رسد، توصیه می‌شود این فصل را با دقت خوانده و مثال‌های آن را کاملاً بفهمید و سپس تمرین‌ها را نیز به دقت حل کنید. حلقه‌های تو در تو مبنای بسیاری از الگوریتم‌هاست و اهمیت زیادی در برنامه‌های مختلف دارد.

۷

فصل هفتم
هندسه یاضی

آرایه‌ها

۱. متغیرهای اندیس دار

در فصل‌های قبل، در هر برنامه به تعدادی که نیاز داشتیم متغیر تعریف می‌کردیم، اما این مسأله را مطرح نکردیم که اگر خواستیم تعداد زیادی متغیر داشته باشیم (مثلاً ۱۰ تا یا ۲۰ تا یا حتی بیشتر!) چطور باید آن‌ها را تعریف کنیم؟ در ریاضیات، برای تعریف متغیرهای زیاد معمولاً از اندیس[1] یا همان زیر نویس استفاده می‌شود، مثلاً برای تعریف ۱۰۰ متغیر، می‌نویسیم a_1 تا a_{100} که نمایانگر ۱۰۰ متغیر است و برای اعداد مابین a_1 تا a_{100} نیز از اندیس‌های متناظر استفاده می‌کنیم، مثلاً a_{12}، a_5، a_3 و ... و یا برای نشان دادن i اُمین متغیر، از a_i استفاده می‌کنیم. اما در کامپیوتر چطور؟ اگر بخواهیم ۱۰۰ متغیر تعریف کنیم باید تک تک آن‌ها را تعریف کرده و استفاده کنیم؟ مثلاً اگر بخواهیم ۱۰۰ عدد از کاربر بگیریم و آن‌ها را نگه داریم، یکی یکی باید آن‌ها را cin کنیم!؟ قطعاً این راه حل معقول به نظر نمی‌رسد در ضمن اگر بنویسیم:

```
for(i=1; i<=100; i++)
  cin>>xi;
```

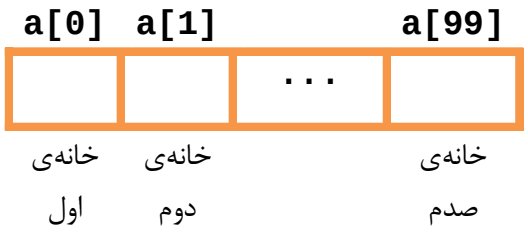
مترجم برنامه‌ی C مانند ریاضی xi را i اُمین x فرض نخواهد کرد، بلکه xi را یک متغیر جدید می‌پندارد که از به هم پیوستن حروف x و i پدید آمده است! و هر بار مقدار i را در کنار x جایگزین نمی‌کند تا x1، x2 و... حاصل شود. پس راه حل چیست؟

راه حل استفاده از آرایه[2]هاست. آرایه‌ها همان متغیرهای مرتب، پشت سرهم و اندیس داری هستند که انتظار داریم باشند. به عنوان مثال اگر بنویسیم:

```
int a[100];
```

[1] Index
[2] Array

یعنی متغیری به نام a و از جنس int تعریف کن و سپس به a اجازه بده تا با علامت [ ] بتوان به اندیس‌های مختلف a دست یافت، مثلاً با نوشتن cout<<a[5]، a[5] چاپ شود. یک نکته‌ی مهم در مورد آرایه‌ها در زبان C این است که آرایه‌ها از اندیس صفر شروع می‌شوند، یعنی به طور مثال در خط برنامه‌ی قبلی، خانه‌ی اول a، a[0] است نه a[1] و به این ترتیب آخرین خانه نیز a[99] است نه a[100] (شکل ۷-۱).



شکل ۷-۱: نحوه‌ی آرایش خانه‌های آرایه و اندیس هر خانه به ترتیب

۲. ترسیم یک n-ضلعی

اکنون به عنوان اولین برنامه، فرض کنید کاربر ابتدا عدد n و سپس مختصات y و x مربوط به n نقطه را وارد می‌کند که این n نقطه، رئوس یک n-ضلعی هستند و ما می‌خواهیم این n-ضلعی را ترسیم کنیم. برای نوشتن برنامه باید آرایه‌ای از اعداد x و آرایه‌ای از اعداد y تعریف کنیم تا به ترتیب طول و عرض نقاط را نگه دارند، سپس یکی یکی آن‌ها را از کاربر گرفته و سپس در محیط گرافیکی به ترسیم n-ضلعی بپردازیم. البته باید حواسمان را به چند نکته جلب کنیم، یکی این که اندیس آرایه را از صفر شروع کنیم، دوم این که هنگام ترسیم n-ضلعی، در حلقه‌ی for، باید رأس i اُم را به رأس i+1 اُم متصل کنیم، پس باید در حلقه‌ی for، i حداکثر

n-2 باشد تا i+1، n-1 باشد تا از حدود آرایه تجاوز نکنیم (فراموش نکنید که خانه‌ی n اُم آرایه در اندیس n-1 است!)

بیرون زدن از حدود آرایه در زبان C تولید هیچ خطایی نمی‌کند که بسیار خطرناک است! یعنی اگر آرایه‌ای به طول ۱۰۰ تعریف کنید و در خانه‌ی ۱۰۷ آن چیزی بنویسید، سیستم عامل جلوی این کار را نمی‌گیرد ولی شما قطعاً در جایی که مجاز به آن نبوده‌اید، چیزی نوشته‌اید یا از مقداری استفاده کرده‌اید که درون آرایه قرار نداشته و متعلق به متغیر دیگری است. این عمل، ممکن است به بروز خطاهای بسیار بدی در برنامه منجر شود که فهمیدن منشأ آن پس از نوشتن برنامه به هیچ وجه کار آسانی نیست. پس سعی کنید از همان ابتدا که با آرایه‌ها برنامه‌نویسی می‌کنید، عادت کنید حدود آرایه را چک کنید تا از محدوده‌ی مجاز تجاوز نکنید.

همچنین در حلقه‌ی for، رأس 0 به 1، 1 به 2، 2 به 3 و... n-2 به n-1 وصل می‌شود، پس در حلقه‌ی for، رأس n-1 به 0 (آخر به اول) وصل نمی‌شود که باید جداگانه و پس از پایان حلقه‌ی for این دو رأس را نیز به هم وصل کنیم. پس از ذکر این توضیحات، بد نیست نگاهی به کد ۷-۱ بیندازید!

کد ۷-۱

```
1  #include <iostream>
2  #include <graphics.h>
3  using namespace std;
4  int main()
5  {
6     int x[100],y[100],n,i;
7     cout<<"Enter n:";
8     cin>>n;
9     for(i=0;i<n;i++)
10    {
11       cout<<"Enter x["<<i+1<<"]:";
12       cin>>x[i];
13       cout<<"Enter y["<<i+1<<"]:";
14       cin>>y[i];
15    }
16    initwindow(1280,800,"",-3,-3);
17    for(i=0;i<n-1;i++)
18        line(x[i],y[i],x[i+1],y[i+1]);
19    line(x[0],y[0],x[n-1],y[n-1]);
20    getch();
21    closegraph();
22    return 0;
23 }
```

در برنامه‌ی نوشته شده در کد ۷-۱ و در خط ۶، آرایه‌های x و y به طول 100 تعریف شده‌اند. دلیل این امر آن است که در زبان C، باید سایز آرایه‌ها از ابتدا مشخص باشد تا سیستم عامل هنگام اجرای برنامه، حافظه‌ی مورد نظر را به برنامه اختصاص دهد. بنابراین معمولاً در هنگام کار با آرایه‌ها، به مقداری که حدس می‌زنیم کافی است، طول آرایه را تعریف می‌کنیم. مثلاً در این جا به احتمال بسیار زیاد کسی بیش از صد نقطه به عنوان ورودی به برنامه نخواهد داد! بنابراین طول آرایه‌های تعریف شده به احتمال بسیار زیاد کافیست. ممکن است این سؤال پیش بیاید که چرا ابتدا n را از کاربر نمی‌گیریم تا به همان اندازه خانه‌های

آرایه را تعریف کنیم؟ پاسخ این است که اگر چنین چیزی در زبان C بنویسید، مثلاً کدی شبیه زیر:

```
int n;
cin>>n;
int x[n], y[n];
```

مترجم زبان C به برنامه‌ی شما ایراد گرفته و آن را اجرا نمی‌کند! زیرا همان‌طور که قبلاً هم توضیح داده شد، قبل از اجرای برنامه، طول آرایه باید مشخص باشد تا حافظه‌ی مورد نیاز از سیستم عامل اخذ شود، بنابراین نمی‌توان به طریقی که در بالا نوشته شد، طول آرایه را به صورت پویا[3] و در زمان اجرا مشخص کرد.

اخذ حافظه‌ی پویا و آرایه با طول دلخواه در زمان اجرا امکان‌پذیر است، اما نه به صورتی که تاکنون گفته شد، بلکه برای این کار باید از اشاره‌گرها استفاده کرد که مطالب مربوط به آن در فصل آخر آمده است.

۳. مرتب‌سازی حبابی[4]

یکی از مسایل مهم که معمولاً هنگام مواجهه با آرایه‌ای از اعداد مطرح می‌شود، مرتب‌سازی آن‌هاست. یعنی به عنوان مثال اگر آرایه‌ی a به صورت زیر باشد:

[3] Dynamic
[4] Bubble Sort

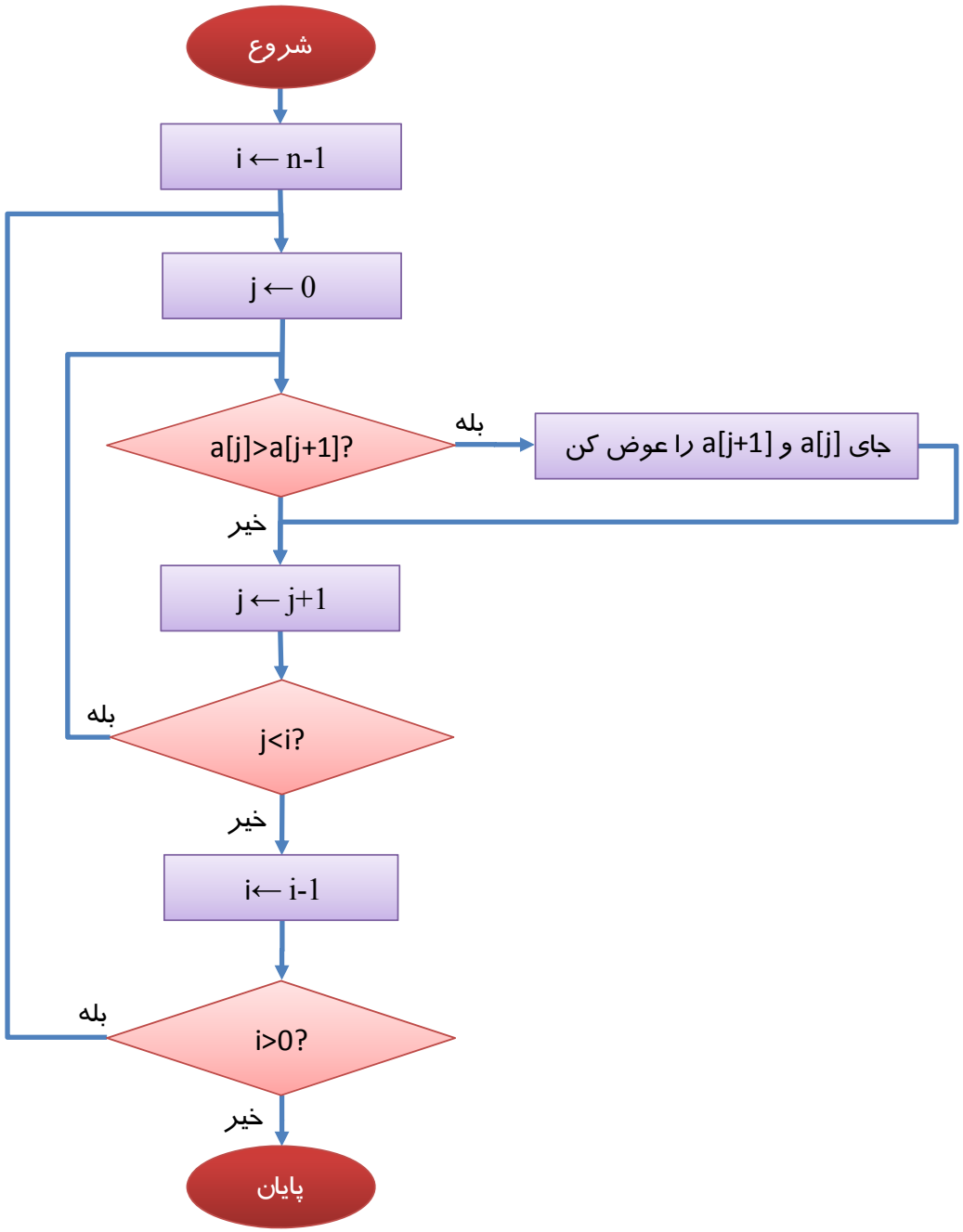
35	17	5	18	76	12

پس از مرتب‌سازی به صورت صعودی به صورت زیر در خواهد آمد:

5	12	17	18	35	76

مرتب‌سازی کاربردهای بسیار وسیعی در حوزه‌ی برنامه‌نویسی دارد و روش‌های بسیار متنوعی برای آن ارائه شده است. یکی از روش‌های ارائه شده برای مرتب کردن اعداد، روش مرتب سازی حبابی است. نمودار گردشی این الگوریتم مرتب‌سازی در شکل ۷-۲ آمده است.

همان‌طور که احتمالاً با دیدن این نمودار گردشی فهمیده‌اید، الگوریتم مرتب‌سازی حبابی از یک حلقه‌ی تو در تو تشکیل شده. حلقه‌ی بیرونی مربوط به متغیر i است و از n-1 شروع می‌شود و هر بار یکی کم می‌شود تا به صفر برسد. حلقه‌ی داخلی نیز با متغیر j کنترل می‌شود که j نیز هر بار از 0 تا i-1 می‌رود، سپس در هر بار گردش حلقه‌ی مربوط به متغیر j، چک می‌شود که آیا a[j] از a[j+1] بزرگ‌تر هست یا خیر؟ چنانچه a[j] از عدد بعدی خود یعنی a[j+1] بزرگ‌تر بود، جای آن‌ها عوض می‌شود. به این ترتیب، چون i اول الگوریتم برابر n-1 است، در اولین اجرا و به ازای i=n-1، بزرگ‌ترین عدد آرایه در خانه‌ی آخر یعنی a[n-1] قرار می‌گیرد. سپس i یکی کم می‌شود (طول حداکثر آرایه یکی کم می‌شود) و همان الگوریتم قبلی تکرار می‌شود، منتها این بار، بزرگ‌ترین عدد بین n-1 عدد باقی مانده به خانه‌ی n-1 اُم، یعنی a[n-2] منتقل می‌شود.

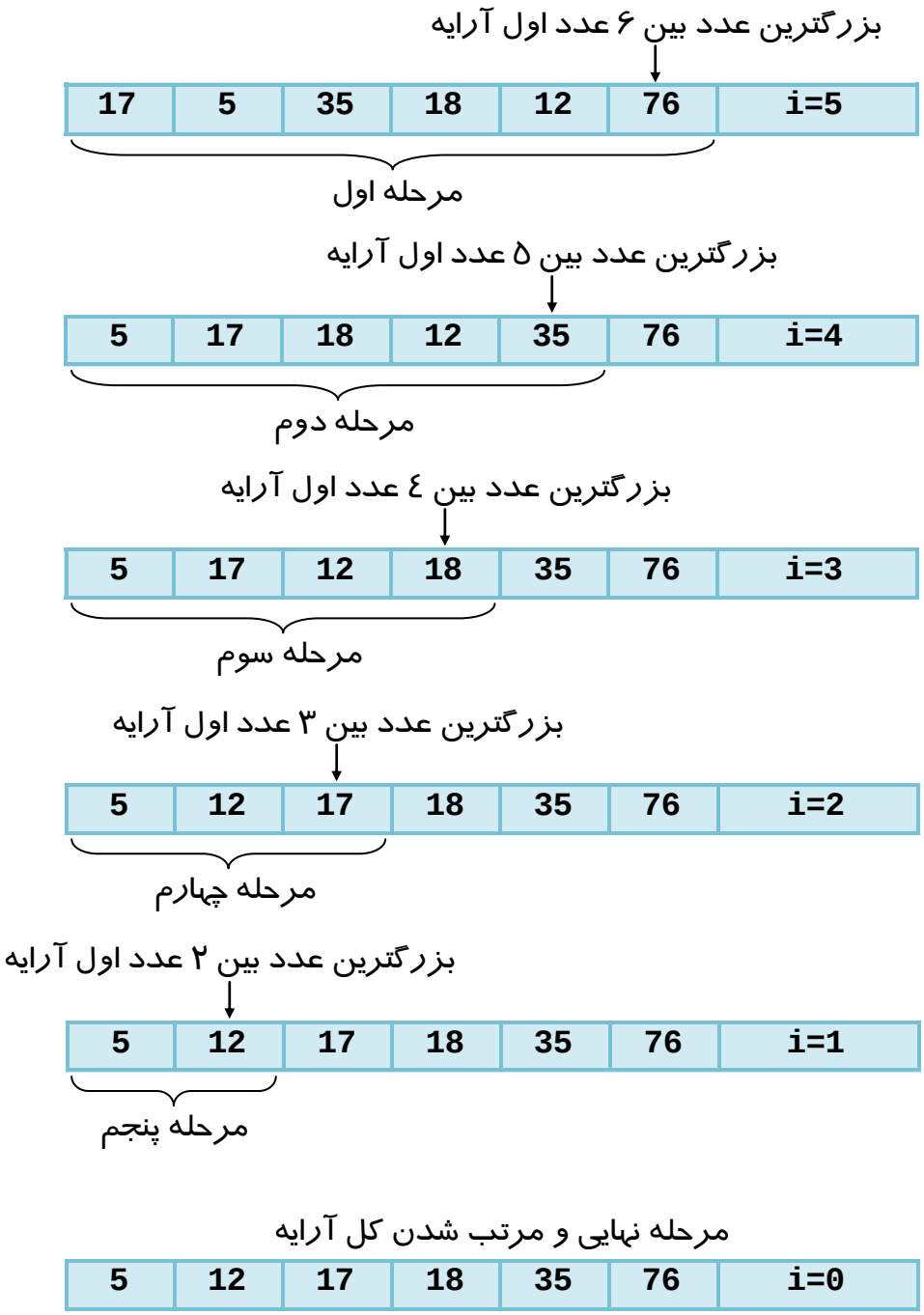


شکل ۷-۲: نمودار گردشی مرتب سازی حبابی

به این ترتیب اگر i به صفر برسد، چون هر بار بزرگ‌ترین مقدار موجود، به انتهای آرایه رسیده، در انتها کل آرایه‌ی a به صورت صعودی مرتب خواهد بود. شکل ۷-۳ نحوه‌ی مرتب

شدن آرایه و رفتن بزرگ‌ترین عدد به انتهای آرایه را در اولین اجرای حلقه‌ی i نشان می‌دهد. شکل ۷-۴ نیز نشان می‌دهد چطور در هر مرحله بزرگ‌ترین عدد به انتهای آرایه‌ی به طول i+1 می‌رسد.

شکل ۷-۳: نحوه‌ی مرتب شدن آرایه و رفتن بزرگ‌ترین عدد به انتهای آرایه در اولین اجرای حلقه‌ی i



شکل ۷-۴: نحوه‌ی رسیدن بزرگ‌ترین عدد به انتهای آرایه‌ی به طول i+1 در هر مرحله

البته دقت می‌کنید در شکل ۷-۴، نتیجه‌ی مرحله‌ی چهارم و پنجم با هم فرقی ندارند که البته لزوماً همواره درست نیست و برای درست بودن الگوریتم، همواره باید تا آخر، آن را اجرا نمود.

برنامه‌ی گرفتن n عدد از کاربر و سپس مرتب کردن آن با استفاده از الگوریتم مرتب‌سازی حبابی در کد ۷-۲ آمده است.

کد ۷-۲

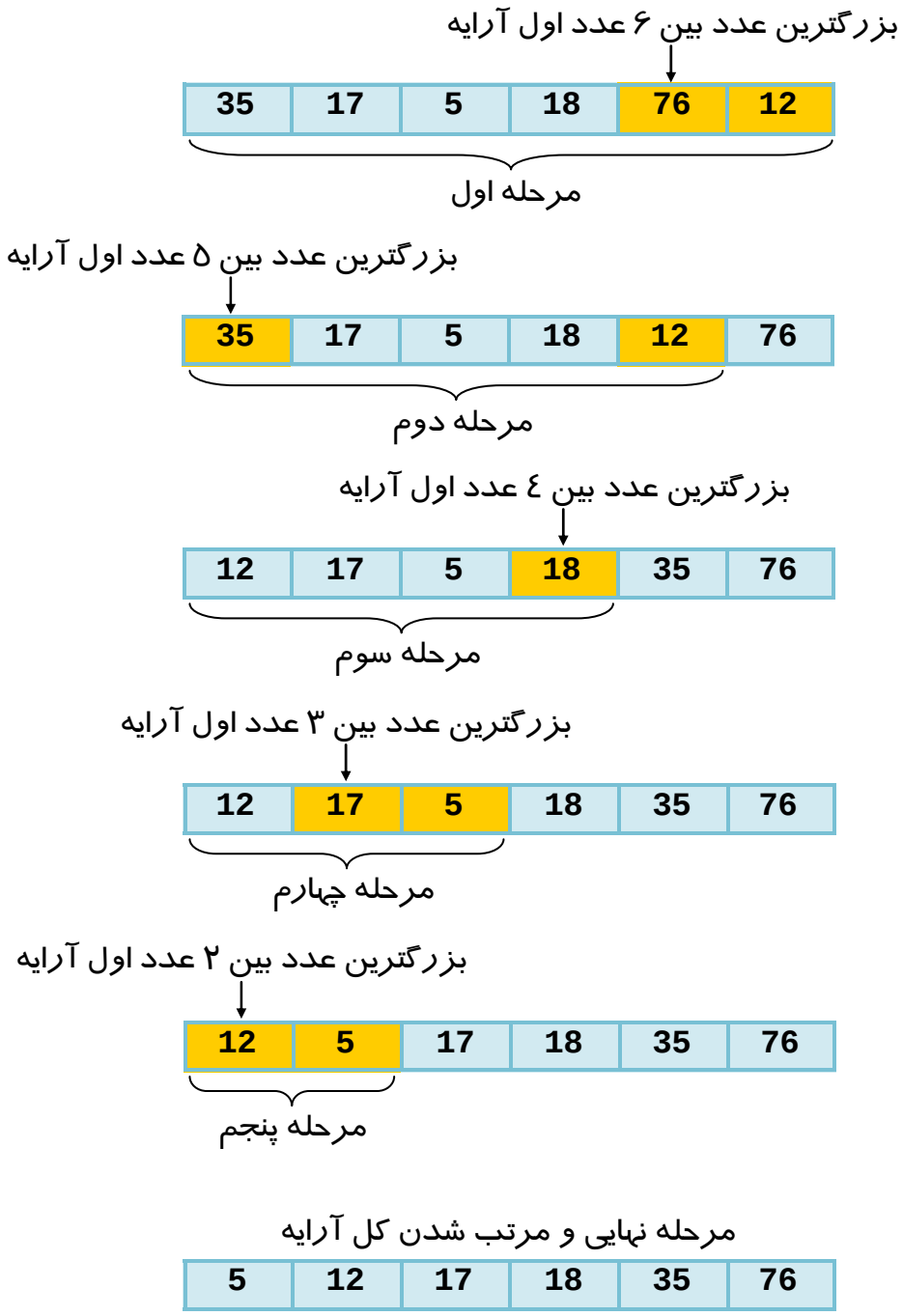
```
1  #include<iostream>
2  #include<conio.h>
3  using namespace std;
4  int main()
5  {
6     int i,j,n,t,a[100];
7     cout<<"enter n(n<=100):";
8     cin>>n;
9     for(i=0;i<n;i++)
10    {
11       cout<<"enter a["<<i+1<<"]:";
12       cin>>a[i];
13    }
14    for(i=n-1;i>0;i--)
15       for(j=0;j<i;j++)
16          if(a[j]>a[j+1])
17          {
18             t=a[j+1];
19             a[j+1]=a[j];
20             a[j]=t;
21          }
22    cout<<"The sorted list is:"<<endl;
23    for(i=0;i<n;i++)
24       cout<<a[i]<<endl;
25    getch();
26    return 0;
27 }
```

۴. مرتب‌سازی انتخابی[5] (*)

مرتب‌سازی انتخابی شبیه مرتب‌سازی حبابی است، منتها با این تفاوت که تعداد عملیات جابجا کردن (یا همان swap کردن) متغیرها در آن کمتر است. الگوریتم به این شکل است که ابتدا مقدار بیشینه را در آرایه پیدا کرده و سپس عضو انتهایی آرایه را با آن جابجا می‌کند تا به این ترتیب عدد بیشینه در انتهای آرایه قرار بگیرد. سپس مجدداً عدد بیشینه را در n-1 عدد اول آرایه پیدا کرده و با عدد n-1 جابجا می‌کند تا عدد بیشینه بین n-1 عدد در خانه‌ی n-1 اُم آرایه قرار بگیرد. به این ترتیب همانند مرتب‌سازی حبابی، پیش رفته و پس از رسیدن به دو عدد ابتدای آرایه، بزرگ‌ترین را بین آن‌ها انتخاب کرده و با عدد خانه‌ی دوم آرایه جابجا می‌کنیم تا کل آرایه مرتب شود. مراحل مرتب کردن یک آرایه به این روش در شکل ۷-۵ مشخص شده است.

به این ترتیب، از تعداد عملیات جابجایی (swap) کمتری نسبت به مرتب‌سازی حبابی استفاده می‌شود. البته این الگوریتم و الگوریتم مرتب‌سازی حبابی تقریباً از یک مرتبه‌ی زمانی در اجرا هستند و تفاوت خیلی اساسی ندارند. البته الگوریتم‌های کاراتری نسبت به مرتب‌سازی حبابی و انتخابی وجود دارد که در این فصل به آن‌ها پرداخته نمی‌شود و در کتاب دبیر به بررسی بعضی از آن‌ها پرداخته شده است که به صلاحدید دبیر در کلاس مطرح خواهد شد. برنامه‌ی مرتب‌سازی انتخابی در کد ۷-۳ نوشته شده است.

[5] Selection sort



شکل ۷-۵: مراحل مختلف مرتب سازی انتخابی؛ در هر مرحله خانه‌هایی که باید با هم تعویض شوند با رنگی متمایز مشخص شده‌اند.

کد ۷-۳

```
1  #include<iostream>
2  #include<conio.h>
3  using namespace std;
4  int main()
5  {
6     int i,j,n,t,max,a[100];
7     cout<<"enter n(n<=100):";
8     cin>>n;
9     for(i=0;i<n;i++)
10    {
11       cout<<"enter a["<<i+1<<"]:";
12       cin>>a[i];
13    }
14    for(i=n;i>0;i--)
15    {
16       max=0;
17       for(j=0;j<i;j++)
18       {
19          if(a[j]>a[max])
20          {
21             max=j;
22          }
23          t=a[max];
24          a[max]=a[i-1];
25          a[i-1]=t;
26       }
27    }
28    cout<<"The sorted list is:"<<endl;
29    for(i=0;i<n;i++)
30       cout<<a[i]<<endl;
31    getch();
32    return 0;
33 }
```

۵. غربال اِراتستن[۶] (*)

[6] Selection sort

یکی از روش‌های کارای پیدا کردن کلیه‌ی اعداد اول بین ۲ تا n، استفاده از غربال اِراتستن است. احتمالاً با این روش آشنایی دارید ولی برای یادآوری، یک‌بار دیگر الگوریتم یافتن کلیه‌ی اعداد اول از ۲ تا n توسط غربال اِراتستن را بیان می‌کنیم. فرض کنید n برابر ۱۵ باشد، به این ترتیب اگر کلیه‌ی اعداد ۲ تا ۱۵ را بنویسیم:

② ۳ ۴ ۵ ۶ ۷ ۸ ۹ ۱۰ ۱۱ ۱۲ ۱۳ ۱۴ ۱۵

دور ۲ به عنوان اولین عدد اول خط کشیده شده (یعنی ۲ به عنوان عدد اول علامت زده شده است) سپس، کلیه‌ی مضارب ۲ در بین این اعداد را، به عنوان عدد مرکب خط می‌زنیم:

② ۳ ~~۴~~ ۵ ~~۶~~ ۷ ~~۸~~ ۹ ~~۱۰~~ ۱۱ ~~۱۲~~ ۱۳ ~~۱۴~~ ۱۵

سپس اولین عدد باقی مانده و خط نخورده که قبلاً به عنوان عدد اول انتخاب نشده را به عنوان عدد اول بعدی انتخاب کرده و سپس مضارب آن را خط می‌زنیم:

② ③ ~~۴~~ ۵ ~~۶~~ ۷ ~~۸~~ ~~۹~~ ~~۱۰~~ ۱۱ ~~۱۲~~ ۱۳ ~~۱۴~~ ~~۱۵~~

و مجدداً همین عمل را تکرار می‌کنیم:

② ③ ~~۴~~ ⑤ ~~۶~~ ۷ ~~۸~~ ~~۹~~ ~~۱۰~~ ۱۱ ~~۱۲~~ ۱۳ ~~۱۴~~ ~~۱۵~~

② ③ ~~۴~~ ⑤ ~~۶~~ ⑦ ~~۸~~ ~~۹~~ ~~۱۰~~ ⑪ ~~۱۲~~ ⑬ ~~۱۴~~ ~~۱۵~~

دقت داشته باشید که پس از ۷، هر عددی که باقی مانده و خط نخورده عدد اول است، زیرا کم‌ترین مضرب آن باید دو برابر خودش باشد که از حد اعداد ما یعنی ۱۵ خارج است (زیرا اگر $n\geq ۸$ آن گاه $۲n\geq ۱۶$) پس هیچ عددی خط نخواهد خورد.

این الگوریتمی که بیان شد، شیوه‌ی اجرای الگوریتم غربال اِراتستن بر روی کاغذ است، اما برای اجرای این الگوریتم در کامپیوتر چه باید کرد؟ ابتدا باید مفهوم دنباله‌ی اعداد را مشخص کنیم، این کار با توجه به مفهوم آرایه که تعریف کردیم چندان سخت نیست، کافیست تا یک آرایه‌ی n تایی تعریف کنیم، البته چون در C اندیس آرایه از صفر شروع می‌شود و خود آرایه به $n-1$ ختم می‌شود، باید یک آرایه‌ی به طول $n+1$ تعریف کنیم.

سپس، همان‌طور که در تعریف نحوه‌ی اجرای الگوریتم غربال اِراتستن دیده شد، باید سه وضعیت برای هر عدد تعیین کرد، خط نخورده، خط خورده، عدد اول (یا اعدادی که دورشان خط کشیده شده) می‌توانیم به هرکدام از این وضعیت یک عدد نسبت بدهیم، مثلاً صفر برای خط نخورده، یک برای خط خورده و دو برای عدد اول. به این ترتیب در ابتدای کار که همه‌ی اعداد وضعیت خط نخورده دارند، باید در خانه‌های آرایه صفر وجود داشته باشد، سپس در ادامه‌ی کار، اگر آن عدد خط خورد، آن خانه را برابر ۱ قرار می‌دهیم و اگر به عنوان عدد اول تشخیص داده شد، آن خانه را برابر ۲ قرار می‌دهیم. نحوه‌ی تشخیص عدد اول نیز چنین است که از اول آرایه (در واقع از خانه‌ی ۲) شروع کرده و پیش می‌رویم تا اولین خانه‌ای که خط نخورده و قبلاً هم به عنوان عدد اول انتخاب نشده پیدا کنیم. در واقع اولین خانه‌ای که برابر ۱ یا ۲ نیست (زیرا یک به معنی خط خورده و دو به معنای عدد اول است) را برابر ۲ قرار می‌دهیم تا به عنوان عدد اول انتخاب شود. سپس کلیه‌ی مضارب آن را در آرایه برابر ۱ قرار

می‌دهیم که معادل همان خط زدن آن‌ها بر روی کاغذ است. به این ترتیب و با این توضیحات، برنامه‌ی غربال اِراتستن را می‌توان نوشت. متن برنامه‌ی غربال اِراتستن در کد ۷-۴ آمده است.

کد ۷-۴

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
   int i,j,k,n,a[1000];
   cout<<"enter n(n<1000):";
   cin>>n;
   for(i=0;i<=n;i++)
      a[i]=0;
   i=0;
   while(i<n/2)
   {
      j=2;
      while((a[j]!=0)&&(j<n/2))
         j++;
      i=j;
      a[j]=2;
      for(k=2*j;k<=n;k+=j)
         a[k]=1;
   }
   for(i=i;i<=n;i++)
      if(a[i]==0)
         a[i]=2;
   cout<<"Prime Numbers 2.."<<n<<":"<<endl;
   for(i=2;i<=n;i++)
      if(a[i]==2)
         cout<<i<<endl;
   getch();
   return 0;
}
```

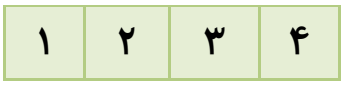
۶. اعداد صحیح بزرگ (*)

اعداد صحیحی که به صورت معمول در زبان C وجود دارند (مثل int یا long long int) ظرفیت محدودی برای نگهداری اعداد دارند. به عنوان مثال محدوده‌ی نگهداری اعداد برای int یا long long int در زیر آمده است:

int:	−2,147,483,648...2,147,483,647
long long int:	−9,223,372,036,854,775,808... 9,223,372,036,854,775,807

برای اعداد بزرگ‌تر، می‌توان از double استفاده کرد، اما باید به یاد داشت که اعداد double نیز در نگهداری تعداد ارقام بعد از ممیز محدودیت دارند. مثلاً اگر بخواهیم یک عدد ۴۰ رقمی صحیح داشته باشیم، این عدد به صورت کامل در یک عدد double ذخیره نمی‌شود، بلکه از جایی به بعد، عدد گرد می‌شود. مثلاً اگر بخواهیم عدد ۱۲۳۴۵۶۷۸۹۰۱۲۳۴۵۶۷۸۹ را با دقت ۹ رقم اعشار پس از ممیز نگه داریم، عدد مورد نظر به صورت $۱/۲۳۴۵۶۷۸۹۰\times۱۰^{۱۸}$ در خواهد آمد، یعنی ارقام ۱۲۳۴۵۶۷۸۹ بعد از ممیز به صفر گرد شده‌اند که این خطای ایجاد شده در بسیاری از کاربردها قابل قبول نیست.

یکی از راه‌های نگهداری و استفاده از اعداد صحیح بزرگ، به کار بردن آرایه‌هاست. به این ترتیب می‌توان در هر خانه‌ی آرایه یک رقم ذخیره کرد. به عنوان مثال می‌توان عدد ۱۲۳۴ را در آرایه مانند شکل زیر ذخیره کرد:



اگر طول آرایه بیش از عددی است که در آن ذخیره می‌شود، مثلاً در همین مثال قبل، اگر طول آرایه ۲۰ است ولی عدد ۴ رقمی است، لازم است که انتهای عدد با عددی غیر از ۰ تا ۹ مشخص شود، مثلاً می‌توانیم انتهای عدد را با ۱- مشخص کنیم:

اعدادی که بعد از ۱- هستند، دیگر معتبر نیستند (زرد رنگ)، زیرا ۱- به منزله‌ی پایان عدد است، پس بعد از ۱- دیگر قرار نیست رقمی بر رقم‌های عدد افزوده شود. پس بنابراین اگر خواستیم یک عدد صحیح بزرگ معرفی کنیم می‌توانیم بنویسیم:

```
short int a[100];
```

به این ترتیب a را به عنوان یک عدد حداکثر ۹۹ رقمی (خانه‌ی آخر آرایه محل ۱- انتهای آن است و جزء ارقام عدد محاسبه نخواهد شد) می‌توان در نظر گرفت. اما اولین نکته در مورد اعداد، گرفتن آن‌ها از ورودی است، یعنی کاربر بتواند عددی با استفاده از صفحه کلید وارد کند و ما آن را در a قرار بدهیم. تا به حال با استفاده از cin ورودی را می‌گرفتیم، اما نباید فراموش کرد که a یک آرایه است، پس نمی‌توان با استفاده از cin آن را خواند. همچنین اگر بخواهیم رقم به رقم عدد را با استفاده از cin بگیریم، چون به ازای هر بار خواندن با cin، باید یا Enter یا Space را فشرد، وارد کردن عدد به این شیوه نیز مطلوب نیست. بهترین کار

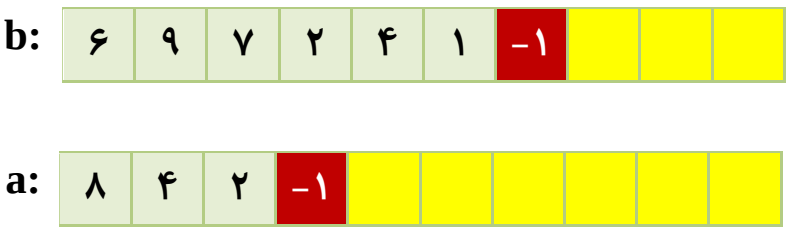
استفاده از getch(); است، زیرا کاربر تنها با فشردن کلیدها و سپس زدن Enter می‌تواند عدد را وارد کند. برای این کار باید مد نظر داشت که کد اَسکی کلیدهای ۰ تا ۹ برابر ۴۸ تا ۵۷ است، یعنی با کم کردن ۴۸ از کد کلیدهای ورودی، می‌توان به اعداد ۰ تا ۹ رسید. کد کلید Enter نیز ۱۳ است، بنابراین خواندن یک عدد از ورودی و ذخیره‌ی آن در a را می‌توان مانند کد ۷-۵ نوشت:

کد ۷-۵

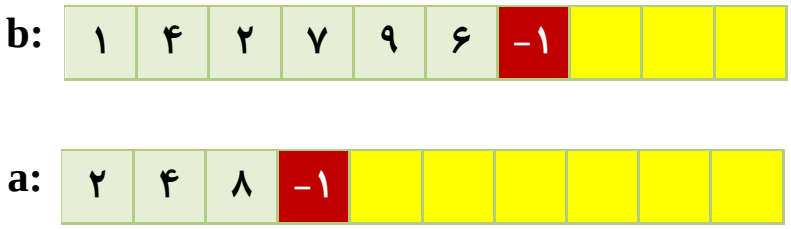
```
1  #include<iostream>
2  #include<conio.h>
3  using namespace std;
4  int main()
5  {
6     short int i,a[100];
7     char ch;
8     i=0;
9     while((i<100)&&(ch!=13))
10    {
11       ch=getch();
12       if((ch>='0')&&(ch<='9'))
13       {
14          cout<<ch;
15          a[i]=ch-48;
16          i++;
17       }
18    }
19    a[i]=-1;
20    return 0;
21 }
```

همان‌طور که در کد ۷-۵ دیده می‌شود، ابتدا باید مطمئن باشیم کلیدِ فشرده شده، کلید مربوط به عدد بوده، سپس آن را وارد آرایه کنیم، به این ترتیب جلوی خطاها نیز گرفته می‌شود. در انتها نیز عدد ۱- را باید در انتهای آرایه قرار دهیم (خط ۲۰ کد ۷-۵)، به معنای انتهای عدد.

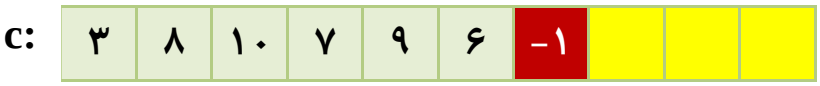
تا به اینجای کار موفق شدیم عدد را وارد آرایه کنیم که موفقیت کمی هم نیست! اما وارد کردن یک عدد به تنهایی، کار ساز نیست، زیرا عمده‌ی ارزش اعداد در کامپیوتر به قابلیت انجام محاسبه بر روی آن‌هاست. اکنون به حل این مسأله می‌پردازیم که اگر بنا باشد دو عددی که به روش گفته‌شده وارد دو آرایه‌ی a و b شده‌اند را با هم جمع کنیم، چطور باید این کار را انجام دهیم؟ یکی از مهم‌ترین مسایل در جمع زدن دو عدد این است که ممکن است دو عدد هم طول نباشند، یعنی مثلاً بخواهیم یک عدد ۳ رقمی را با یک عدد ۶ رقمی جمع کنیم، در این صورت a[0] و b[0] ارزش مکانی یکسانی ندارند، زیرا a[0] صدگان است و b[0] صدهزارگان، پس نمی‌توان آن‌ها را با هم جمع کرد. این مشکل را می‌توان با معکوس کردن دو عدد انجام داد، به این ترتیب a[0] و b[0] هر دو یکان خواهند شد، همین‌طور a[0] و b[1] هر دو دهگان و ... به عنوان مثال اگر a و b به ترتیب برابر ۸۴۲ و ۶۹۷۲۴۱ باشند به این شکل در آرایه ذخیره خواهد شد:



اگر دو عدد را در a و b بر عکس کنیم به این ترتیب خواهند شد:



حال می‌توان تا سه رقم دو عدد را جمع کرد و در آرایه‌ی سومی مثل c قرار دارد، پس از آن که عدد کوچک‌تر تمام شد، مابقی ارقام از عددی که هنوز تمام نشده (در این‌جا b)، عیناً در c یکی خواهد شد. به این ترتیب آرایه‌ی c گفته شده به ترتیب زیر خواهد شد:



اما همان‌طور که ملاحظه می‌شود، ممکن است در حین جمع زدن، برخی از ارقام بزرگ‌تر از ۹ شوند که در این صورت باید عمل ده بر یک را انجام داد، یعنی باقی‌مانده‌ی آن رقم بر ۱۰ را نگه داشت و خارج قسمت آن بر ۱۰ را با رقم بعدی جمع کرد. در مورد رقم آخر نیز، اگر در نهایت دو رقمی شد، یک بار دیگر باید طول عدد را افزایش داد، به این ترتیب که رقم آخر c چک می‌شود، اگر بزرگ‌تر از ۹ بود، طول عدد یک واحد افزایش می‌یابد و باقی‌مانده‌ی رقم آخر بر ۱۰ در خودش مانده و خارج قسمت آن بر ۱۰ در رقم بعدی ذخیره می‌شود. سپس ۱- در یک رقم بعد درج می‌شود. برای چاپ c نیز، می‌توان آن را معکوس کرده و سپس چاپ کرد، یا این که آن را برعکس نکنیم، ولی به ترتیب معکوس آن را چاپ کنیم. در کد ۷-۶ که

کل مراحل گفته شده در آن نوشته شده، از روش اول استفاده شده، یعنی عدد مجدداً معکوس شده است.

کد ۷-۶

```
1  #include<iostream>
2  #include<conio.h>
3  using namespace std;
4  int main()
5  {
6     int i,j,k,t,min,max;
7     short int a[100],b[100],c[100];
8     char ch=0;
9     i=0;
10    while((i<100)&&(ch!=13))
11    {
12       ch=getch();
13       if((ch>='0')&&(ch<='9'))
14       {
15          cout<<ch;
16          a[i]=ch-48;
17          i++;
18       }
19    }
20    a[i]=-1;
21    cout<<endl;
22    ch=0;
23    j=0;
24    while((j<100)&&(ch!=13))
25    {
26       ch=getch();
27       if((ch>='0')&&(ch<='9'))
28       {
29          cout<<ch;
30          b[j]=ch-48;
31          j++;
32       }
33    }
34    b[j]=-1;
```

```
35      cout<<endl;
36      for(k=0;k<i/2;k++)
37      {
38         t=a[k];
39         a[k]=a[i-k-1];
40         a[i-k-1]=t;
41      }
42      for(k=0;k<j/2;k++)
43      {
44         t=b[k];
45         b[k]=b[j-k-1];
46         b[j-k-1]=t;
47      }
48      if(i>j)
49      {
50         max=i;
51         for(k=0;k<j;k++)
52            c[k]=a[k]+b[k];
53         for(k=j;k<i;k++)
54            c[k]=a[k];
55      }
56      else
57      {
58         max=j;
59         for(k=0;k<i;k++)
60            c[k]=a[k]+b[k];
61         for(k=i;k<j;k++)
62            c[k]=b[k];
63      }
64      for(k=0;k<max-1;k++)
65      {
66         c[k+1]+=c[k]/10;
67         c[k]=c[k]%10;
68      }
69      if(c[max-1]>10)
70      {
71         c[max]=c[max-1]/10;
72         c[max-1]=c[max-1]%10;
73         c[max+1]=-1;
74         max++;
75      }
76      else
77      {
```

```
78          c[max]=-1;
79      }
80      for(k=0;k<max/2;k++)
81      {
82          t=c[k];
83          c[k]=c[max-k-1];
84          c[max-k-1]=t;
85      }
86      cout<<"Sum of two numbers="<<endl;
87      for(k=0;k<max;k++)
88          cout<<c[k];
89      getch();
90      return 0;
91  }
```

در کد ۷-۶ که نسبتاً مفصل نیز هست، کلیه‌ی مراحل گرفتن اعداد، جمع کردن و نمایش نتیجه‌ی نهایی نوشته شده است. با دقت در این مراحل، نکات بسیار خوبی راجع به برنامه‌نویسی آرایه‌ها خواهید آموخت.

آیا برای ضرب می‌توان از تعدادی جمع استفاده کرد؟ چه راه حل بهینه‌ای برای ضرب دو عدد حداکثر ۹۹ رقمی به ذهنتان می‌رسد؟

۷. رشته‌ها

مبحثی که در این بخش مطرح می‌شود، مبحث جدیدی نیست، زیرا عمده‌ی آن مربوط به همان آرایه‌هاست. رشته‌ها، در اصل همان آرایه‌های کاراکتری هستند که کمی امکانات بیشتر در اختیار قرار می‌دهند. به عنوان مثال اگر بخواهید رشته‌ی "Ali" را به عنوان یک ورودی از

کاربر بگیرید و در یک متغیر ذخیره کنید، این متغیر باید از جنس رشته باشد. ابتدا به کد ۷-۷ دقت کنید:

کد ۷-۷

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     string s;
7     cout<<"Enter your name:";
8     cin>>s;
9     cout<<"Hi "<<s<<"!"<<endl;
10    cout<<"Your name has "<<s.length()<<" characters";
11    getch();
12    return 0;
13 }
```

در این برنامه ابتدا یک رشته از ورودی توسط کاربر گرفته می‌شود، سپس جمله‌ی سلام بر اساس نام وارد شده چاپ می‌شود. در انتها نیز تعداد حروف نام وارد شده بر روی صفحه چاپ می‌شود. تعداد حروف رشته‌ی وارد شده، همان‌طور که از خط دهم کد ۷-۷ نیز مشخص است، به صورت یک دستور که **با نقطه از متغیر s منشعب شده** است به دست می‌آید. این گونه ساختارها، یعنی داشتن توابع و یا متغیرهایی در دل یک متغیر دیگر، در زبان C مرسوم است و نحوه‌ی به‌کارگیری آن آسان است. البته نحوه‌ی تعریف چنین متغیرهایی در زبان C را زودتر از فصل آخر نخواهیم دید. به عنوان مثال فرض کنید می‌خواهیم تساوی دو رشته را چک کنیم، کافی است از همان عملگر == که برای اعداد استفاده می‌کردیم استفاده کنیم. مفاهیم > و < نیز، به ترتیب به این معنا هستند: اگر a<b در مورد دو رشته، درست باشد، یعنی a، در ترتیب

حروف الفبایی، قبل از b خواهد آمد، یعنی اگر a و b را بر حسب حروف الفبا مرتب کنیم a زودتر از b ظاهر می‌شود. به عنوان مثال "sand"<"water" درست است اما، "balloon"<"acid" درست نیست، زیرا در ترتیب حروف الفبایی اول کلمه‌ی acid خواهد آمد و سپس balloon. برای علامت < هم درست عکس همین مطلب برقرار است.

علامت > یا < برای حروف بزرگ به این گونه عمل می‌کند که ابتدا حروف بـزرگ را در ترتیب حروف الفبایی در نظر می‌گیرد، یعنی "AB"<"ab" صحیح است، یعنی "AB" کوچک‌تر از "ab" خواهد بود.

چون رشته‌ها، ذاتاً آرایه هستند، می‌توان از عملگرهای [ ] برای دسترسی به خانه‌های آرایه استفاده کرد. به عنوان مثال اگر بنویسیم s="Hadi"، آن گاه s[0] برابر H، s[1] برابر a، s[2] برابر d و s[3] برابر i خواهد بود. به این ترتیب می‌توان به تک تک حروف یک رشته دسترسی داشت. فرض کنید به عنوان مثال، می‌خواهیم یک رشته به ورودی برنامه داده و سپس بزرگی و کوچکی حروف آن را برعکس کنیم، به عنوان مثال به یک ورودی و خروجی نمونه دقت کنید:

ورودی : **I write programs with C++**

خروجی : **i WRITE PROGRAMS WITH c++**

برای این کار کافی است بزرگ یا کوچک بودن یک حرف را چک کنیم، اگر حرف بزرگ بود آن را تبدیل به حرف کوچک کرده و اگر حرف کوچک بود آن را تبدیل به حرف بزرگ کنیم. برنامه‌ی انجام این کار در کد ۷-۸ آمده است.

کد ۷-۸

```
1  #include <iostream>
2  #include <conio.h>
3  #include <nodet.h>
4  using namespace std;
5  int main()
6  {
7     int i;
8     string s;
9     cout<<"Enter your string:";
10    getline(cin,s);
11    for(i=0;i<s.length();i++)
12       if(s[i]==chartoupper(s[i]))
13          s[i]=chartolower(s[i]);
14       else
15          s[i]=chartoupper(s[i]);
16    cout<<"Converted String:"<<s;
17    getch();
18    return 0;
19 }
```

باید دقت داشت، که cin و cout فاصله‌های سفید (مثل Space و Tab) را نه از کاربر می‌گیرند و نه اگر در رشته باشد، چاپ می‌کنند. راه حل، استفاده از دستور getline(cin,s); است (همچنان که در خط ۱۰ کد ۷-۸ آمده) تا فاصله‌های سفید را نیز بخواند (می‌توانید یک بار با cin عادی رشته‌ی s امتحان کنید، خواهید دید که تا اولین فاصله‌ی سفید بیشتر وارد s نخواهد شد).

حال می‌خواهیم برنامه‌ای بنویسیم که یک رشته و سپس یک کاراکتر گرفته و سپس رشته را از محل آن کاراکتر تجزیه کند، به عنوان مثال اگر رشته‌ی ورودی"abc,de,fg,h" باشد و کاراکتر ورودی ',' خروجی برنامه عبارتست از:

abc
de
fg
h

نوشتن این برنامه نیز بسیار ساده است و کافیست رسیدن به کاراکتر مورد نظر را با یک if چک کنیم و پس از رسیدن به آن کاراکتر، یک endl چاپ کنیم. البته صحیح‌تر این است که هر کدام از عبارات abc، de، fg و h را در یک رشته قرار دهیم و سپس آن‌ها را چاپ کنیم. ابتدا برنامه‌ی نوشته شده به روش اول یعنی چاپ endl را در کد ۷-۹ ببینید:

کد ۷-۹

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     int i;
7     char ch;
8     string s;
9     cout<<"Enter your string:";
10    getline(cin,s);
11    cout<<"Enter your separator character:";
12    cin>>ch;
13    for(i=0;i<s.length();i++)
14       if(s[i]==ch)
15          cout<<endl;
16       else
17          cout<<s[i];
18    getch();
```

```
19      return 0;
20  }
```

همان‌طور که می‌بینید این کد بسیار ساده است و تنها به جای کاراکتر مورد نظر، endl چاپ می‌کند. اما اگر بخواهیم هر رشته را درون یک متغیر بریزیم چه باید بکنیم؟ باید آرایه‌ای از رشته‌ها داشته باشیم تا هر کدام از رشته‌ها را در یکی از اعضای آرایه بریزیم. این مطلب تازه‌ای نیست و قبلاً هم این کار را کرده‌ایم، یعنی زمانی که می‌خواستیم تعدادی عدد را به صورت مرتب و پشت سر هم داشته باشیم، از آرایه استفاده می‌کردیم. به این ترتیب برای جدا کردن هر کدام از رشته‌های گفته شده با توجه به کاراکتر ورودی و قرار دادن آن‌ها داخل یک رشته‌ی مجزا، باید برنامه‌ی نوشته شده مطابق کد ۷-۱۰ باشد.

کد ۷-۱۰

```
1   #include <iostream>
2   #include <conio.h>
3   using namespace std;
4   int main()
5   {
6      int i,j,k;
7      char ch,str[256];
8      string s, t[100];
9      cout<<"Enter your string:";
10     getline(cin,s);
11     cout<<"Enter your separator character:";
12     cin>>ch;
13     j=0;
14     k=0;
15     for(i=0;i<s.length();i++)
16         if(s[i]==ch)
17         {
18            str[k]=0;
19            t[j]=string(str);
```

```
20              j++;
21              k=0;
22          }
23          else
24          {
25              str[k]=s[i];
26              k++;
27          }
28      if(k!=0)
29      {
30          str[k]=0;
31          t[j]=string(str);
32          j++;
33      }
34      for(i=0;i<j;i++)
35          cout<<t[i]<<endl;
36      getch();
37      return 0;
38  }
```

در این کد، از یک آرایه‌ی کاراکتری کمکی به نام str استفاده شده تا کاراکتر به کاراکتر هر بخش را بسازد، سپس، هر بار که به کاراکتر جدا کننده رسیدیم، با استفاده از تبدیل نوع، کل آرایه‌ی کاراکتری را به رشته (string) تغییر داده و در t[j] قرار می‌دهیم (خط ۱۹ کد ۷-۱۰).

۸. آرایه‌های دو بعدی

برای درک مفهوم آرایه‌های دو بعدی، کافیست به آخرین مسأله‌ای که حل کردیم رجوع کنیم. در این مسأله، آرایه‌ای از رشته‌ها تعریف کردیم، به این ترتیب t[i]، یعنی i+1 اُمین رشته در آرایه‌ی t، از طرفی t[i] خود یک رشته است، مثلاً "abc"، پس می‌توان به حروف آن

دسترسی داشت، یعنی t[i][0]، می‌شود اولین حرف رشته‌ی t[i] یا همان 'a'، t[i][1] می‌شود دومین حرف رشته‌ی t[i] یا همان 'b' و... پس بنابراین t، دیگر یک آرایه‌ی ساده نیست، بلکه آرایه‌ی دوبعدی است، زیرا با دو اندیس کنترل می‌شود، t[i][j] یعنی حرف j+1 اُم از رشته‌ی i+1 اُم، و با تغییر i و j، حرف خروجی هم عوض می‌شود. در حالت کلی، یک آرایه‌ی دو بعدی به این صورت تعریف می‌شود:

```
int a[M][N];
```

در این‌جا int نوع آرایه‌ای است که تعریف می‌کنیم و می‌تواند هر نوع دیگری هم باشد، M و N نیز طول آرایه را در هر بعد مشخص می‌کنند. به این ترتیب a یک جدول خواهد شد که از اندیس a[0][0] تا a[M-1][N-1] را شامل می‌شود:

	0	1		N-1
0	a[0][0]	a[0][1]	…	a[0][N-1]
1	a[1][0]	a[1][1]	…	a[1][N-1]
2	a[2][0]	a[2][1]	…	a[2][N-1]
	.	.	.	.
	.	.	.	.
	.	.	.	.
M-2	a[M-2][0]	a[M-2][1]	…	a[M-2][N-1]
M-1	a[M-2][0]	a[M-2][1]	…	a[M-1][N-1]

به این ترتیب a[i][j] خانه‌ای در سطر i+1 اُم و ستون j+1 اُم است. برای آشنایی بیشتر با آرایه‌های دو بعدی، برنامه‌ای می‌نویسیم که یک جدول ضرب 10 در 10 را در یک آرایه‌ی دو بعدی ذخیره کند و با گرفتن m و n از کاربر ($1 \le m,n \le 10$) یک جدول ضرب m در n بر

روی صفحه چاپ کند، منتها در هنگام چاپ، به جای محاسبه، اعداد از روی آرایه خوانده می‌شود. برنامه‌ی انجام این کار در کد ۷-۱۱ آمده است:

کد ۷-۱۱

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     int i,j,m,n,a[10][10];
7     for(i=0;i<10;i++)
8        for(j=0;j<10;j++)
9           a[i][j]=(i+1)*(j+1);
10    cout<<"Enter m,n:";
11    cin>>m>>n;
12    for(i=0;i<m;i++)
13    {
14       for(j=0;j<n;j++)
15          cout<<a[i][j]<<"\t";
16       cout<<endl;
17    }
18    getch();
19    return 0;
20 }
```

برنامه‌ی نوشته شده در کد ۷-۱۱ بسیار ساده است و تنها به هدف آشنایی با آرایه‌های دو بعدی آمده است.

فصل هشتم
هشت ضلعی

کاربافلپل

۱. حافظه‌ی موقتی و دائمی

فرض کنید برنامه‌ای نوشته‌ایم که ابتدا n و سپس n عدد از کاربر گرفته و در آرایه‌ای ذخیره کند و سپس آن‌ها را مرتب کند. در هنگام اجرا، n را برابر ۱۰ وارد کرده و شروع به وارد کردن اعداد می‌کنیم، پس از ورود ۵ عدد از این ۱۰ عدد، ناگهان برق قطع می‌شود! پس از وصل برق و روشن شدن کامپیوتر، آیا برنامه از محل قبلی خود اجرا می‌شود؟ یعنی اعدادی که وارد کرده‌ایم در حافظه‌ی کامپیوتر ثبت شده یا خیر؟

همان‌طور که می‌دانیم جواب سؤال بالا منفی است، یعنی اعدادی را که وارد کردیم در کامپیوتر ذخیره نشده‌اند. پس مفهوم حافظه در کامپیوتر چیست؟ چرا این اعداد که در حافظه‌ی کامپیوتر بودند پاک شدند؟ در پاسخ باید گفت دو نوع حافظه وجود دارد: حافظه‌ی موقتی و حافظه‌ی دائمی. حافظه‌ی موقت که معمولاً سرعت بسیار بالاتری نسبت به حافظه‌های دائمی دارد، تنها هنگامی که جریان برق برقرار است می‌تواند متغیرها را ذخیره کند و در صوت قطع جریان برق، کلیه‌ی اطلاعاتش پاک خواهد شد. حافظه‌ی RAM کامپیوتر (که در مدل ساده‌ی کامپیوتر در شکل ۲-۱ نمایش داده شده) از نوع حافظه‌ی موقتی است، بنابراین برای نگهداری دائمی اطلاعات نیاز به یک حافظه‌ی دیگر داریم. حافظه‌ی دیسک سخت[1] کامپیوتر این امکان را برای ما فراهم می‌آورد تا اطلاعات خود را در قالب فایل[2] بر روی آن ذخیره کنیم. حتماً تا به حال با فایل‌های زیادی سروکار داشته‌اید، مثل فایل‌های عکس، متنی ساده، ورد[3] و اِکسِل[4] یا هر فایل دیگری. فایل‌ها برای خود نام و پسوند دارند (البته معمولاً پسوند فایل‌ها در ویندوز دیده نمی‌شود، برای این که پسوند فایل‌ها را ببینید کافی است وقتی پنجره‌ی My

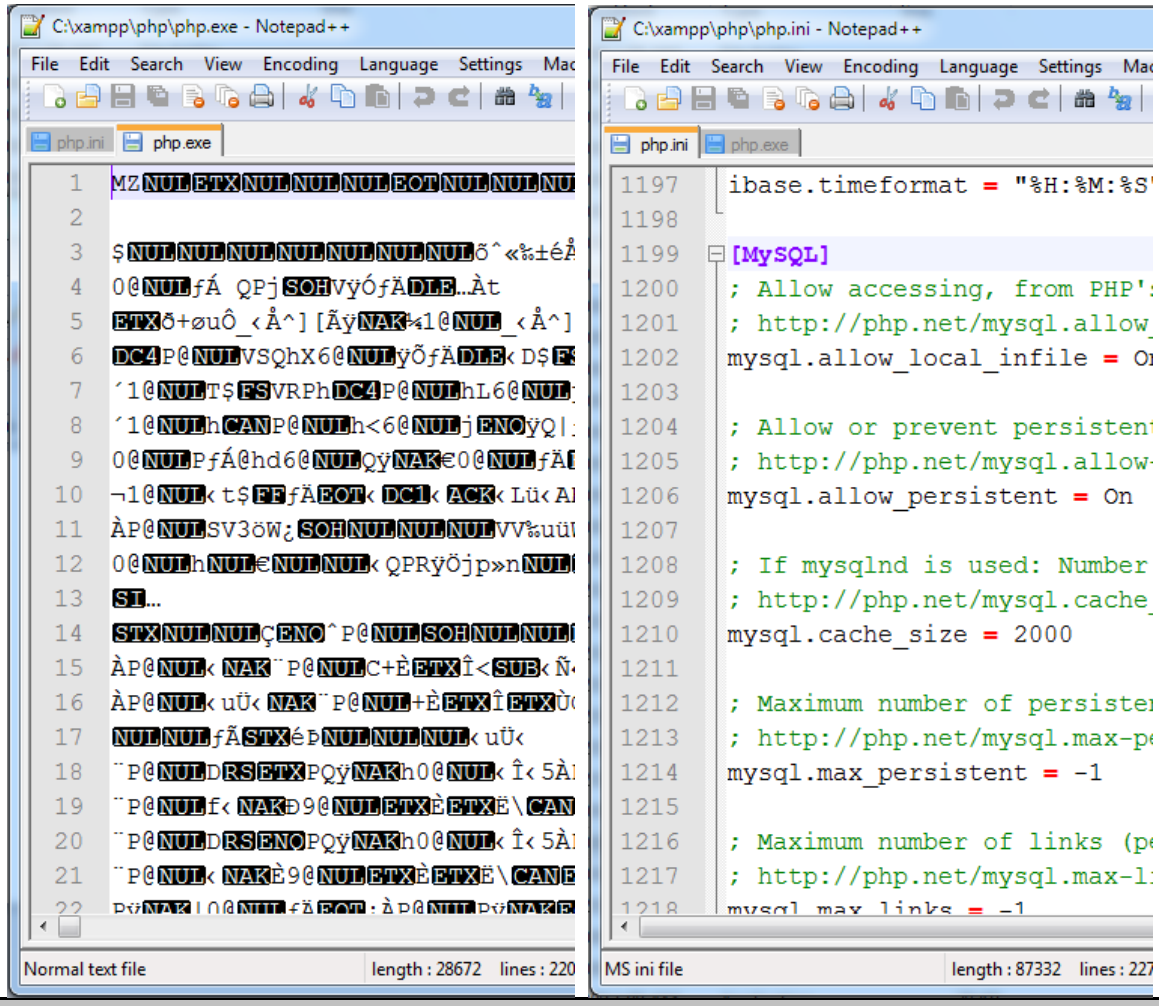
[1] Hard Disk
[2] File
[3] Word
[4] Excel

Computer باز است Alt+T را فشرده و از منوی ظاهر شده Folder Options را انتخاب کنید و سپس در بخش View تیک گزینه‌ی Hide Extensions for known file types را بردارید)، مثلاً یک فایل متنی ساده (که معمولاً با notepad باز می‌شود) پسوند txt دارد، یعنی مثلاً اگر اسم فایل File1 و نوع آن متنی ساده باشد، نام کامل فایل همراه با پسوندش File1.txt خواهد بود. البته این پسوندها صرفاً یک راهنما هستند که می‌توانند اطلاعات غلط نیز داشته باشند، مثلاً اگر پسوند همان فایل File1.txt را از txt به xlsx تغیر دهید، آیکن آن به شکل یک فایل اِکسِل در می‌آید (به خاطر پسوندش که عوض شده و آن را یک فایل اِکسِل معرفی می‌کند)، ولی اگر بر روی آن دو بار کلیک کنید به درستی باز نخواهد شد. فایل‌ها همچنین یک آدرس کلی نیز دارند، مثلاً اگر همان File1.txt گفته شده در فولدر Folder1 در درایو C قرار داشته باشد، آدرس کامل آن C:\Folder1\File1.txt خواهد بود. هیچ دو فایلی با نام و آدرس یکسان بر روی دیسک سخت کامپیوتر نمی‌توانند وجود داشته باشند، پس نام و آدرس فایل‌ها یکتاست.

۲. انواع فایل و نحوه‌ی خواندن محتوای فایل‌های متنی

فایل‌ها بر اساس نوع ذخیره شدن به دو دسته‌ی کلی تقسیم می‌شوند: متنی و دودویی. ما در این کتاب به خواندن از فایل‌های دودویی (نظیر عکس‌ها، فیلم‌ها، فایل وُرد و ...) نمی‌پردازیم. به زبان خیلی ساده فایل‌های متنی، همان فایل‌هایی هستند که اگر آن‌ها را با notepad (یا هر ویرایشگر متن ساده مثل notepad++ و EmEditor) باز کنیم، داخل آن را می‌فهمیم، یعنی عبارات و اعدادی نوشته شده که غیرعادی و عجیب و غریب نیست! برعکس، اگر فایل‌های دودویی (نظیر یک فایل jpg یا bmp) را با notepad باز کنیم، کاراکترهای عجیب و غیر قابل

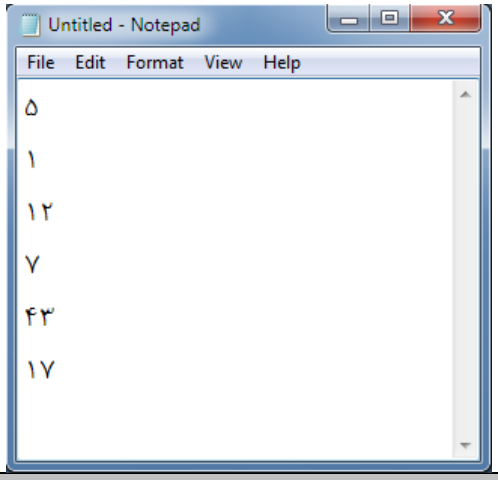
فهم می‌بینیم. در شکل ۸-۱ این مقایسه بین دو فایل متنی و دودویی بهتر انجام داده شده است.



شکل ۸-۱: نمایش یک فایل متنی (راست) ساده و یک فایل دودویی (چپ) در کنار یکدیگر

فایل‌های متنی را می‌توان به راحتی توسط notepad ساخت، به عنوان مثال، فرض کنید می‌خواهیم یک عدد n و سپس n عدد را (هر کدام در یک سطر) در یک فایل قرار دهیم و سپس از فایل ابتدا عدد n و سپس n عدد را خوانده و همه‌ی آن‌ها و حاصل جمعشان را چاپ کنیم. در این صورت ابتدا باید فایل را بسازیم، فرض کنید می‌خواهیم ۵ عدد ۱ و ۱۲ و ۷ و

۴۳ و ۱۷ را داخل فایل بنویسیم، در این صورت چیزی که در notepad می‌نویسیم باید شبیه شکل ۸-۲ باشد.



شکل ۸-۲: ساختن یک فایل متنی دلخواه در notepad

حالا باید فایل را ذخیره کنیم، مثلاً با نام input.txt، سپس این فایل را در کنار فایل برنامه‌ای که می‌خواهیم بنویسیم قرار می‌دهیم تا احتیاج به آدرس‌دهی کامل نباشد. تنها نکته‌ای که باقی مانده این است که چگونه باید از روی فایل، این اعداد را بخوانیم؟ پاسخ آن بسیار ساده است! ابتدا به برنامه‌ای که گفته شد در کد ۸-۱ دقت کنید:

کد ۸-۱

```
1  #include<iostream>
2  #include<fstream>
3  #include<conio.h>
4  using namespace std;
5  int main()
6  {
```

```
7     ifstream i;
8     int n,S=0,x,k;
9     i.open("input.txt",ios::in);
10    i>>n;
11    for(k=0;k<n;k++)
12    {
13       i>>x;
14       S+=x;
15       cout<<x<<endl;
16    }
17    i.close();
18    cout<<"Sum="<<S;
19    getch();
20    return 0;
21 }
```

شاید در نگاه اول کد ۸-۱ کمی پیچیده به نظر برسد ولی اگر کمی دقت کنید واقعاً ساده است! کافیست قدم به قدم کد را با هم مرور کنیم: اولین خط کد ناآشنا در خط ۲ کد ۸-۱ دیده می‌شود که <fstream> را به لیست فایل‌های حاوی دستورات اضافه کرده است، این کار برای آن است که در متن برنامه بتوانیم از نوع متغیرها و دستورات کار با فایل استفاده کنیم. سپس اولین کاری که در main در مورد کار با فایل انجام داده‌ایم تعریف متغیری از نوع ifstream است (خط ۷ کد ۸-۱)، یعنی متغیری که قابلیت اتصال به یک فایل در دیسک سخت و خواندن از روی آن را دارد، به این ترتیب، i از نوع فایل ورودی (یعنی برای خواندن، نه برای نوشتن) تعیین می‌شود. سپس در خط ۹ کد ۸-۱ مجدداً به کدی جدید برخورد می‌کنیم، این خط برای آن است که فایل input.txt (که باید در کنار فایل برنامه‌ی ما باشد) باز شود و به عنوان ورودی تعریف شود (این تعریف به عنوان ورودی، از روی ios::in نیز معلوم می‌شود، یعنی چون نوع فایل ورودی است، ios::in را به عنوان پارامتر دوم open ذکر کردیم، وگرنه آن را ios::out یا چیز دیگری قرار می‌دادیم).

اگر فایل input.txt یا هر فایل دیگری که می‌خواهیم آن را در برنامه باز کنیم، در کنار فایل برنامه‌ی ما نباشد، باید آدرس کامل آن را ذکر کنیم مثلاً اگر این فایل در درایو D و در فولدر MyFolder است، باید آدرس فایل را به جای "input.txt"، "D:\\MyFolder\\input.txt" ذکر کنیم. دلیل این که به جای \ باید\\ بگذاریم این است که \ در رشته‌ها یک کاراکتر ویژه است که با سایر کاراکترها ترکیب می‌شود و معنای خاصی به خود می‌گیرد، مثلاً "\t" به معنای TAB، "\n" به معنای Enter یا endl یا همان رفتن به خط بعد است، پس خود\ باید به صورت \\ بیاید تا برنامه برای فهم آن دچار اشکال نشود.

پس از آن که به فایل متصل شدیم، حال می‌توانیم از آن عدد بخوانیم. این کار به راحتی و درست شبیه cin کردن اعداد انجام می‌شود (خط ۱۰ کد ۸-۱)، یعنی با نوشتن ;i>>n، اولین عدد در فایل (که مطابق قراردادی که کرده بودیم باید تعداد کل بقیه‌ی اعداد موجود در فایل باشد) خوانده شده و در n وارد می‌شود، دقیقاً مثل زمانی که عدد را وارد کرده و سپس Enter را می‌زنیم. سپس، هر بار که ;i>>x را می‌نویسیم، عدد جدیدی از فایل خوانده شده و وارد x می‌شود. این خواندن، به ترتیب انجام می‌شود و همواره رو به جلو پیش می‌رویم، یعنی بازگشت به عقب نداریم، درست مثل زمانی که با cin ورودی می‌گرفتیم. در این مثال (شکل ۸-۲) n برابر ۵ شده و سپس x به ترتیب مقادیر ۱ و ۱۲ و ۷ و ۴۳ و ۱۷ را به خود می‌گیرد و برنامه اجرا می‌شود. بین هر دو ورودی در فایل باید یک فضای سفید[5] وجود داشته باشد، یعنی حداقل یکی (و یا ترکیبی) از Space، TAB یا Enter بین دو متغیر مجزا موجود باشد. در انتها که کارمان با فایل تمام شد، باید فایل را ببندیم (خط ۱۷ کد ۸-۱). به این ترتیب فرایند اصلی کار با فایل را با هم مرور کردیم.

[5] White Space

فایل به غیر از خاصیت نگهداری داده‌ها یک خاصیت مهم دیگر نیز دارد و آن این که ما را از وارد کردن چند باره‌ی اطلاعات راحت می‌کند. مثلاً فرض کنید می‌خواهیم یک برنامه‌ی مرتب‌سازی بنویسیم، به این ترتیب هر بار باید n عدد وارد کنیم تا ببینیم برنامه‌ای که نوشته‌ایم درست کار می‌کند یا نه؟ در صورتی که اگر ورودی‌ها از فایل باشد، یک‌بار آن‌ها را در notepad نوشته و ذخیره می‌کنیم، سپس هر چند بار که برنامه اجرا شود، اعدادی از روی فایل خوانده می‌شوند و احتیاجی به ورود اطلاعات توسط ما نیست.

۳. خواندن از فایل و مرتب سازی داده‌ها و نوشتن خروجی در فایل

اکنون می‌پردازیم به مثالی دیگر، فرض کنید می‌خواهیم از فایلی، ابتدا یک عدد n و سپس n عدد بخوانیم (درست مثل مثال قبل) و این n عدد را در یک آرایه قرار داده و سپس آن آرایه را به صورت صعودی مرتب کنیم و سپس نتیجه‌ی مرتب شده را در فایلی دیگر بنویسیم. فرض می‌کنیم فایل ورودی input.txt و فایل خروجی output.txt نام دارند. در این مثال، می‌خواهیم نوشتن در فایل را نیز به برنامه اضافه کنیم. برای این کار کافیست یک متغیر از جنس ofstream برای نوشتن در فایل تعریف کنیم؛ و سپس هنگام باز کردن فایل، پارامتر دوم را به جای ios::in، ios::out قرار دهیم. در این صورت، چنانچه فایل وجود نداشته باشد ایجاد می‌شود و اگر وجود داشته باشد، کلیه‌ی محتوای قبلی آن پاک می‌شود و آماده می‌شود برای نوشتن بر روی آن.

اگر نخواهیم کلیه‌ی محتویات قبلی پاک شود و هرچه که می‌نویسیم بـه انتهـای فایل اضافه شود، باید به جای ios::out از ios::app استفاده کنیم.

به این ترتیب مراحل کار به این صورت خواهد بود:

۱) باز کردن فایل input.txt

۲) خواندن اعداد از فایل input.txt

۳) بستن فایل input.txt

۴) مرتب‌سازی اعداد

۵) باز کردن فایل output.txt

۶) نوشتن اعداد مرتب شده در فایل output.txt

۷) بستن فایل output.txt

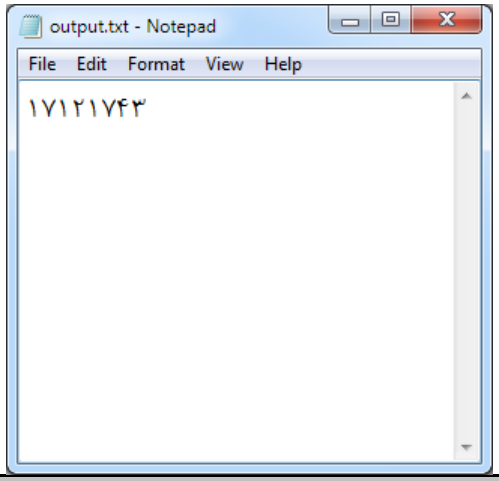
متن برنامه‌ی مورد نظر که توضیحات آن داده شد در کد ۸-۲ آمده است.

کد ۸-۲

```
1  #include<iostream>
2  #include<fstream>
3  #include<conio.h>
4  using namespace std;
5  int main()
6  {
7     ifstream i;
8     ofstream o;
9     int n,S=0,k,j,t,a[1000];
10    i.open("input.txt",ios::in);
```

```
11      i>>n;
12      for(k=0;k<n;k++)
13         i>>a[k];
14      i.close();
15      for(k=n-1;k>0;k--)
16         for(j=0;j<k;j++)
17            if(a[j]>a[j+1])
18            {
19               t=a[j];
20               a[j]=a[j+1];
21               a[j+1]=t;
22            }
23      o.open("output.txt",ios::out);
24      for(k=0;k<n;k++)
25         o<<a[k]<<endl;
26      o.close();
27      return 0;
28  }
```

همان‌طور که در کد ۸-۲ مشاهده می‌شود، نوشتن در فایل نیز دقیقاً مثل استفاده از cout است و حتی ترتیب ایجاد خروجی نیز همان‌گونه است. دقت کنید اگر در خط ۲۵ کد ۸-۲ از endl استفاده نکنیم، اعداد مرتب شده از هم جدا نمی‌شوند و به هم می‌چسبند، شبیه شکل ۸-۳، که استفاده‌ی مجدد از آن را ناممکن می‌سازد، چون مرز بین اعداد از بین رفته است.



شکل ۸-۳: عدم استفاده از endl در فایل خروجی و چسبیدن کلیه‌ی اعداد به یکدیگر

۴. مشخص شدن انتهای فایل

حال فرض کنید می‌خواهیم تعدادی عدد از فایل ورودی بخوانیم، اما نمی‌دانیم چند عدد در فایل وجود دارد؟ یعنی اولین عدد فایل، تعداد بقیه‌ی اعداد را مشخص نمی‌کند. در این حالت راه حل این است که این قدر عدد بخوانیم تا به انتهای فایل[۶] برسیم. متغیرهایی از جنس ifstream، ofstream، به غیر از توابع open و close، چند تابع دیگر نیز دارند. مثلاً ifstream تابعی با نام eof() دارد که اگر مقدار درستی برگرداند یعنی به انتهای فایل رسیده‌ایم و اگر مقدار نادرست برگرداند یعنی هنوز به انتهای فایل نرسیده‌ایم. بر همین اساس می‌توان آن قدر از فایل عدد خواند تا به انتهای فایل برسیم. البته معمولاً بهتر است برای اینکه آخرین عدد موجود در فایل هم خوانده شود، یک خط خالی بعد از آن در فایل درج شود. به عنوان مثال برنامه‌ی نوشته شده در کد ۸-۳ برای خواندن کلیه‌ی اعداد فایل input.txt و چاپ آن بر روی صفحه است.

کد ۸-۳

```
1  #include<iostream>
2  #include<fstream>
3  #include<conio.h>
4  using namespace std;
5  int main()
6  {
7     ifstream i;
8     int k,x;
9     i.open("input.txt",ios::in);
10    while(!i.eof())
11    {
12       i>>x;
13       if(i.good())
```

[6] End of File

```
14          cout<<x<<endl;
15      }
16      i.close();
17      getch();
18      return 0;
    }
```

تابع good() که در خط ۱۳ کد ۸-۳ نوشته شده است نیز یک تابع است که اگر مقدار درستی برگرداند، یعنی آخرین عملیات انجام شده (خواندن/نوشتن) موفقیت‌آمیز بوده است و در غیر این صورت عملیات انجام شده معتبر نیست، یعنی مثلاً اگر عددی را از فایل خوانده‌ایم، آن عدد معتبر نیست. به این ترتیب و با استفاده از این برنامه، می‌توانید میزان قابلیت اطمینان[۷] برنامه‌های خود را بالا ببرید.

توابع دیگری مربوط به متغیرهای فایلی وجود دارد که مربوط به بررسی وضعیت فایل هستند، به عنوان مثال، تابع fail() زمانی مقدار درست برمی‌گرداند که باز کردن فایل با شکست مواجه شده باشد (چه برای خواندن، چه برای نوشتن). مثلاً ممکن است فایل مورد نظر وجود نداشته باشد یا برای نوشتن یک فایل، ما اجازه‌ی نوشتن روی دیسک سخت را نداشته باشیم. بنابراین خوب است هرگاه فایلی را باز می‌کنیم، چک کنیم که آیا به درستی باز شده یا خیر؟ و اگر فایل به درستی باز نشده بود پیغام خطایی چاپ کرده و برنامه را ادامه ندهیم.

۵. ذخیره کردن محل نشان‌گر موس

پس از این توضیحات فرض کنید می‌خواهیم برنامه‌ای بنویسیم که پس از اجرا وارد محیط گرافیکی شود، سپس هرگاه موس تکان خورد، مکان جدید آن را در فایل cursor.txt ذخیره

[7] Reliability

کند، به این صورت که ابتدا x و سپس با یک فاصله y مختصات اشاره‌گر موس را داخل فایل نوشته و سپس به اول خط بعد می‌رود (با یک endl). برنامه با فشردن یک کلید از روی صفحه‌کلید، خاتمه می‌یابد. این برنامه در کد ۸-۴ آمده است.

کد ۸-۴

```
#include <iostream>
#include <fstream>
#include <graphics.h>
using namespace std;
int main()
{
   int x=-100,y=-100;
   ofstream o;
   o.open("cursor.txt",ios::out);
   if(o.fail())
   {
      cout<<"Error creating Cursor.txt...";
      getch();
   }
   else
   {
      initwindow(1280,800,"",-3,-3);
      while(!kbhit())
      {
         if((x!=mousex())||(y!=mousey()))
         {
            x=mousex();
            y=mousey();
            o<<x<<" "<<y<<endl;
         }
      }
      o.close();
      closegraph();
   }
   return 0;
}
```

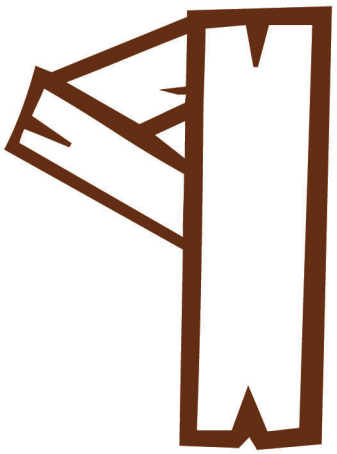
حال فرض کنید می‌خواهیم از فایل cursor.txt مرحله‌ی قبل استفاده کنیم و با فواصل زمانی مشخص (مثلاً هر ۱۰۰ میلی ثانیه یک‌بار) هر بار مختصات یک نقطه را خوانده و در آن نقطه یک دایره با شعاع ۱۰ پیکسل ترسیم کنیم. به این ترتیب مسیر حرکت موس در برنامه‌ی قبلی، در این برنامه نمایش داده می‌شود. متن این برنامه که بسیار ساده است در کد ۸-۵ آمده است.

کد ۸-۵

```
1   #include <iostream>
2   #include <fstream>
3   #include <graphics.h>
4   using namespace std;
5   int main()
6   {
7      int x=-100,y=-100;
8      ifstream i;
9      i.open("cursor.txt",ios::in);
10     if(i.fail())
11     {
12        cout<<"Error creating Cursor.txt...";
13        getch();
14     }
15     else
16     {
17        initwindow(1280,800,"",-3,-3);
18        while(!i.eof())
19        {
20           i>>x>>y;
21           if(i.good())
22           {
23               circle(x,y,10);
24               delay(100);
25           }
```

```
26          }
27          i.close();
28          closegraph();
29      }
30      return 0;
31  }
```

به این ترتیب با نحوه‌ی کار با فایل‌های متنی در کامپیوتر آشنا شدید. این فصل نکته‌ی خیلی زیادی نسبت به سایر فصول نداشت و تنها با کمی دقت در صورت مسأله و کمی تمرین در این زمینه، به راحتی مطالب این فصل را فرا خواهید گرفت.



فصل نهم
همه با هم

برنامه‌نویسی پیمانه‌ای

۱. مفهوم و مزایای تابع و برنامه‌نویسی پیمانه‌ای[1]

تا رسیدن به این فصل، دستورات زیادی را دیده‌ایم، نظیر sqrt، getch، circle و...، هدف از به کار بردن این دستورات، انجام یک کار مشخص بوده است. مثلاً sqrt(x)، ریشه‌ی دوم x یا همان رادیکال x را محاسبه می‌کند، getch() کد یک کلید که از صفحه‌کلید فشرده شده برمی‌گرداند، circle یک دایره ترسیم می‌کند و ... پس کار این دستورات یا توابع، این است که با گرفتن یک سری ورودی، یک خروجی تولید کنند. تا به حال ما از این دستورات یا توابع به صورت آماده استفاده می‌کردیم، اما در این فصل می‌خواهیم با نحوه‌ی نوشتن توابع دلخواه خود آشنا شویم، یعنی خودمان دستور و تابع بنویسیم و استفاده کنیم. دلیل نوشتن دستورات و توابع توسط برنامه‌نویس چند مورد است.

۱) نوشتن تابع، باعث می‌شود متن اصلی برنامه خلاصه شود. مثلاً فرض کنید اگر بخواهیم تکه کدی بنویسیم که دایره ترسیم کند، لازم باشد که ۱۰ خط بنویسیم، در این صورت هر کجا که بخواهیم یک دایره ترسیم کنیم، باید این ۱۰ خط را بنویسیم، حال آن که اگر کل این ۱۰ خط را در یک تابع خلاصه کنیم (نظیر تابع circle) کافیست در یک خط آن را صدا بزنیم. این خلاصه نویسی باعث افزایش خوانایی برنامه می‌شود، یعنی راحت‌تر می‌توان برنامه را خواند و فهمید.

۲) تصحیح متن کد با نوشتن توابع آسان می‌شود، به طور مثال فرض کنید در ۱۰ خط کدی که قرار بود دایره ترسیم کند، ۲ اشتباه تایپی وجود داشته باشد و ما ۱۰ جای برنامه احتیاج به ترسیم دایره داشته باشیم و ۱۰ بار این کد را کپی کرده باشیم، زمانی که به اشتباه خود پی بردیم باید بگردیم و یکی یکی این ۱۰ جا را پیدا کرده و غلط‌ها را

[1] Modular Programming

رفع کنیم (این کار زمانی که برنامه بزرگ شود کار بسیار سخت‌تری خواهد بود)، در حالی که اگر یک بار تابع تعریف کرده باشیم و در برنامه ۱۰ جا آن را صدا زده باشیم، در هر ۱۰ جا، برای اجرای نهایی به کد تعریف تابع رجوع می‌شود، پس اگر یک‌بار کد تعریف تابع را تصحیح کنیم کل کد ما تصحیح می‌شود.

۳) اشکال‌زدایی برنامه با وجود توابع راحت‌تر می‌شود، زیرا تعداد خطوط برنامه‌ی اصلی کاهش می‌یابد و برنامه به تعدادی زیر برنامه‌ی مشخص شکسته می‌شود که هر کدام وظیفه‌ی مشخصی دارد که می‌توان درستی کارکرد هرکدام را به صورت جداگانه بررسی کرد و چون هر زیر برنامه یا تابع حجم کم‌تری نسبت به کل برنامه دارد این بررسی راحت‌تر خواهد بود.

به شکستن برنامه به یک سری زیر بخش (که معمولاً تابع هستند) و استفاده از مزایایی که گفته شد، برنامه‌نویسی پیمانه‌ای می‌گویند.

پس از ادای این توضیحات، می‌پردازیم به معرفی انواع توابع در زبان C و تعریف پارامتر ورودی و خروجی تابع. یک مفهوم که در بسیاری از موارد ممکن است باعث ابهام شود مفهوم خروجی است، خروجی در تابع به معنای هر خروجی در برنامه (نظیر چاپ یک عدد بر روی صفحه نمایش یا ترسیم یک خط) نیست، بلکه معنای خروجی در تابع، عددی است که حاصل اجرای آن تابع است. مثلاً خروجی تابع sqrt(x)، رادیکال x است یا خروجی تابع getch() کد یک کلید است، بنابراین می‌توان متغیری را مساوی این توابع قرار داد، مثلاً می‌توانیم بنویسیم y=sqrt(x); که به معنای آن است که "رادیکال x را محاسبه کن و در y قرار بده"، و یا ch=getch(); به معنای آن است که"منتظر فشرده شدن یک کلید باش و سپس کد آن را در ch قرار بده". پس مفهوم خروجی در تابع باید مشخص شده باشد، به عنوان مثال اگر تابعی

بنویسیم که یک مقدار را بر روی صفحه نمایش چاپ کند، ولی آن را برنگرداند، خروجی محسوب نمی‌شود، زیرا نمی‌توان متغیری را مساوی با آن قرار داد. در مورد این مفهوم در این فصل بیشتر صحبت خواهیم کرد.

مفهوم دیگری که حائز اهمیت است، لیست پارامترهای ورودی است. هر تابع می‌تواند ورودی‌هایی داشته باشد، نظیر circle که سه ورودی می‌گیرد. یا sqrt که یک ورودی می‌گیرد. البته برخی از توابع نیز ورودی نمی‌گیرند مثل getch، یعنی کارکرد آن وابسته به پارامتر ورودی نخواهد بود. بنابراین بر اساس ورودی داشتن یا نداشتن و خروجی داشتن یا نداشتن، ۴ حالت پیش می‌آید:

۱) تابع هم پارامتر ورودی و هم پارامتر خروجی داشته باشد: مثل sqrt
۲) تابع پارامتر ورودی داشته باشد ولی خروجی نداشته باشد: مثل circle
۳) تابع پارامتر ورودی نداشته باشد ولی خروجی داشته باشد: مثل getch
۴) تابع نه پارامتر ورودی داشته باشد و نه خروجی: مثل cleardevice

بار دیگر در مورد حالت (۲) در لیست بالا تذکر می‌دهیم، خروجی داشتن به معنای برگرداندن عددی به عنوان حاصل محاسبات است، نه یک ترسیم، به عنوان مثال نمی‌توانیم بنویسیم y=circle(100,100,50); چون معنایی ندارد، زیرا تابع circle چیزی را حساب نمی‌کند بلکه تنها یک شکل دایره در محیط گرافیکی ترسیم می‌کند.

۲. نحوه‌ی تعریف تابع در زبان C

پس از معرفی مفهوم تابع و بررسی انواع آن از نظر ورودی و خروجی، می‌پردازیم به نحوه‌ی تعریف تابع در زبان C. فرض کنید می‌خواهیم تابعی با نام zarb بنویسیم که دو پارامتر ورودی صحیح گرفته و حاصل‌ضرب آن‌ها را برگرداند، در این صورت باید آن را به صورت نوشته شده در کد ۹-۱ بنویسیم:

کد ۹-۱

```
1  int zarb(int x, int y)
2  {
3     return x*y;
4  }
```

int نوشته شده در پشت نام تابع، نوع خروجی را مشخص می‌کند. در این‌جا چون نوع خروجی یک عدد صحیح است، int نوشته شده. اگر نوع خروجی یک عدد اعشاری بود double یا float می‌نوشتیم، اگر یک کاراکتر یا رشته بود char یا string می‌نوشتیم. zarb نیز نام تابع است، مثل getch یا circle. معمولاً نام تابع باید نمایانگر عملی باشد که آن تابع انجام می‌دهد. در ادامه و در داخل پرانتز لیست پارامترهای ورودی قرار دارد. دقت کنید قبل از **هر پارامتر ورودی** نوع آن مشخص می‌شود، حتی اگر کلیه‌ی پارامترهای ورودی از یک نوع باشند حق نداریم از نوع آن فاکتور بگیریم و مثلاً بنویسیم int zarb (int x,y)، چنین کاری صحیح نیست.

تابع با { شروع می‌شود و با } تمام می‌شود و متن تابع، بین این دو آکولاد نوشته می‌شود. برگرداندن خروجی نیز با کلمه‌ی کلیدی return انجام می‌شود که قبلاً در هر برنامه از آن استفاده کرده‌ایم. در main زمانی که return 0 اجرا می‌شود، عدد 0 به معنای اجرای

موفقیت‌آمیز برنامه به سیستم‌عامل برگردانده می‌شود. اما return در تابعی که نوشتیم به معنای برگرداندن خروجی تابع است (که در این‌جا ضرب دو پارامتر ورودی است). اما چطور می‌توان از تابعی که نوشتیم استفاده کرد؟ به سادگی! ابتدا به کد ۹-۲ توجه کنید:

کد ۹-۲

```
1  #include<iostream>
2  #include<conio.h>
3  using namespace std;
4  int zarb(int x, int y)
5  {
6     return x*y;
7  }
8  int main()
9  {
10    int i,j;
11    cout<<"Enter two numbers:";
12    cin>>i>>j;
13    cout<<i<<"*"<<j<<"="<<zarb(i,j);
14    getch();
15    return 0;
16 }
```

همان‌طور که در کد ۹-۲ مشاهده می‌شود، استفاده از توابعی که نوشتیم، درست همانند استفاده از توابع دیگر است. فقط باید حواسمان باشد چون برنامه خط به خط و از بالا به پایین اجرا می‌شود، اگر می‌خواهیم در main از تابعی استفاده کنیم باید آن را قبل از main تعریف کنیم.

۳. نوشتن تابع محاسبه‌ی n!

به عنوان مثال بعدی، فرض کنید می‌خواهیم تابعی بنویسیم که پارامتر ورودی n را گرفته و n! را محاسبه کرده و برگرداند. متن برنامه‌ای که این کار را انجام دهد در کد ۹-۳ آمده است.

کد ۹-۳

```
1  #include<iostream>
2  #include<conio.h>
3  using namespace std;
4  unsigned fact(unsigned n)
5  {
6     unsigned r=1,i;
7     for(i=2;i<=n;i++)
8        r*=i;
9     return r;
10 }
11 int main()
12 {
13    int n;
14    cout<<"Enter n:";
15    cin>>n;
16    cout<<n<<"!="<<fact(n);
17    getch();
18    return 0;
19 }
```

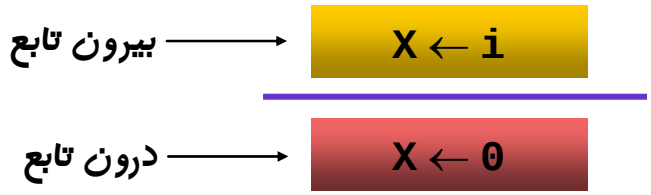
در این‌جا ذکر یک نکته ضروریست و آن این که در توابع مختلف، دنیای جداگانه‌ای وجود دارد! یعنی این که در هر تابعی (از جمله main که تابع اصلی هر برنامه است) هر متغیری با هر نامی که تعریف شود مربوط به همان تابع است و ربطی به متغیرهایی با اسامی مشابه ندارد. همچنین متغیرهایی که به عنوان ورودی به تابع می‌دهیم، اگر داخل تابع دچار تغییر شوند، انعکاسی بر روی خود پارامترهای ارسالی ندارند، زیرا کپی مقادیر متغیرهایی که به

عنوان پارامتر ورودی به تابع ارسال می‌شود به تابع فرستاده می‌شود و نه خودشان. برای فهم بهتر این قضیه به کد ۹-۴ دقت کنید:

کد ۹-۴

```
1  #include<iostream>
2  #include<conio.h>
3  using namespace std;
4  int f(int x)
5  {
6     x=0;
7     return x;
8  }
9  int main()
10 {
11    int i=10,j;
12    j=f(i);
13    cout<<"i="<<i<<" ,j="<<j;
14    getch();
15  return 0;
16 }
```

به نظر شما مقادیر i و j پس از اجرای برنامه چه خواهد بود؟ شاید انتظار داشته باشیم i و j هر دو صفر باشند. در مورد j انتظار ما صحیح است، زیرا x که پارامتر ورودی تابع است در ابتدای تابع صفر شده و سپس برگردانده می‌شود، پس خروجی تابع در هر حال صفر است. اما چرا i صفر نمی‌شود؟ دلیلش را توضیح دادیم، زیرا در واقع مقدار i که در این مثال برابر ۱۰ است، در متغیر x کپی می‌شود و سپس x برابر صفر می‌شود. پس i همچنان دست نخورده باقی خواهد ماند. در شکل ۹-۱ این موضوع نشان داده شده است.



شکل ۹-۱: عدم تغییر پارامتر ارسالی به تابع (کپی متغیر i در x قرار می‌گیرد و سپس x صفر می‌شود و بنابراین i تغییری نخواهد کرد)

۴. نوشتن توابع گرافیکی

حال فرض کنید می‌خواهیم تابعی بنویسیم که یک مستطیل را با رنگ و مختصات تصادفی بر روی صفحه گرافیک ترسیم کند. ابتدا به متن برنامه در کد ۹-۵ دقت کنید:

کد ۹-۵

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <graphics.h>
4  using namespace std;
5  void rand_rect()
6  {
7     int x1,y1,x2,y2,R,G,B;
8     x1=rand()%getmaxx();
9     y1=rand()%getmaxy();
10    x2=rand()%getmaxx();
11    y2=rand()%getmaxy();
12    R=rand()%256;
13    G=rand()%256;
14    B=rand()%256;
15    setcolor(COLOR(R,G,B));
16    rectangle(x1,y1,x2,y2);
17 }
18 int main()
19 {
20    srand(time(NULL));
21    initwindow(1280,800,"",-3,-3);
```

```
22      while(!kbhit())
23      {
24         rand_rect();
25         delay(100);
26      }
27      closegraph();
28      return 0;
29  }
```

همان‌طور که در کد ۹-۵ دیده می‌شود، برای تعریف تابعی که خروجی ندارد (یعنی مقداری را محاسبه نکرده و بر نمی‌گرداند) از کلمه‌ی کلیدی ***void*** استفاده می‌شود.

نکته‌ی دیگری که در کد ۹-۵ وجود دارد و مربوط به کلیه‌ی توابعی است که ترسیم‌های گرافیکی انجام می‌دهند، این است که به هیچ وجه در تابع مربوط به ترسیم‌های گرافیکی initwindow را اجرا نمی‌کنیم، زیرا initwindow باید یک بار در main و قبل از هرگونه ترسیم گرافیکی اجرا شود، استفاده‌ی چند باره‌ی initwindow باعث می‌شود نتیجه‌ی مورد نظر ما حاصل نشود (می‌توانید یک بار این کار را امتحان کنید!). همچنین تابع srand(time(NULL)) را (که برای راه‌اندازی تولید اعداد تصادفی است) نیز، تنها یک بار در ابتدای main() صدا می‌زنیم.

در کد ۹-۵ تا زمانی که کلیدی زده نشده با فواصل زمانی ۱۰۰ میلی ثانیه مستطیل‌های تصادفی بر روی صفحه ترسیم می‌شود.

۵. توابع گرافیکی با پارامتر ورودی

به عنوان مثال بعدی فرض کنید می‌خواهیم تابع ترسیم یک جدول m×n از k دایره‌ی متحدالمرکز با شعاع مضارب r را با گرفتن این ۴ پارامتر (m,n,k,r) بنویسیم (برای یادآوری می‌توانید به کد ۶-۱۱ و توضیحات آن رجوع کنید). البته می‌توانیم علاوه بر چهار پارامتر فوق، (x,y) گوشه‌ی بالا و چپ شروع این ترسیم‌ها را نیز به عنوان ورودی بگیریم تا ترسیم را همیشه از (0,0) شروع نکنیم. به این ترتیب نسخه‌ی کامل این تابع با ۶ پارامتر ورودی در کد ۶-۹ نوشته شده است.

کد ۶-۹

```
1  #include <iostream>
2  #include <graphics.h>
3  using namespace std;
4  void mycircles(int x0,int y0,int m,int n,int k,int r)
5  {
6     int i,j,x,y,t;
7     for(i=1;i<=m;i++)
8        for(j=1;j<=n;j++)
9        {
10          x=x0+(2*j-1)*k*r+(j-1)*r;
11          y=y0+(2*i-1)*k*r+(i-1)*r;
12          for(t=1;t<=k;t++)
13             circle(x,y,t*r);
14       }
15 }
16 int main()
17 {
18    int r,m,n,k,x,y;
19    cout<<"Enter m,n,k:";
20    cin>>m>>n>>k;
21    cout<<"Enter r:";
22    cin>>r;
23    cout<<"Enter x,y:";
24    cin>>x>>y;
25    initwindow(1280,800,"",-3,-3);
26    mycircles(x,y,m,n,k,r);
```

```
27     getch();
28   closegraph();
29   return 0;
30  }
```

۶. یک تابع ریاضی: بررسی اول بودن یک عدد

مثال بعدی مجدداً در مورد ریاضیات است، می‌خواهیم تابعی بنویسیم که اگر پارامتر ورودی آن، عددی اول بود true و در غیر این صورت false برگرداند. سپس از این تابع استفاده کرده و کلیه‌ی اعداد اول از ۲ تا n را (n ورودی کاربر است) چاپ می‌کنیم. برنامه‌ی مورد نظر در کد ۹-۷ نوشته شده است.

کد ۹-۷

```
1  #include<iostream>
2  #include<conio.h>
3  #include<math.h>
4  using namespace std;
5  bool isprime(int x)
6  {
7     int i;
8     bool p=true;
9     for(i=2;i<=sqrt(x);i++)
10       if(x%i==0)
11          p=false;
12    return p;
13 }
14 int main()
15 {
16    int i,n;
17    cout<<"Enter n:";
18    cin>>n;
19    for(i=2;i<=n;i++)
20       if(isprime(i))
21          cout<<i<<" is prime."<<endl;
```

```
22      getch();
23      return 0;
24  }
```

در این تابع isprime از نوع bool یا همان منطقی تعریف شده که یا مقدار درست یا نادرست برمی‌گرداند و می‌توان از آن در if استفاده کرد.

۷. فراخوانی با ارجاع در مقابل فراخوانی با مقدار (*)

تا کنون گفتیم که اگر پارامتر ارسالی به تابع را عوض کنیم، بر روی پارامتر قرار گرفته در تابع، در هنگام صدا زدن تابع، اثری ندارد. به این مدل، فراخوانی با مقدار[۲] می‌گویند، زیرا به جای پارامترهای تابع، می‌توان مستقیماً مقدار نوشت، مثلاً می‌توان تابع circle را هم با یکسری متغیر به صورت circle(x,y,r); صدا زد و هم با یکسری مقدار ثابت به صورت circle(100,100,50);. در مقابل، مدل دیگری نیز برای تعریف تابع وجود دارد که به آن فراخوانی با ارجاع[۳] می‌گویند و در آن مدل، می‌توان پارامترهای ورودی تابع را طوری تعریف کرد که *اثر تغییرات آن‌ها بر روی متغیرهای ارسال شده به تابع منعکس شود*. در ضمن بدیهی است در این مدل، به جای پارامترهایی که با ارجاع فراخوانی شده‌اند نمی‌توان عدد قرار داد. (چرا؟) به عنوان مثال به کد ۹-۸ دقت کنید که در آن پارامتر ارسالی به تابع صفر می‌شود و اثر آن نیز بر روی j منعکس می‌شود.

[2] Call by value
[3] Call by reference

کد ۹-۸

```
#include<iostream>
#include<conio.h>
using namespace std;
int f(int &x)
{
   x=0;
   return x;
}
int main()
{
   int i=10,j;
   j=f(i);
   cout<<"i="<<i<<" ,j="<<j;
   getch();
   return 0;
}
```

البته این مثال صرفاً جهت آشنایی با نحوه‌ی نوشتن این گونه توابع بوده است. همان‌طور که می‌بینید تنها در تعریف تابع تفاوت ایجاد شده نه در استفاده و صدا زدن آن.

اکنون می‌پردازیم به یک مثال کاربردی‌تر از فراخوانی با ارجاع، جابجایی محتوای دو متغیر یا swap. بدیهی است برای تعویض محتوای دو متغیر با یکدیگر، باید از فراخوانی با ارجاع استفاده کرد، زیرا در نهایت باید پارامترهای ارسالی به تابع، واقعاً عوض شوند. کد ۹-۹ نحوه‌ی نوشتن این تابع و استفاده از آن را نشان می‌دهد:

کد ۹-۹

```
1  #include<iostream>
2  #include<conio.h>
3  using namespace std;
4  void swap(int &x,int &y)
5  {
6     int t;
7     t=x;
8     x=y;
9     y=t;
10 }
11 int main()
12 {
13    int i,j;
14    cout<<"Enter i,j:";
15    cin>>i>>j;
16    cout<<"You have entered i="<<i<<" ,j="<<j<<endl;
17    swap(i,j);
18    cout<<"Now after swap i="<<i<<" ,j="<<j;
19    getch();
20    return 0;
21 }
```

شما می‌توانید توابع خود را در یک فایل با نام myheader.h نوشته و سپس آن را کنار فایل‌های برنامه‌ی خود و یا در فولدر include در Dev C++ کپی کنید. پس از آن می‌توانید با نوشتن #include<myheader.h> لیست توابعی که در آن نوشته‌اید را به لیست دستورات برنامه‌ی خود بیفزایید، یعنی برنامه‌ی شما توابعی که در آن نوشته‌اید را می‌شناسد. این فایل فقط باید شامل توابع باشد و احتیاجی به main ندارد.

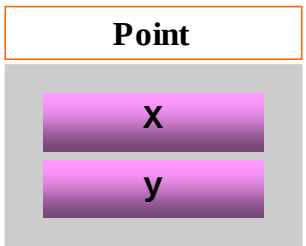
۱۰

فصل دهم
همه با هم

برنامه‌نویسی پیشرفته (*)

۱. ساختمان‌ها[1]

در این بخش، می‌پردازیم به نحوه‌ی ساخت انواع جدیدتر و پیچیده‌تر در زبان C. تا کنون هر نوع متغیر که تعریف می‌کردیم، یک عدد، حرف یا رشته بود، نظیر نوع int، double، char یا string. این نوع متغیرها، جزء انواع ساده محسوب می‌شوند، یعنی هرکدام، یک متغیر را نگه می‌دارند، به عنوان مثال، اگر بخواهیم مختصات یک نقطه را در صفحه نگه داریم، باید دو متغیر تعریف کنیم: یکی x برای طول نقطه و دیگری y برای عرض نقطه. به عبارت دیگر خطوط نقطه به عنوان یک نوع از قبل در زبان C وجود ندارند. اما این امکان در زبان C وجود دارد تا بتوان با ترکیب متغیرهای ساده، متغیرهایی جدید ساخت که شامل چند متغیر باشند. به این متغیرهایی که می‌سازیم اصطلاحاً ساختمان می‌گویند که در زبان C با کلمه‌ی کلیدی struct تعریف می‌شود. به عنوان مثال فرض کنید می‌خواهیم نقطه را به عنوان نوع جدید معرفی کنیم، به این ترتیب کافیست همانند کد ۱۰-۱، داخل struct، یک x و y تعریف کنیم، به این معنا که این نوع جدید، خود شامل دو متغیر صحیح به نام‌های x و y است (شکل ۱۰-۱). فقط به یک استثنا باید دقت کنید و آن این که پس از علامت { یک ; می‌آید، که تا به حال در جای دیگر چنین طرز نوشتاری نداشتیم.



شکل ۱۰-۱: طرحواره‌ی یک ساختمان نقطه، حاوی مختصات x و y

[1] Structures

کد ۱۰-۱

```
1  struct Point
2  {
3     int x;
4     int y;
5  };
```

همان‌طور که در کد ۱۰-۱ دیده می‌شود، نحوه‌ی تعریف متغیر از جنس Point که خودمان آن را تعریف کردیم، به سادگیِ تعریف متغیرهایی نظیر int و char است، برای دست‌یابی به اطلاعات درون یک ساختمان نیز از نقطه استفاده می‌شود. به این ترتیب، یک متغیر جدید تعریف کردیم. حال یک گام دیگر به جلو برداشته و تابعی تعریف می‌کنیم که دو متغیر از نوع Point به عنوان ورودی بگیرد و سپس یک خط بین آن‌ها ترسیم کند. چنین تابعی در کد ۱۰-۲ نوشته شده است.

کد ۱۰-۲

```
1  #include<iostream>
2  #include <graphics.h>
3  using namespace std;
4  struct Point
5  {
6     int x;
7     int y;
8  };
9
10 void MyLine(Point p1, Point p2)
11 {
12    line(p1.x, p1.y, p2.x, p2.y);
13 }
14 int main()
15 {
16    Point p1, p2;
17    cout<<"Enter x1,y1:";
```

```
18      cin>>p1.x>>p1.y;
19      cout<<"Enter x2,y2:";
20      cin>>p2.x>>p2.y;
21      initwindow(1280,800,"",-3,-3);
22      MyLine(p1,p2);
23      getch();
24      closegraph();
25      return 0;
26  }
```

۲. ساختمان‌ها به عنوان پارامتر ورودی تابع

همان‌طور که دیده می‌شود چون نوع Point نوع جدیدی است که ما خودمان تعریف کردیم، نمی‌توان مستقیماً آن را cin کرد، بلکه باید x و y آن را به صورت جداگانه cin کرد. اما می‌توان برای راحتی تابعی نوشت که یک Point (به صورت فراخوانی با ارجاع) گرفته و x و y آن را از کاربر بگیرد. به این ترتیب می‌توانیم کد ۱۰-۲ را بازنویسی کنیم تا کد ۱۰-۳ حاصل شود.

کد ۱۰-۳

```
1   #include<iostream>
2   #include <graphics.h>
3   using namespace std;
4   struct Point
5   {
6      int x;
7      int y;
8   };
9   void getPoint(Point &p)
10  {
11     cout<<"Enter x:";
12     cin>>p.x;
13     cout<<"Enter y:";
14     cin>>p.y;
```

```
15  }
16  void MyLine(Point p1, Point p2)
17  {
18     line(p1.x, p1.y, p2.x, p2.y);
19  }
20  int main()
21  {
22     Point p1, p2;
23     cout<<"Enter Point 1:"<<endl;
24     getPoint(p1);
25     cout<<"Enter Point 2:"<<endl;
26     getPoint(p2);
27     initwindow(1280,800,"",-3,-3);
28     MyLine(p1,p2);
29     getch();
30     closegraph();
31     return 0;
32  }
```

همان‌طور که در کد ۱۰-۳ می‌بینید، یک محدودیت اصلی این است که تابع getPoint تنها یک نقطه را به عنوان ورودی می‌پذیرد و بنابراین به تعداد نقطه‌هایی که می‌خواهیم بگیریم باید این تابع صدا زده شود. در ادامه‌ی این فصل خواهیم آموخت چگونه توابعی بنویسیم که به تعداد دلخواه و از پیش نامعلوم ورودی داشته باشند.

اکنون برای این که نشان بدهیم چگونه می‌توان برنامه‌های وسیع‌تر را با استفاده از توابع مختلفی که می‌نویسیم ساده‌تر و بهتر نوشت، برنامه‌ی قبلی را این طور گسترش می‌دهیم: می‌خواهیم تابعی بنویسیم که به عنوان ورودی یک آرایه از Point ها و سپس تعداد آن‌ها را گرفته و یک چند ضلعی ترسیم کنیم. برای گرفتن مختصات نقطه‌ها از getPoint استفاده می‌کنیم و برای ترسیم اضلاع چند ضلعی از تابع MyLine که نوشتیم. این برنامه و توابع مورد نیاز آن در کد ۱۰-۴ نوشته شده است:

کد ۱۰-۴

```
#include<iostream>
#include <graphics.h>
using namespace std;
struct Point
{
   int x;
   int y;
};
void getPoint(Point &p)
{
   cout<<"Enter x:";
   cin>>p.x;
   cout<<"Enter y:";
   cin>>p.y;
}
void MyLine(Point p1, Point p2)
{
   line(p1.x, p1.y, p2.x, p2.y);
}
void MyPolygon(Point p[], int n)
{
   int i;
   for(i=0;i<n-1;i++)
      MyLine(p[i],p[i+1]);
   MyLine(p[0],p[n-1]);
}
int main()
{
   int i,n;
   Point p[100];
   cout<<"Enter n:";
   cin>>n;
   for(i=0;i<n;i++)
   {
      cout<<"Enter Point "<<i+1<<":"<<endl;
      getPoint(p[i]);
   }
   initwindow(1280,800,"",-3,-3);
   MyPolygon(p,n);
   getch();
   closegraph();
   return 0;
```

```
43 }
```

۳. ساختمان‌ها به عنوان خروجی تابع

توابعی که تا به حال نوشتیم همگی بدون خروجی (یا همان void) بودند، اکنون می‌خواهیم تابعی بنویسیم که دو ورودی Point داشته باشد و خروجی آن نیز از نوع Point باشد. این تابع که نام آن را MidPoint می‌گذاریم، وسط پاره‌خط AB را محاسبه می‌کند و A و B نیز دو نقطه‌ی ورودی آن، یعنی دو سر پاره‌خط هستند. سپس با استفاده از این تابع و توابع MyLine و MyPolygon، یک مثلث را به همراه میانه‌هایش ترسیم می‌کنیم. به نحوه‌ی به کار گیری MidPoint در تابع MyLine دقت کنید، این که چطور و بدون قرار دادن خروجی MidPoint در متغیری از نوع Point، صرفاً به صورت موقت از آن استفاده می‌کنیم. متن این برنامه در کد ۱۰-۵ آمده است.

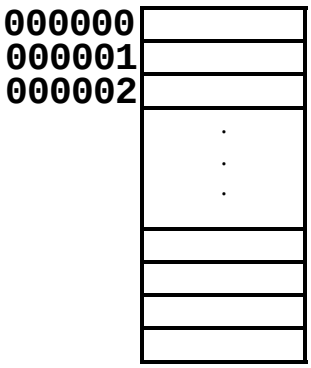
کد ۱۰-۵

```
1  #include<iostream>
2  #include <graphics.h>
3  using namespace std;
4  struct Point
5  {
6     int x;
7     int y;
8  };
9  void getPoint(Point &p)
10 {
11    cout<<"Enter x:";
12    cin>>p.x;
13    cout<<"Enter y:";
14    cin>>p.y;
15 }
```

```
16 void MyLine(Point p1, Point p2)
17 {
18    line(p1.x, p1.y, p2.x, p2.y);
19 }
20 void MyPolygon(Point p[], int n)
21 {
22    int i;
23    for(i=0;i<n-1;i++)
24       MyLine(p[i],p[i+1]);
25    MyLine(p[0],p[n-1]);
26 }
27 Point MidPoint(Point A, Point B)
28 {
29    Point r;
30    r.x=(A.x+B.x)/2;
31    r.y=(A.y+B.y)/2;
32    return r;
33 }
34 int main()
35 {
36    int i;
37    Point p[3];
38    cout<<"Enter 3 Points of a triangle:"<<endl;
39    for(i=0;i<3;i++)
40    {
41       cout<<"Enter Point "<<i+1<<":"<<endl;
42       getPoint(p[i]);
43    }
44    initwindow(1280,800,"",-3,-3);
45    MyPolygon(p,3);
46    setcolor(4) ;
47    MyLine(p[0],MidPoint(p[1],p[2]));
48    MyLine(p[1],MidPoint(p[0],p[2]));
49    MyLine(p[2],MidPoint(p[1],p[0]));
50    getch();
51    closegraph();
52    return 0;
53 }
```

۴. اشاره‌گر[۲] ها

در مورد حافظه‌ی کامپیوتر و انواع آن صحبت کرده‌ایم و گفتیم RAM حافظه‌ی اصلی کامپیوتر است که برنامه‌ها بر روی آن اجرا می‌شوند. می‌توانیم حافظه‌ی کامپیوتر را به صورت نواری فرض کنیم که هر خانه‌ی آن یک آدرس منحصر به فرد دارد (شکل ۱۰-۲).



شکل ۱۰-۲: مدل حافظه و آدرس‌دهی خانه‌های آن

چنین تصوری از حافظه، تصور معقولی است، زیرا به هر حال خانه‌های حافظه باید آدرسی یکتا داشته باشند که به وسیله‌ی آن بتوان به آن‌ها دسترسی داشت و مقداری بر روی آن‌ها نوشت یا مقداری را خواند. در عمل نیز واقعاً چنین است، یعنی تک تک خانه‌های حافظه‌ی شماره‌ای منحصر به فرد دارند، مثلاً وقتی می‌گوییم خانه‌ی ۱۰۰۰ حافظه، منظور یک خانه‌ی خاص است که با شمارش از اول حافظه تا ۱۰۰۰ به آن می‌رسیم. کلیه‌ی متغیرهایی که تا الآن تعریف می‌کردیم نیز در این قانون صدق می‌کنند، یعنی زمانی که می‌نویسیم int x,y;، در هنگام اجرای برنامه، به هر کدام از x و y، خانه‌هایی از حافظه با آدرس مشخص نسبت داده می‌شود.

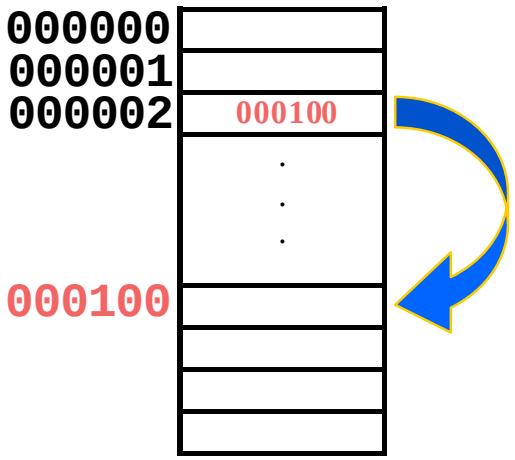
[2] Pointer

مثلاً x خانه‌ی ۱۰۰۰ و y خانه‌ی ۱۰۰۴ می‌شوند. اما چطور می‌توان آدرس متغیرها را استخراج کرد؟ به سادگی! کافی است تا از عملگر & قبل از متغیر استفاده شود، به این ترتیب می‌توان آدرس یک متغیر را با دستوری شبیه زیر چاپ کرد:

```
Cout<<"Addness of x is: "<<&x<<endl;
```

البته اگر چنین دستوری بنویسید و اجرا کنید، عددی که می‌بینید کمی عجیب و غریب است، مثلاً به صورت 0x22ff4c، منظور از 0x که در ابتدای عدد آمده، این است که عدد، یک عدد مبنای ۱۶ است و عددی که به دنبال آن می‌آید، همان آدرس متغیر است.

در زبان C می‌توان متغیرهایی از جنس آدرس یک متغیر نیز داشت، یعنی متغیرهایی که به جای یک عدد صحیح یا یک کاراکتر، آدرس یک عدد صحیح یا یک کاراکتر را نگه دارند. به چنین متغیرهایی اشاره‌گر می‌گویند، زیرا محتوای آن‌ها و مقدارشان، در اصل آدرس یک متغیر دیگر است و یا به بیان دیگر به یک متغیر دیگر اشاره می‌کند (شکل ۱۰-۳).



شکل ۱۰-۳: نگهداری آدرس یک متغیر و مفهوم اشاره‌گر

نحوه‌ی تعریف اشاره‌گر به یک نوع نیز ساده است، کافی است قبل از نام متغیری که می‌خواهیم اشاره‌گر باشد یک * قرار دهیم. به عنوان مثال در خط زیر x یک متغیر از نوع int و px یک متغیر از نوع اشاره‌گر به int است.

```
int x,*px;
```

px را نمی‌توان برابر عددی قرار داد (مگر آن که مطمئن باشیم آدرس یک int است که مجاز به دسترسی به آن هستیم)، بلکه می‌توان آن را برابر آدرس یک متغیر int قرار داد، مثلاً می‌توانیم بنویسیم:

```
px=&x;
```

به این ترتیب، px برابر آدرس x می‌شود، یعنی به آن اشاره می‌کند. حال با استفاده از px که آدرس x است، می‌توان محتوای x را نیز عوض کرد، به عنوان مثال می‌توانیم بنویسیم:

```
*px=10;
```

به این ترتیب، محتوای x برابر ۱۰ می‌شود (می‌توان با چاپ x پس از این دستور این واقعیت را چک کرد). پس px*، درست مانند یک عدد صحیح است، عدد صحیحی که آدرس آن در px است. مثلاً می‌توان نوشت ;++(px*) و به این ترتیب یک واحد به x افزوده می‌شود.

پرانتز گذاری هنگام کار با اشاره‌گرها بسیار حائز اهمیت است. مثلاً در همین مثال قبل اگر پرانتز را برداریم، عمل خواسته شده انجام نمی‌شود، بلکه px، یک واحد اضافه شده و به عدد صحیح بعد از x (که لزوماً معتبر نیست) اشاره می‌کند.

جمع و تفریق در اشاره‌گرها به معنای چند خانه جلوتر یا عقب‌تر است، مثلاً px+10، به 10 عدد صحیح بعد از x اشاره می‌کند یا px-2 به دو خانه قبل از x اشاره دارد.

۵. اخذ حافظه‌ی پویا از سیستم عامل

با اشاره‌گرها کارهای زیادی می‌توان انجام داد، مثلاً می‌توان یک متغیر جدید از سیستم‌عامل گرفت، مثلاً برای گرفتن یک عدد صحیح جدید در هنگام اجرای برنامه می‌توان نوشت:

```
px=new int;
```

به این ترتیب یک خانه‌ی حافظه‌ی جدید ایجاد می‌شود و آدرس آن در px قرار می‌گیرد که با px* می‌توان به محتوای آن دسترسی داشت. البته باید حواسمان باشد در انتها خانه‌ی ایجاد شده را پاک کنیم:

```
delete px;
```

عدم حذف حافظه‌هایی که به صورت پویا (dynamic) و در زمان اجرای برنامه (نه قبل از آن) از سیستم‌عامل اخذ شده‌اند، ممکن است باعث بروز مشکل نشتی حافظه (Memory Leakage) و عدم کارکرد درست برنامه یا کند شدن آن شود.

معمولاً ایجاد کردن یک عدد صحیح در هنگام اجرا موضوعیتی ندارد و معمولاً به دنبال ایجاد آرایه‌هایی با طول مختلف در برنامه هستیم. این کار نیز به راحتی امکان‌پذیر است، کافیست تعداد خانه‌هایی که می‌خواهیم اجرا کنیم را در [ ] بنویسیم:

```
px=new int[10];
```

در این حالت یک آرایه به وجود می‌آید که آدرس اولین خانه‌ی آن در px قرار گرفته، یعنی*px اولین خانه‌ی آن، *(px+1) دومین خانه‌ی آن و *(px+9) آخرین خانه‌ی آن است. شاید کار کردن با این طرز نوشتار راحت نباشد، جالب است بدانید در زبان C، *(px+i) با px[i] معادلند! یعنی هر دو یک مفهوم دارند و می‌توان آن‌ها را بدون هیچ اشکالی به جای یکدیگر به کار برد. برای امتحان می‌توانید یک آرایه‌ی عادی از int تعریف کنید و سپس به جای a[i] از *(a+i) استفاده کنید و ببینید که دقیقاً معادل هم عمل می‌کنند. برای درک بهتر، کد ۱۰-۶ را ببینید:

کد ۱۰-۶

```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
   int a[10],i;
   for(i=0;i<10;i++)
      a[i]=i;
   for(i=0;i<10;i++)
      cout<<*(a+i)<<endl;
   getch();
   return 0;
}
```

به این ترتیب، می‌توانیم آرایه‌ای با طول دلخواه در برنامه ایجاد کرده و از آن استفاده کنیم. به عنوان مثال در کد ۱۰-۷ برنامه‌ای نوشته شده که ابتدا n را از کاربر پرسیده و سپس یک آرایه‌ی n تایی از اعداد صحیح ایجاد می‌کند و n عدد از کاربر گرفته و در آن ذخیره می‌کند. سپس یکبار همه‌ی آن‌ها را با هم جمع کرده و حاصل جمع اعضای آرایه را چاپ می‌کند و در انتها نیز حافظه‌ی اخذ شده از سیستم آزاد می‌شود، فقط باید دقت داشت که چون آرایه از سیستم‌عامل گرفته‌ایم باید با دستور delete[] حافظه‌ی آن را آزاد کنیم نه delete.

کد ۱۰-۷

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  int main()
5  {
6     int *p,n,i,S=0;
7     cout<<"Enter n:";
8     cin>>n;
9     p=new int[n];
10    for(i=0;i<n;i++)
11    {
12       cout<<"Enter p"<<i+1<<":";
13       cin>>p[i];
14    }
15    for(i=0;i<10;i++)
16       S+=p[i];
17    cout<<"Sum="<<S;
18    delete[] p;
19    getch();
20    return 0;
21 }
```

۶. اشاره‌گرها و توابع و نوشتن توابعی با تعداد پارامتر ورودی نامشخص

پس تا به حال یک کاربرد اشاره‌گرها که اخذ حافظه‌ی پویا بود را با هم دیدیم. اکنون می‌خواهیم کاربرد اشاره‌گرها را در نوشتن توابع ببینیم. در فصل ۹ در مورد فراخوانی با ارجاع و فراخوانی با مقدار صحبت کرده‌ایم، حال فرض کنید می‌خواهیم یک اشاره‌گر را به تابع بفرستیم، به این ترتیب، اگر آن اشاره‌گر حاوی آدرس متغیری خاص باشد، حتی اگر فراخوانی با مقدار باشد نیز، چون به آدرس آن متغیر دسترسی داریم، می‌توانیم محتوای آن را عوض کنیم. برای درک بهتر مطلب گفته شده کد ۱۰-۸ را ببینید.

کد ۱۰-۸

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  void f(int *x)
5  {
6     *x=0;
7  }
8  int main()
9  {
10    int a=10;
11    cout<<"a="<<a<<endl;
12    f(&a);
13    cout<<"a="<<a<<endl;
14    getch();
15    return 0;
16 }
```

پس از اجرای این کد باید متوجه شده باشید منظور از بیان این مطلب چه بوده است، جالب است بدانید که قبلاً در زبان C، فراخوانی با ارجاع را با این مدل می‌نوشتند، یعنی آدرس متغیر مورد نظر را با عملگر & به تابع ارسال می‌کردند و در تابع نیز با عملگر * به محتوای عملگر

مورد نظر دسترسی پیدا می‌کردند. اکنون باید فهمیده باشید که چرا هنگامی که آرایه به تابع ارسال می‌شود، تغییرات بر روی آن ماندگار است (با توجه به معادل بودن a[i] با (a+i)*).

اکنون که این نکته را دریافتیم، می‌پردازیم به کاری مهم‌تر و آن نوشتن تابع getAllPoints است. در ابتدای این فصل که ساختمان Point را تعریف کردیم، تابع getPoint را نیز برای گرفتن یک نقطه نوشتیم، اما الان می‌خواهیم آن تابع را طوری گسترش دهیم که هر تعداد نقطه که خواستیم به عنوان پارامتر ورودی بگیرد و با ورودی گرفتن از کاربر، آن‌ها را مقداردهی کند. مثلاً اگر می‌خواهیم یک نقطه را از کاربر بگیریم بنویسیم:

```
getAllPoints(1,&p);
```

که 1 که پارامتر اول تابع است، تعداد نقاط را مشخص می‌کند، p نیز متغیری از نوع Point است که چون می‌خواستیم مقدار آن عوض شود، آدرس آن یعنی &p را به تابع ارسال کرده‌ایم. اگر بخواهیم سه نقطه را با این تابع از کاربر بگیریم باید بنویسیم:

```
getAllPoints(3,&p1,&p2,&p3);
```

یعنی ابتدا تعداد پارامترها و سپس آدرس تک تک متغیرهایی از نوع Point که باید از کاربر گرفته شوند ارسال می‌کنیم. در ظاهر این کار غیر ممکن به نظر می‌رسد، زیرا تا به حال هر تابعی که تعریف کردیم، تعداد متغیرهای ورودی آن مشخص بوده است. اما این کار امکان‌پذیر است! برای انجام این کار باید<stdrag.h> را #include کنیم تا بتوانیم از دستورات مربوط به لیست پارامترهای متغیر تابع استفاده کنیم. برای تعریف تعداد پارامترهای ورودی قابل تغییر، باید حداقل یک متغیر را تعریف کنیم و برای ادامه‌ی لیست پارامترها که نمی‌دانیم چند تاست، سه نقطه (...) بگذاریم. در ابتدا بد نیست به تابعی که یکسری ورودی گرفته و میانگین آن‌ها را حساب می‌کند، توجه کنید:

کد ۱۰-۹

```
1  #include <iostream>
2  #include <conio.h>
3  #include <stdarg.h>
4  using namespace std;
5  double average(int Count, ... )
6  {
7     va_list Numbers;
8     va_start(Numbers, Count);
9     int Sum = 0, i;
10    for(i=0; i<Count; i++)
11       Sum += va_arg(Numbers, int);
12    va_end(Numbers);
13    if(Count>0)
14       return (double(Sum)/Count);
15    else
16       return 0;
17 }
18 int main()
19 {
20    int a=1, b=2, c=7;
21 cout<<"("<<a<<"+"<<b<<"+"<<c<<")/3="<<average(3,a,b,c);
22    getch();
23    return 0;
24 }
```

همان‌طور که در کد ۱۰-۹ دیده می‌شود، ابتدا باید لیست پارامترهای ورودی را تعریف کنیم (که در این‌جا نام آن Numbers است)، سپس با شروع از اولین متغیر که نام آن Count است، لیست را آماده‌ی برداشتن سایر متغیرها کنیم (با دستور va_start(Numbers,Count)). سپس در هر مرحله یک متغیر از لیست پارامترهای ورودی برمی‌داریم (با دستور va_arg(Numbers,int)) که نوع این متغیر (در این‌جا int) نیز به

عنوان پارامتر دوم دستور va_arg مشخص می‌شود. در انتها نیز با دستور va_end، برداشتن متغیر از لیست پارامترها را متوقف می‌کنیم.

حال که این مثال را دیدیم، نوشتن تابع getAllPoints و استفاده از آن نباید سخت باشد. ابتدا کد ۱۰-۱۰ را ببینید:

کد ۱۰-۱۰

```
#include <iostream>
#include <graphics.h>
#include <stdarg.h>
using namespace std;
struct Point
{
   int x;
   int y;
};
void MyLine(Point p1, Point p2)
{
   line(p1.x, p1.y, p2.x, p2.y);
}
void getAllPoints(int Count, ... )
{
   Point *P;
   va_list Numbers;
   va_start(Numbers, Count);
   int i;
   for(i=0; i<Count; i++)
   {
      P = va_arg(Numbers, Point*);
      cout<<"Enter Point "<<i+1<<":"<<endl;
      cout<<"Enter x:";
      cin>>P->x;
      cout<<"Enter y:";
      cin>>P->y;
   }
   va_end(Numbers);
```

```
30  }
31
32  int main()
33  {
34     Point p1,p2,p3;
35     getAllPoints(3,&p1,&p2,&p3);
36     initwindow(1280,800,"",-3,-3);
37     MyLine(p1,p2);
38     MyLine(p2,p3);
39     MyLine(p3,p1);
40     getch();
41     closegraph();
42     return 0;
43  }
```

یک نکته‌ی جدید در تابع getAllPoints وجود دارد و آن این که: چنانچه یک اشاره‌گر به ساختمان نظیر p داشته باشیم و بخواهیم به عناصر داخلی آن دسترسی داشته باشیم، می‌توانیم به جای (*p).x از p->x استفاده کنیم و این طرز نوشتن بسیار متداول‌تر و مرسوم‌تر است. به این ترتیب می‌توانیم هر تعداد نقطه که خواستیم از کاربر بگیریم و محدودیتی در این زمینه وجود ندارد. البته شاید به نظر برسد که با آرایه راحت‌تر بتوان این کار را انجام داد، اما باید در نظر داشت در بسیاری از مواقع ما از آرایه‌ها استفاده نمی‌کنیم، مثلاً می‌خواهیم دو نقطه‌ی ابتدا و انتهای یک خط یا سه نقطه‌ی مربوط به سر رأس یک مثلث را از کاربر بگیریم که با این نوع نوشتن معمولاً راحت‌تر خواهد بود. در هر صورت این مدل تعریف تابع نیز جزئی از امکانات مفید زبان C محسوب می‌شود.

۷. برنامه‌نویسی بازگشتی

در ریاضیات، هر مفهومی را که بتوان به نحوی بر اساس خودش توضیح داد یا هر رابطه‌ای را که به تعریف خود بازگشت داشته باشد، مفهوم یا رابطه‌ی بازگشتی می‌گویند. به عنوان مثال، فاکتوریل در ریاضیات به صورت بازگشتی این‌چنین تعریف می‌شود:

$$\begin{cases} n!=n(n-1)! \\ 1!=1 \end{cases}$$

این یعنی این که فاکتوریل را با استفاده از مفهوم خود فاکتوریل تعریف کنیم. اما توجه به یک نکته بسیار ضروریست و آن این که در این‌گونه موارد حتماً به یک شرط اولیه نیاز است، زیرا در غیر این صورت، این قدر از مقدار n کم می‌شود تا به منفی بی‌نهایت میل کند! اما وجود شرط اولیه‌ی $1!=1$ راه را بر این موضوع می‌بندد و اجازه نمی‌دهد کم کردن از n تا ابد ادامه یابد. بد نیست برای فهم بهتر در این زمینه مثالی بزنیم: فرض کنید می‌خواهیم ۴! را حساب کنیم، مطابق تعریف:

$$۴!=۴\times۳!$$

اما ۳! نیز خود مقدار معلومی ندارد و می‌توانیم بنویسیم:

$$۳!=۳\times۲!$$

و اگر ادامه بدهیم داریم:

$$۲!=۲\times۱!$$

اما در مورد ۱! دیگر احتیاجی به کاهش مقدار نداریم، بلکه ۱! بنا به **تعریف اولیه** برابر ۱ است، پس $۲!=۲\times۱=۲$ و بنابراین $۳!=۳\times۲=۶$ و همچنین $۴!=۴\times۶=۲۴$. به این ترتیب با استفاده از مفهوم بازگشتی، ۴! محاسبه می‌شود.

برای نوشتن تابع بازگشتی در زبان C نیز، کافیست تا همین مفاهیم را بنویسیم. به عنوان مثال تابع بازگشتی محاسبه‌ی فاکتوریل در کد ۱۰-۱۱ نوشته شده است.

کد ۱۰-۱۱

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  unsigned fact(unsigned n)
5  {
6     if(n>1)
7        return n*fact(n-1);
8     else
9        return 1;
10 }
11 int main()
12 {
13    int n;
14    cout<<"Enter n:";
15    cin>>n;
16    cout<<n<<"!="<<fact(n);
17    getch();
18    return 0;
19 }
```

if های به کار رفته در تابع بازگشتی محاسبه‌ی فاکتوریل در کد ۱۰-۱۱، برای آزمودن این است که آیا به شرط اولیه رسیده‌ایم یا خیر؟ چنانچه این if ها وجود نداشته باشند، تابع آنقدر خودش را فراخوانی می‌کند تا حافظه سرریز[۳] شود و برنامه با خطا شکسته شود و به اتمام برسد.

[3] Overflow

برنامه‌ی بعدی که می‌خواهیم به صورت بازگشتی بنویسیم، محاسبه‌ی n اُمین عدد دنباله‌ی فیبوناچی است. می‌دانیم در دنباله‌ی فیبوناچی:

$$a_n=a_{n-1}+a_{n-2}\ ,\ a_1=a_2=1$$

بنابراین تابع بازگشتی این رابطه را می‌توان به راحتی نوشت (کد ۱۰-۱۲).

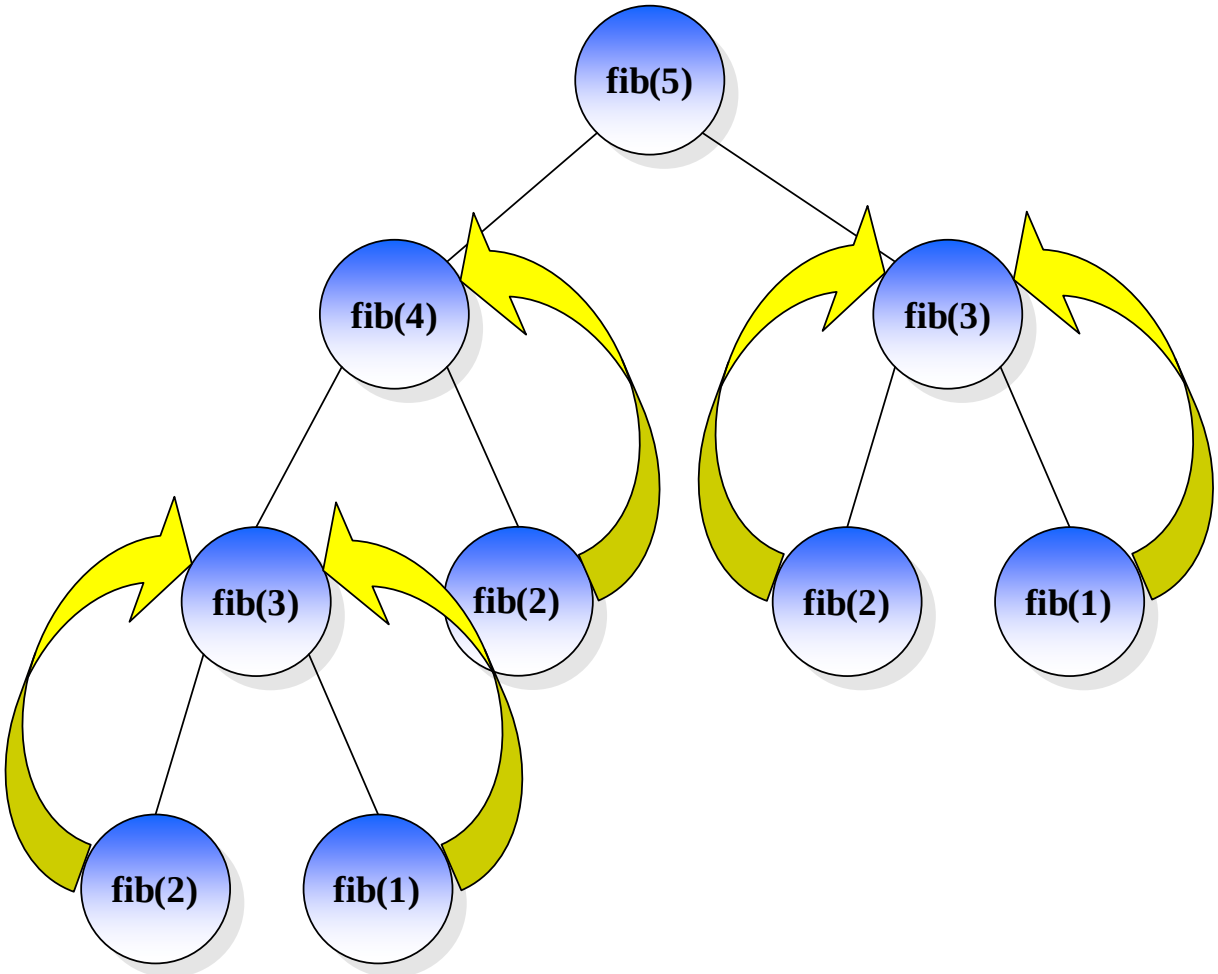
کد ۱۰-۱۲

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4  unsigned fib(unsigned n)
5  {
6     if(n>2)
7        return fib(n-1)+fib(n-2);
8     else
9        return 1;
10 }
11 int main()
12 {
13    int n;
14    cout<<"Enter n:";
15    cin>>n;
16    cout<<"fib("<<n<<")="<<fib(n);
17    getch();
18    return 0;
19 }
```

برای به دست آوردن شهود بیشتر نسبت به نحوه‌ی اجرای توابع بازگشتی، درختی موسوم به درخت بازگشت برای تابع فیبوناچی نوشته شده در کد ۱۰-۱۲ به ازای n=4 در شکل ۱۰-۱۲ رسم شده است. این درخت نشان می‌دهد برای محاسبه‌ی fib(5) به محاسبه‌ی fib(4) و fib(3) نیاز است و برای محاسبه‌ی fib(4) به fib(3) و fib(2) و برای محاسبه‌ی fib(3) به fib(2) و fib(1) و چون fib(1) و fib(2) جزء شرط اولیه محسوب می‌شوند، مستقیماً برابر ۱

محاسبه می‌شوند و از روی آن‌ها بقیه‌ی مقادیر نیز به دست می‌آید (فلش‌های زرد رنگ در شکل ۱۰-۱۲).

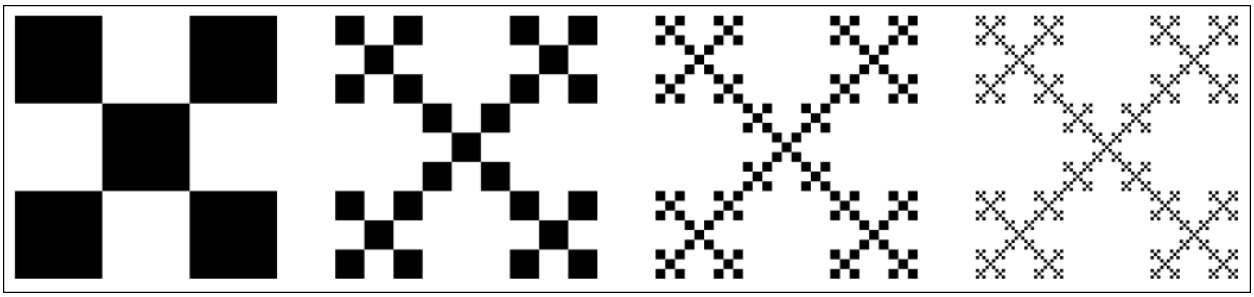
البته همان‌طور که دیده می‌شود، به عنوان مثال fib(2)، سه بار محاسبه می‌شود. این یعنی افزونگی[۴] در محاسبات، یعنی یک محاسبات را چند بار انجام دادن، که طبیعتاً کار درستی نیست و از کارایی برنامه می‌کاهد. این مثال نشان می‌دهد لزوماً بازگشتی نوشتن هر برنامه‌ای که در رابطه‌ی بازگشتی می‌گنجد، کار درستی نیست.



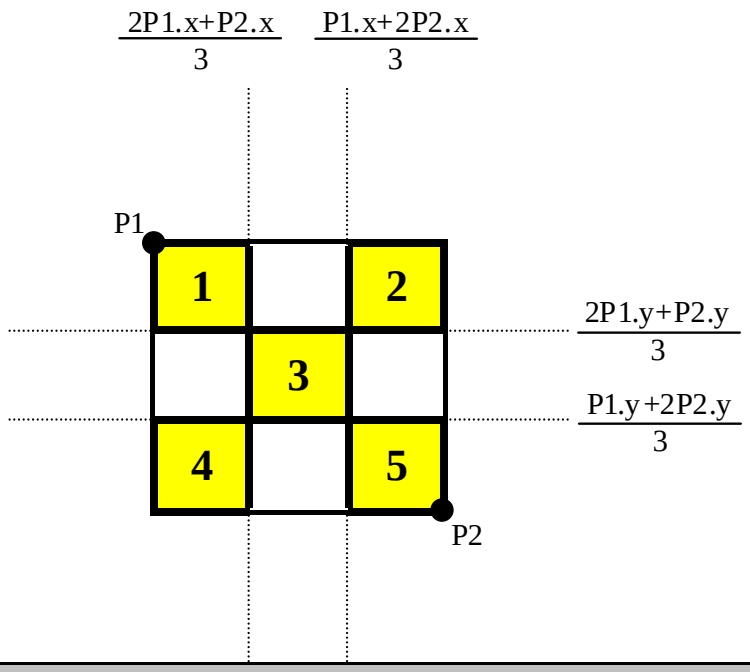
شکل ۱۰-۱۲: درخت مراحل مختلف صدا زدن تابع بازگشتی فیبوناچی (فلش‌های زرد رنگ نشان می‌دهد که به شرط اولیه رسیده‌ایم و عدد ۱ مستقیماً برگردانده می‌شود)

[4] Redundancy

در ادامه و برای آخرین مثال می‌پردازیم به مثال‌های گرافیکی تابع بازگشتی، فرض کنید می‌خواهیم شکلی ایجاد کنیم که به ازای n های مختلف، همانند شکل ۱۰-۱۳ باشد. ایجاد چنین شکلی با نوشتن تابع بازگشتی امکان‌پذیر است. کافیست ابتدا چک کنیم n (ورودی اول تابع) صفر است یا خیر؟ اگر صفر بود که تنها باید یک مستطیل توپُر که با مختصات P1 و P2 بنا می‌شود رسم کنیم (P1 و P2 مختصات نقاط قطر مشخص کننده‌ی مربع هستند). اما اگر n صفر نبود، باید تابع را پنج بار دیگر به ازای n-1 و مختصات مشخص شده در شکل ۱۰-۱۴ صدا بزنیم.



شکل ۱۰-۱۳: یک شکل خود متشابه به ازای n های مختلف. این شکل را می‌توان به صورت بازگشتی ترسیم کرد



شکل ۱۰-۱۴: نمایش نحوه‌ی تقسیم شکل برای فراخوانی تابع به صورت بازگشتی (مربع‌های زرد رنگ، فراخوانی‌های تابع را به ازای n-1 نشان می‌دهند و برای مختصات دو رأس آن باید به محل تلاقی خطوط یک سوم طول و عرض توجه کرد)

به این ترتیب شکل مورد نظر ترسیم خواهد شد. متن تابع و برنامه‌ی مورد نظر در کد ۱۰-۱۳ آمده است.

کد ۱۰-۱۳

```
1  #include <iostream>
2  #include <graphics.h>
3  using namespace std;
4  struct Point
5  {
6     int x;
7     int y;
8  };
9  void getPoint(Point &p)
10 {
11    cout<<"Enter x:";
12    cin>>p.x;
```

```
13      cout<<"Enter y:";
14      cin>>p.y;
15  }
16  void fractal(int n, Point P1, Point P2)
17  {
18      if(n<=0)
19      {
20          bar(P1.x,P1.y,P2.x,P2.y);
21      }
22      else
23      {
24          Point Q1,Q2;
25          Q1.x=P1.x;
26          Q1.y=P1.y;
27          Q2.x=(2*P1.x+P2.x)/3;
28          Q2.y=(2*P1.y+P2.y)/3;
29          fractal(n-1,Q1,Q2);
30          Q1.x=(P1.x+2*P2.x)/3;
31          Q1.y=P1.y;
32          Q2.x=P2.x;
33          Q2.y=(2*P1.y+P2.y)/3;
34          fractal(n-1,Q1,Q2);
35          Q1.x=(2*P1.x+P2.x)/3;
36          Q1.y=(2*P1.y+P2.y)/3;
37          Q2.x=(P1.x+2*P2.x)/3;
38          Q2.y=(P1.y+2*P2.y)/3;
39          fractal(n-1,Q1,Q2);
40          Q1.x=P1.x;
41          Q1.y=(P1.y+2*P2.y)/3;
42          Q2.x=(2*P1.x+P2.x)/3;
43          Q2.y=P2.y;
44          fractal(n-1,Q1,Q2);
45          Q1.x=(P1.x+2*P2.x)/3;
46          Q1.y=(P1.y+2*P2.y)/3;
47          Q2.x=P2.x;
48          Q2.y=P2.y;
49          fractal(n-1,Q1,Q2);
50      }
51  }
52  int main()
53  {
54      int n;
55      Point p1,p2;
```

```
56      cout<<"Enter n:";
57      cin>>n;
58      cout<<"Enter Point 1:"<<endl;
59      getPoint(p1);
60      cout<<"Enter Point 2:"<<endl;
61      getPoint(p2);
62      initwindow(1280,800,"",-3,-3);
63      setfillstyle(1,15);
64      fractal(n,p1,p2);
65      getch();
66      closegraph();
67      return 0;
68  }
```

تمرین ۱

ضمیمه اول
تا ضمیمه

مبانی اشکال زدایی

توضیحات[1] و مستندات[2]

یکی از مهم‌ترین نکات در هنگام برنامه‌نویسی، درج توضیحات در کنار کد است. توضیحات به بخش‌هایی از برنامه می‌گویند که توسط مترجم برنامه به فایل اجرایی نادیده گرفته می‌شود. توضیحات برای یادآوری خود برنامه نویس از کدی که در آن قسمت نوشته، درج می‌شوند. البته برخی اوقات که برنامه بزرگ است و ممکن است چندین برنامه‌نویس بر روی یک برنامه کار کنند، این توضیحات برای فهم سایر برنامه‌نویسان همکار نیز بسیار مهم خواهد بود. فرض کنید می‌خواهید برنامه‌ای بنویسید که در آن، با گرفتن سرعت اولیه و زاویه‌ی یک پرتابه، مسیر حرکت آن را بر روی صفحه ترسیم کند و برای انجام بخشی از این برنامه، تکه کدی نوشته‌اید که قرار است در آن x مختصات پرتابه بر اساس سرعت مرحله قبلی محاسبه شود:

```
x=x+Vx*dt;
```

پس از توضیحاتی که در سطرهای بالا دادیم، منظور از این خط کد مشخص است، اما اگر این خط کد را پس از گذشت مدت زمان مثلاً یک ماه یا بیشتر بخوانیم ممکن است جزئیات آن از یادمان رفته باشد و یا اصلاً ممکن است شخص دیگری بخواهد برنامه‌ی ما را بخواند و بفهمد، پس خیلی خوب خواهد بود اگر به همراه این خطوط کد، توضیحی نیز موجود باشد!

خوشبختانه راه بسیار ساده و سرراستی برای درج توضیحات وجود دارد، اگر توضیحات تک خطی باشد (که اغلب همین گونه است)، می‌توانیم در انتهای خط که دستور اصلی را نوشتیم از دو علامت / استفاده کنیم. به این ترتیب کد بالا به صورت زیر در خواهد آمد:

```
x=x+Vx*dt; //Update x with respect to Vx and Delta t (dt)
```

[1] Comments
[2] Documentations

که جمله‌ی پس از // توضیحاتی است که برای این خط دستور نوشته‌ایم. دقت دارید که کلیه‌ی عباراتی که پس از // می‌آید در ویرایشگر Dev C++ به رنگی دیگر در خواهد آمد (شکل ض۱-۱) که به معنی آن است که کلیه‌ی این عبارات توسط مترجم نادیده گرفته خواهند شد، حتی اگر آن عبارات، عبارات برنامه‌نویسی باشند.

```
int k,x;
i.open("input.txt",ios::in);
while(!i.eof())
{
   i>>x;
   if(i.good())
      cout<<x<<endl; //Prints the sample just read from the file
}
i.close();
getch();
return 0;
```

شکل ض۱-۱: توضیحات در ویرایشگر به رنگی دیگر در می‌آیند

نوع دیگری از توضیحات وجود دارد، مثلاً برای درج توضیحات چند سطری، آن‌ها را بین علامت */ و /* قرار می‌دهیم. همواره */ برای شروع توضیحات و /* برای پایان توضیحات خواهد بود. به این ترتیب می‌توان توضیحات طولانی که معمولاً در اول برنامه‌ها می‌آیند را بین این دو علامت قرار داد (شکل ض۱-۲).

```
/*
########################################
#     This Program is a filing example    #
# and is Written for NODET Computer Book #
########################################
*/
#include<iostream>
#include<fstream>
#include<conio.h>
using namespace std;
int main()
{
```

شکل ض۱-۲: توضیحات طولانی تر در بین دو علامت */ و /* قرار می‌گیرند

البته، کاربرد // و */ و /* تنها برای درج توضیحات در برنامه نیست، بلکه خود یکی از ابزارهای اشکال‌زدایی از برنامه است، به این ترتیب که اگر بخواهیم برخی از قسمت‌های کد را موقتاً حذف کنیم، تا اجرا نشود، می‌توانیم آن بخش را اصطلاحاً تبدیل به توضیحات کنیم (و یا در زبان رایج برنامه‌نویس‌ها comment کنیم). به این ترتیب می‌توانیم اثر حذف و یا حضور یک بخش از کد را بر روی برنامه‌ی خود ببینیم. این کاربرد اگر چه خیلی مفید به نظر نمی‌رسد اما برخی مواقع بسیار سودمند است و تصحیح برنامه‌ها با آن بسیار آسان خواهد شد.

رفع خطاهای ایجاد شده توسط مترجم

خطاهایی که در نوشتن یک برنامه ایجاد می‌شوند دو دسته هستند:

۱) خطاهای زمان ترجمه

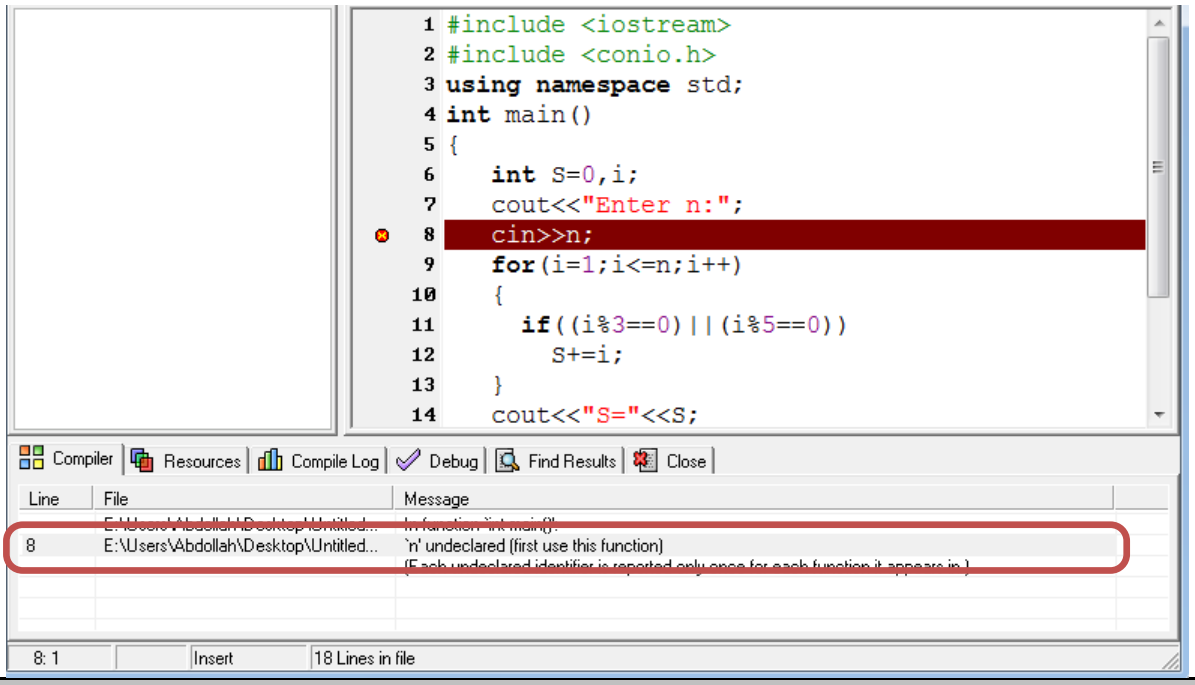
۲) خطاهای زمان اجرا

خطاهای زمان ترجمه یا اصطلاحاً خطاهای دستور زبانی، خطاهایی هستند که ناشی از نوشتن اشتباه برنامه هستند. این خطاها با اندکی تجربه در مورد خطاهای رایج برنامه‌نویسی و دقت به

لیست خطاهای ایجاد شده توسط مترجم برطرف می‌شوند. مثلاً چنانچه در انتهای یک خط ; گذاشته نشود و یا یک دستور اشتباه درج شود و یا دستوری استفاده شود اما header مربوطه include نشود، چنین خطاهایی بروز پیدا می‌کنند و برنامه اصلاً اجرا نخواهد شد تا این خطاها رفع گردد. در زیر چند مثال از خطاهای رایج در Dev C++ آمده است:

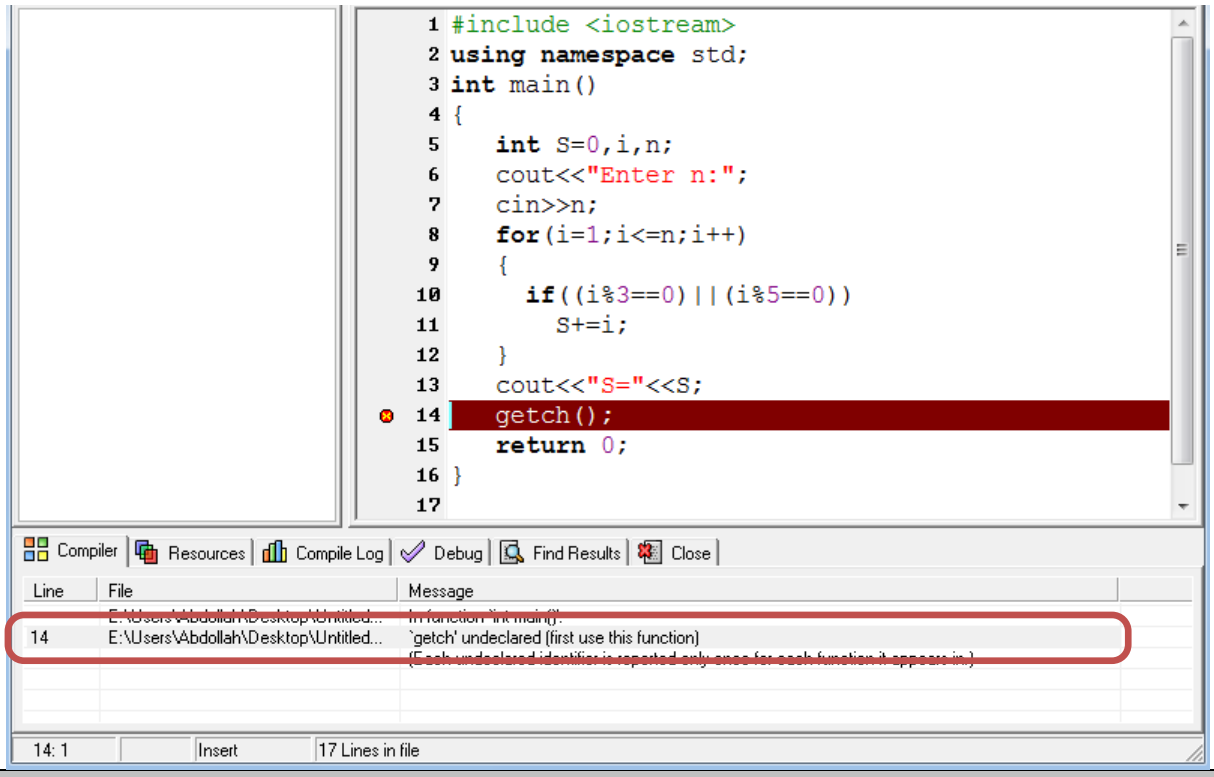
۱) عدم تعریف متغیر یا تابع

زمانی که از یک متغیر یا تابع استفاده می‌کنیم، باید آن را قبل از استفاده تعریف کنیم. به عنوان مثال در شکل ض۱-۳ خطایی در خط هشتم برنامه به وجود آمده که ناشی از عدم تعریف متغیر n است. برای رفع این خطا باید متغیر n را درلیست متغیرها در خط ۶ تعریف کنیم.



شکل ض۱-۳: خطای ناشی از عدم تعریف متغیر n

این مشکل ممکن است در مورد دستورات یا سایر توابع نیز رخ دهد، مثلاً چنانچه <math.h> را #include نکنیم اما از دستوری که در آن است (مثلاً sqrt) استفاده کنیم، خطایی مشابه ایجاد می‌شود. در شکل ض۱-۴ عدم #include فایل <conio.h> باعث شده دستور getch() شناخته نشود.

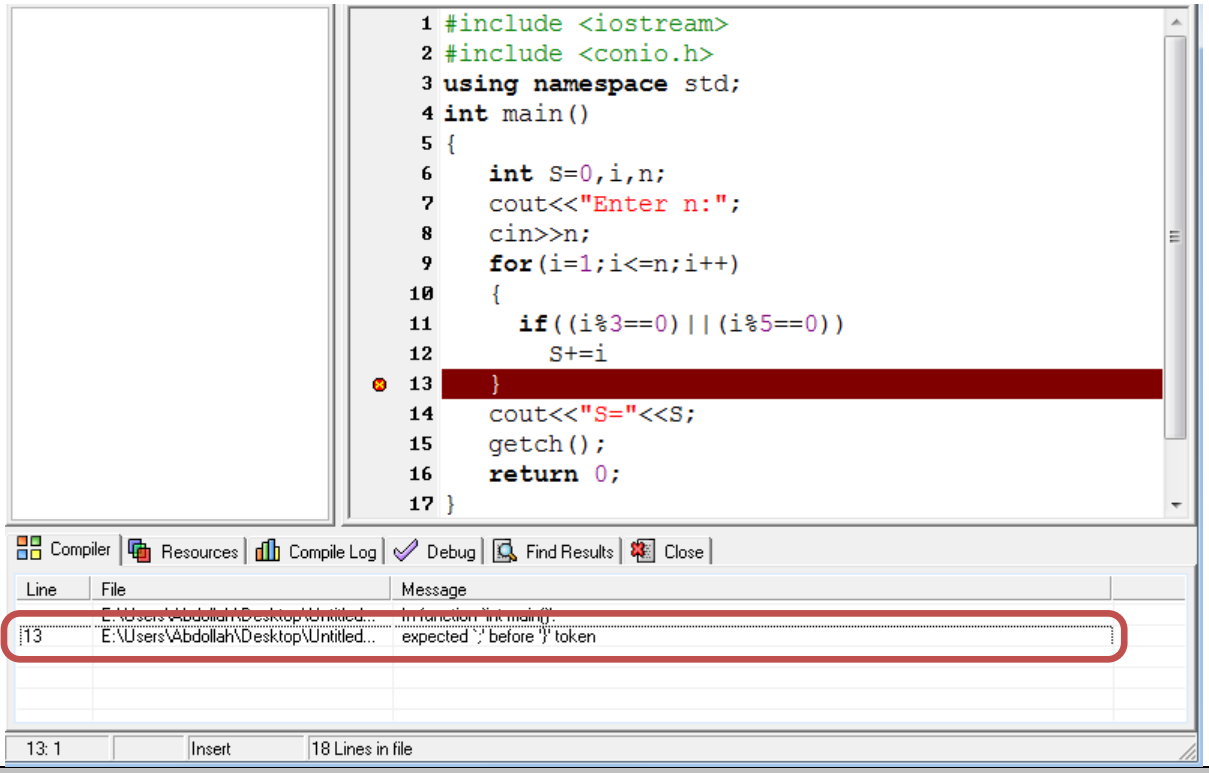


شکل ض۱-۴: خطای ناشی از استفاده از getch() بدون include کردن <conio.h>

دقت کنید که شماره‌ی خط هر خطا در کنار آن نوشته شده، مثلاً در شکل ض۱-۳ خطا در خط ۸ و در شکل ض۱-۴ خطا در خط ۱۴ اتفاق افتاده است.

۲) عدم درج نقطه-ویرگول در پایان دستورات

می‌دانیم در پایان دستورات در زبان C باید علامت ; قرار بگیرد، پس هرجا که این علامت جا بیفتد تولید خطا خواهد شد. در شکل ض۱-۵ نمونه‌ی این خطا را می‌بینید:



شکل ض۱-۵: خطای ناشی از عدم درج ; پس از اتمام دستور (باید قبل از علامت } نقطه ویرگول یا همان ; درج شود)

دقت کنید که خطای عدم درج ; در خط بعد از آن بروز پیدا می‌کند. مثلاً در شکل ض۱-۵، عدم درج ; مربوط به خط ۱۲ است حال آن که خطا در خط ۱۳ بروز پیدا کرده است.

۳) عدم درج عبارت using namespace std; برای استفاده دستورات ورودی-خروجی

برای استفاده از دستورات ورودی-خروجی، یعنی همان cin و cout، باید عبارت گفته شده را در ابتدای برنامه (قبل از ()main) بنویسیم. اگر این عبارت درج نشود، مثل این است که cin و cout تعریف نشده باشند.

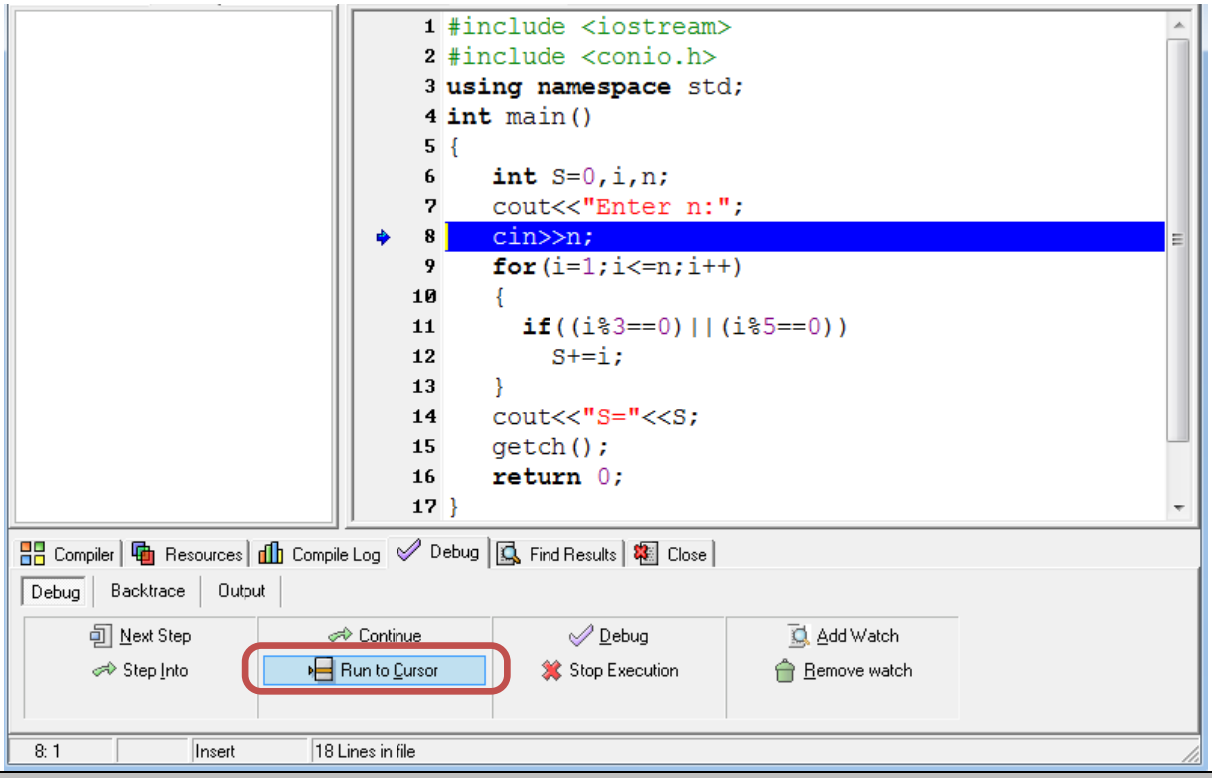
۴) عدم تنظیم پارامترها در پروژه‌های گرافیکی

پروژه‌های گرافیکی، پارامترهای ترجمه دارند که باید تنظیم شود (به ضمیمه‌ی بعدی مراجعه کنید)، عدم درج این تنظیمات باعث بروز خطا و عدم شناخت دستورات گرافیکی می‌شود.

اشکال‌زدایی از برنامه با اجرای قدم به قدم

نوع دوم خطاها، خطاهای زمان اجرا هستند که فهمیدن علت و رفع کردن آن‌ها به مراتب سخت‌تر از خطاهای نوع اول است. برای فهمیدن منشأ خطاهای از این نوع، ابتدا باید حدس زد که کدام بخش برنامه زمینه‌ی بروز خطا را فراهم کرده و سپس برای تعیین محل و سطر دقیق خطا، برنامه را خط به خط اجرا کرد و در صورت لزوم، مقادیر متغیرها را در هنگام اجرا رؤیت و بررسی کرد. حدس زدن منشأ خطا اصولاً کار راحتی نیست و بسیاری از برنامه‌نویسان حرفه‌ای نیز در این راه به مشکل بر می‌خورند. بهترین راه برای کشف منشأ خطا، اجرای خط به خط از اولین خط برنامه است. برای این کار باید از امکانات اشکال‌زدایی Dev C++ استفاده کرد. برای اجرای برنامه تا خطی مشخص، کافیست تا از منوی Debug در پایین

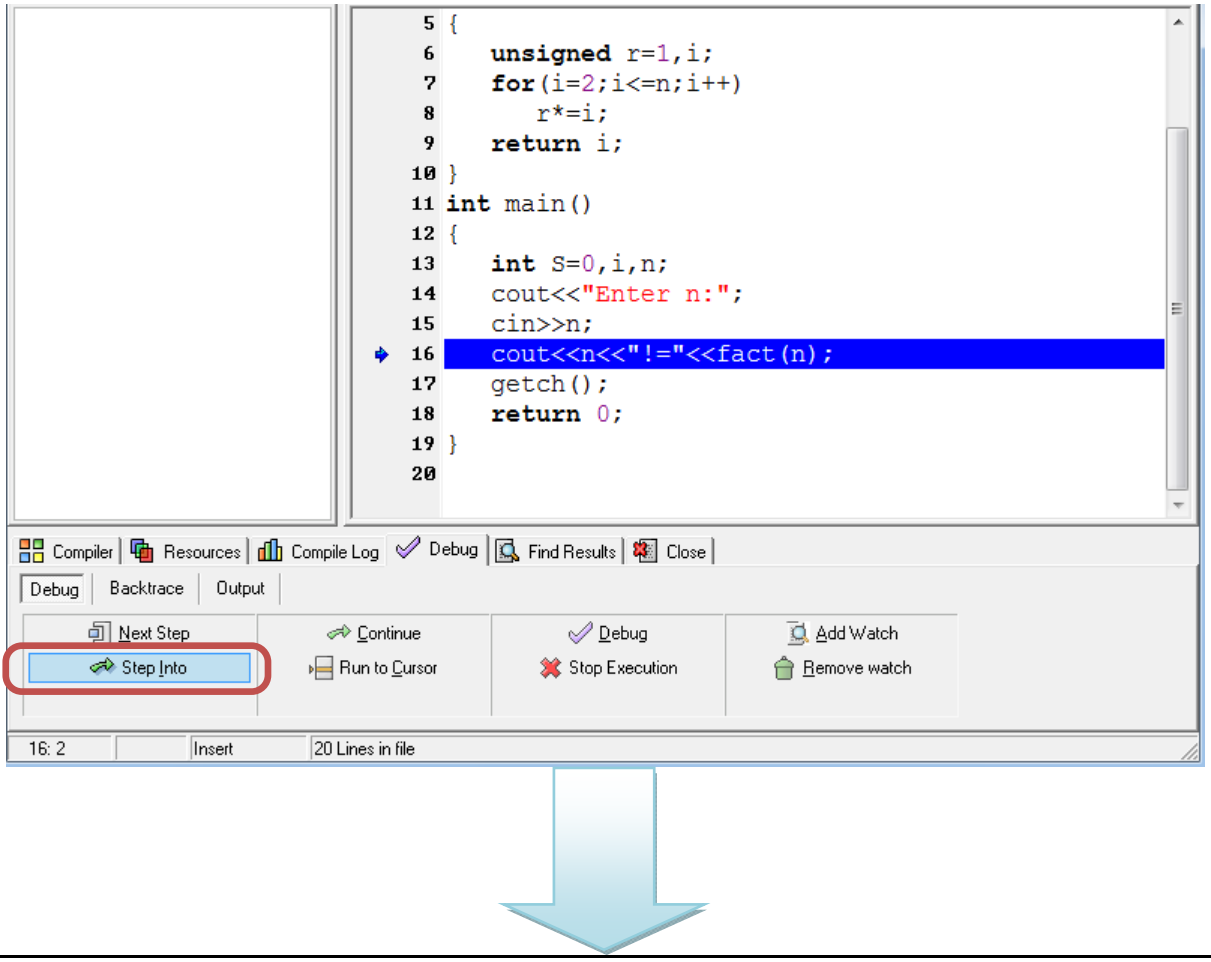
محیط Dev C++، دکمه‌ی Run to Cursor را بزنیم. در این صورت، همان‌طور که از نام این دکمه نیز پیداست، تا محل چشمک زن برنامه اجرا می‌شود (شکل ض۱-۶).

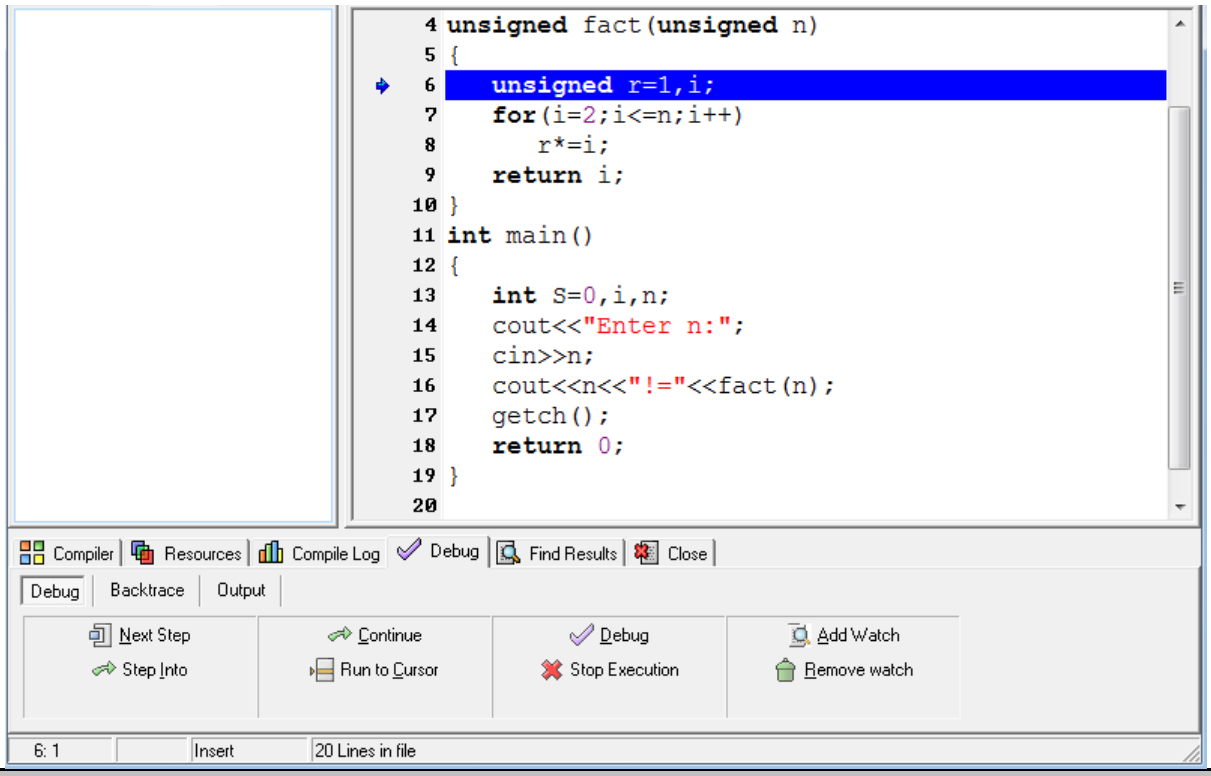


شکل ض۱-۶: اجرای برنامه تا خطی که چشمک زن در آن قرار دارد

در این صورت، خطی که برنامه تا آن اجرا شده، به رنگ آبی در می‌آید و خروجی تولید شده تا این خط نیز نمایش داده می‌شود. مثلاً در شکل ض۱-۶، عبارت Enter n: بر روی صفحه چاپ می‌شود. برای اجرای خط بعدی، از Next Step یا Step Into (در شکل ض۱-۶ در سمت چپ دکمه‌ی Run to Cursor قرار دارند) استفاده می‌شود. اولی برای مواقعی است که در این خط تابعی توسط خود ما نوشته نشده، یعنی در خود آن خط، دیگر در جستجوی چیزی نیستیم و فقط می‌خواهیم کل آن خط یکجا اجرا شود ونتیجه‌اش را ببینیم. اما اگر بخواهیم وارد

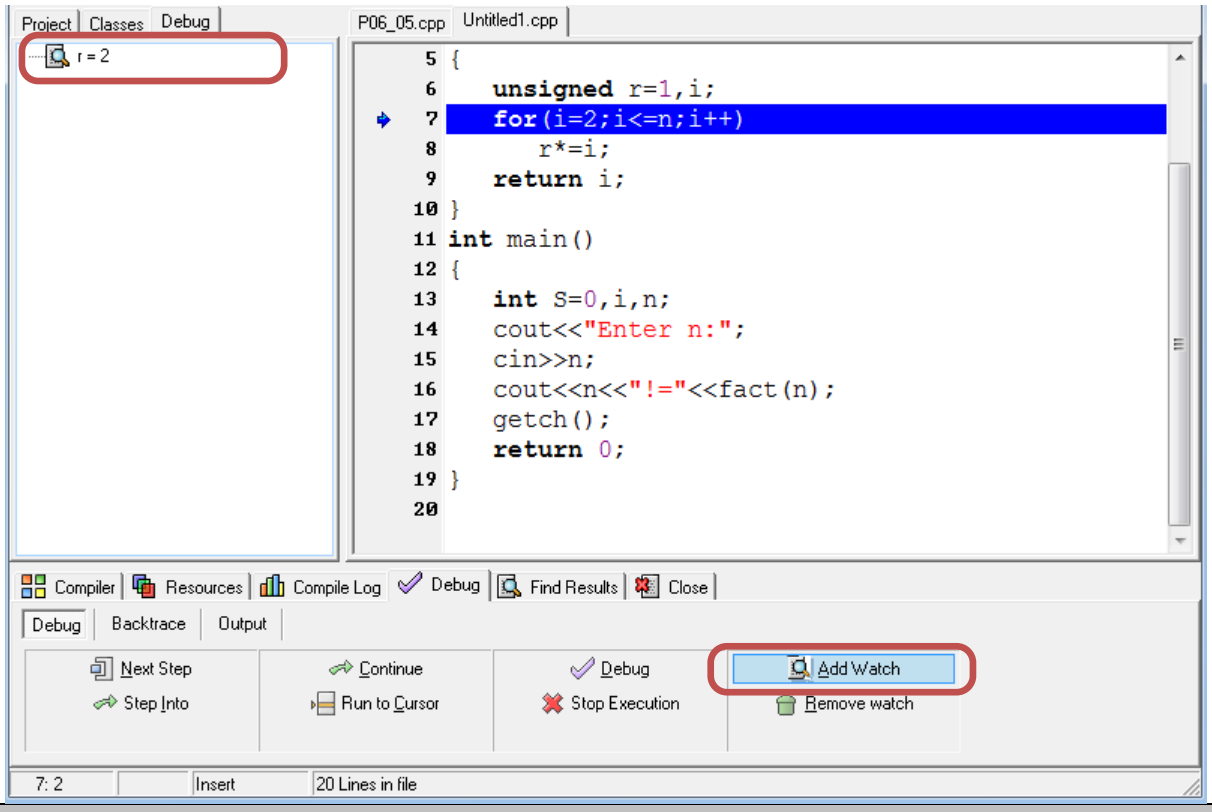
تابع یا دستوری که خودمان نوشتیم، بشویم، باید از Step Into استفاده کنیم. به عنوان مثال در شکل ض۱-۷، پس از Step Into از خط ۱۶ به اولین خط تابع fact در خط ۶ منتقل می‌شویم و می‌توانیم خط به خط تابع را اجرا کرده و نحوه‌ی تغییر مقدار متغیرها را دنبال کنیم.





شکل ض۱-۷: انجام Step Into و رفتن به داخل تابع

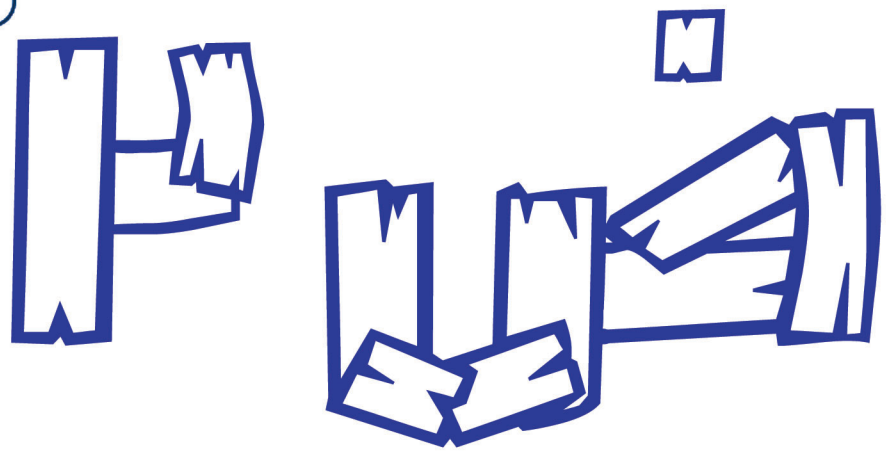
اما نحوه‌ی تغییر متغیرها را چگونه دنبال کنیم؟ مثلاً در تابع fact که در شکل ض۱-۷ نشان داده شده است، چگونه مقدار r و تغییرات آن را پس از هر بار اجرای خطوط مختلف، ببینیم؟ برای این کار کافیست تا Add Watch را در سمت راست دکمه‌های بخش Debug بزنیم و سپس در کادری که ظاهر می‌شود نام متغیر (مثلاً r) را وارد کنیم. سپس در بخش سمت چپ، مقدار متغیر قابل مشاهده خواهد بود (شکل ض۱-۸).



شکل ض ۱-۸: مشاهده‌ی مقدار متغیرها با اضافه کردن آن‌ها به بخش نظارت (Watch)

می‌توان مقدار متغیرها را به‌صورت ترکیبی از یک ساختمان یا اشاره‌گر به ساختمان نیز وارد کرد، مثلاً به‌صورت p1.y یا Head->x.

شایان ذکر است برای قطع کردن برنامه در هر جا که لازم بود، کافیست دکمه‌ی Stop Execution را بزنیم.



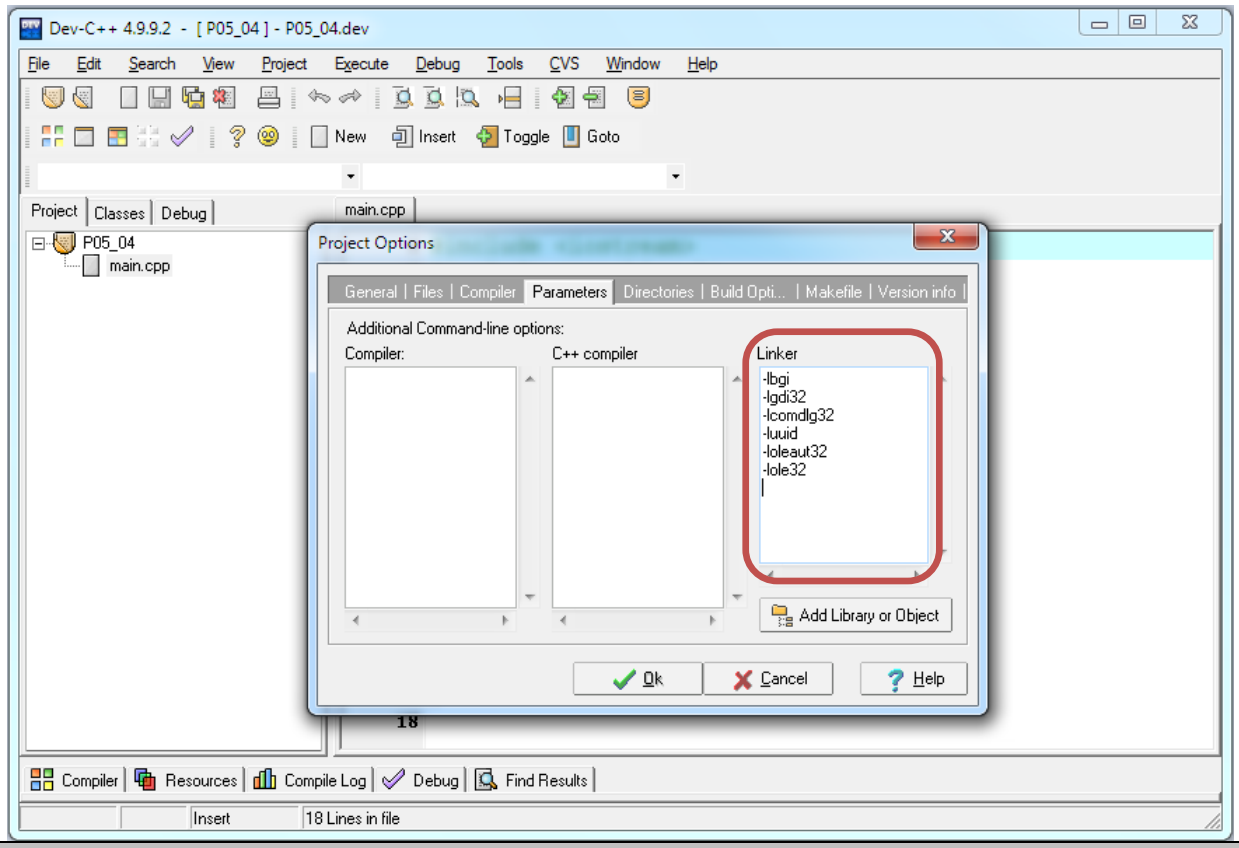
ضمیمه دوم

آشنایی با ساخت پروژه و دستورات گرافیکی

ساختن پروژه‌ی گرافیکی

برای نوشتن برنامه‌های عادی، کافیست تا یک Source File جدید را از منوی فایل (یا با Ctrl+N) ایجاد کنیم و سپس برنامه را نوشته و ترجمه و اجرا کنیم. برای برنامه‌های گرافیکی، به دلیل تنظیمات خاص آن‌ها نمی‌توان از یک فایل ساده استفاده کرد، بلکه باید یک پروژه ساخت. هر پروژه حداقل یک فایل پروژه، یک فایل Source با پسوند cpp برای نوشتن برنامه (که معمولاً نام آن main.cpp است) و یک فایل تنظیمات ترجمه دارد. توصیه می‌شود فایل‌های مربوط به هر پروژه را داخل یک پروژه با نام آن پروژه قرار دهید تا به این ترتیب هنگامی که تعداد پروژه‌هایی که ساختید زیاد شد، دچار سر در گمی نشوید و فایل‌ها با یکدیگر تداخل پیدا نکنند. در فصل اول نحوه‌ی ایجاد پروژه و نکاتی راجع به آن توضیح داده شده است، برای یادآوری می‌توانید به آن رجوع کنید.

پس از ایجاد پروژه و نوشتن برنامه‌ی گرافیکی (برای نوشتن برنامه‌های گرافیکی #include کردن <graphics.h> ضروریست)، برای ترجمه و اجرای صحیح برنامه، باید از منوی Project→Project Options (و یا دکمه‌های ترکیبی Alt+P) و سپس انتخاب Parameters، پارامترهای مربوط به ترجمه‌ی صحیح برنامه‌های گرافیکی را در آن درج کنید (شکل ض۲-۱).



شکل ض۲-۱: درج پارامترهای مربوط به ترجمه‌ی صحیح برنامه‌های گرافیکی

لوح

این پارامترها به همراه توضیح سایر دستورات گرافیکی در آدرس زیر موجود است:

CDROM:\IDE\Dev C++\DEV Cpp – BGI

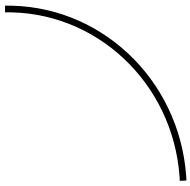
نباید کپی کردن فایلهای گرافیکی در فولدر مربوط به خودشان را فراموش کرد. نحوه‌ی انجام این کار در انتهای روش نصب Dev C++ در لوح فشرده‌ی همراه کتاب آمده است.

خلاصه‌ای از دستورات مهم گرافیکی

در این بخش برخی از دستورات مهم و پر کاربرد گرافیکی معرفی می‌شوند. این دستورات به ترتیب حروف الفبای انگلیسی در زیر آمده است:

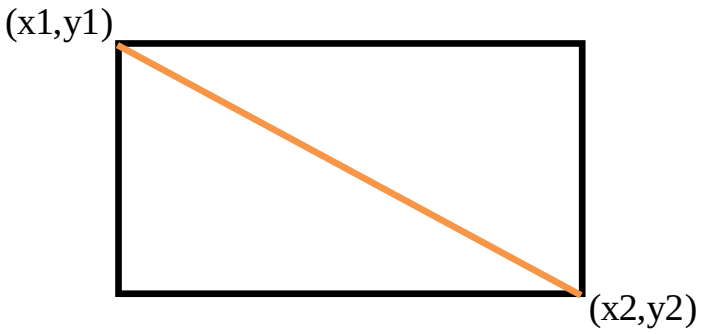
۱) void arc(int x, int y, int stangle, int endangle, int radius);

این دستور یک کمان از دایره‌ای به مرکز x, y و به شعاع r ترسیم می‌کند که از زاویه‌ی start به صورت پادساعت‌گرد شروع شده و به زاویه‌ی end (زوایا بر حسب درجه) ختم می‌شود. به عنوان مثال اگر زاویه‌ی شروع صفر و زاویه‌ی ختم ۹۰ درجه باشد شکلی به صورت زیر پدید می‌آید.



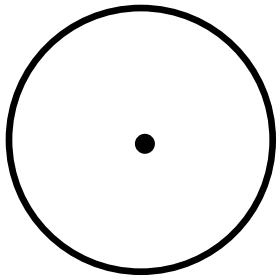
۲) void bar(int x1, int y1, int x2, int y2);

این دستور برای رسم یک مستطیل تو پر که قطر آن پاره خطی است که دو سر آن (x1,y1) و (x2,y2) هستند به کار می‌رود.



۳) void circle(int x, int y, int r);

این دستور برای رسم دایره‌ای به مرکز (x,y) و شعاع r کاربرد دارد.



۴) void closegraph()

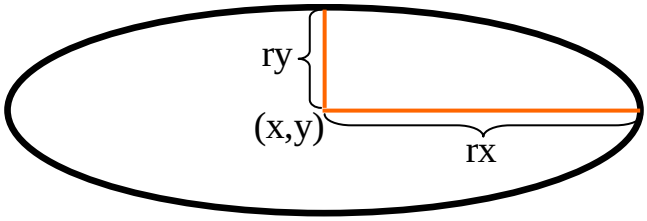
این دستور برای بستن محیط گرافیکی به کار می‌رود. معمولاً در انتهای برنامه‌های گرافیکی چنین دستوری وجود دارد.

۵) void delay(int t);

این دستور برای ایجاد t میلی ثانیه وقفه به کار می‌رود.

۶) void ellipse(int x, int y, int stangle, int endangle, int rx, int ry);

این دستور برای رسم بخشی از بیضی به کار می‌رود. مرکز بیضی در (x,y) قرار می‌گیرد و بخشی از بیضی با شروع از زاویه‌ی stangle و ختم به endangle بر حسب درجه ترسیم می‌شود. شعاع‌های راستای افقی و عمودی نیز در شکل مشخص شده است.



۷) void fillellipse(int x, int y, int rx, int ry);

این تابع برای ترسیم یک بیضی تو پر است. نقش پارامترهای ورودی درست مثل دستور قبلی است، با این تفاوت که بیضی کامل است و اجازه‌ی تعیین زاویه‌ی شروع و پایان را برای آن نداریم.

۸) void floodfill(int x, int y, int border);

این دستور برای رنگ کردن صفحه به کار می‌رود، به این صورت که از مختصات (x,y) رنگ شدن آغاز می‌شود و تا رسیدن به رنگ border ادامه می‌یابد. اگر رنگ border در صفحه موجود نباشد، کل صفحه رنگ می‌شود. رنگ و الگوی رنگ شدن صفحه توسط دستور setfillstyle مشخص می شود.

۹) int getmaxx(void);

حداکثر مختصات قابل دسترسی در جهت x را بر می‌گرداند.

۱۰) int getmaxy(void);

حداکثر مختصات قابل دسترسی در جهت y را بر می‌گرداند.

۱۱) unsigned getpixel(int x, int y);

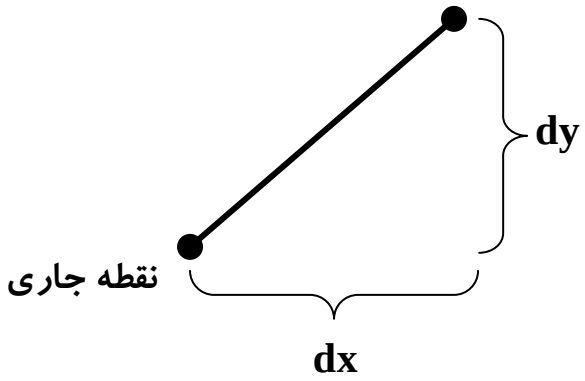
این دستور، رنگ موجود در مختصات (x,y) را بر می‌گرداند.

۱۲) int initwindow(int w, int h, const char* title, int x=0, int y=0);

این دستور، برای ایجاد یک پنجره برای ترسیمات گرافیکی است. ابعاد پنجره w×h خواهد بود و عنوان آن برابر رشته‌ی title است. این پنجره از مختصات (x,y) در پنجره اصلی کامپیوتر باز می‌شود که مقدار پیش‌فرض آن (0,0) است.

۱۳) void linerel(int dx, int dy);

این دستور، از مختصات جاری فعلی، خطی که به‌اندازه‌ی dx در راستای x و به‌اندازه‌ی dy در راستای y نسبت به نقطه‌ی جاری اختلاف مکان دارد رسم می‌کند. به شکل زیر دقت کنید.

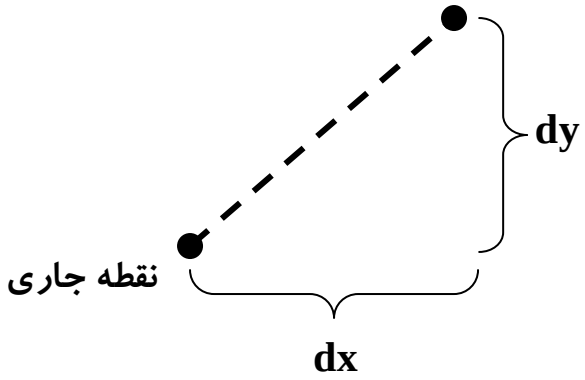


۱۴) void lineto(int x, int y);

این دستور، از مختصات جاری فعلی، خطی به مختصات (x,y) ترسیم می‌کند.

۱۵) void moverel(int dx, int dy);

این دستور، از مختصات جاری فعلی، به نقطه‌ای که به‌اندازه‌ی dx در راستای x و به‌اندازه‌ی dy در راستای y نسبت به نقطه‌ی جاری اختلاف مکان دارد *فقط جابجا می‌شود*. به شکل زیر دقت کنید.



۱۶) void moveto(int x, int y);

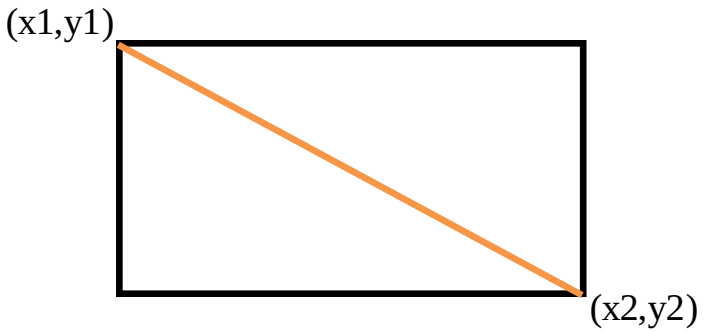
این دستور، از مختصات جاری فعلی، به مختصات (x,y) *فقط جابجا می‌شود*.

۱۷) void putpixel(int x, int y, int color);

این دستور، یک نقطه به رنگ color در مختصات (x,y) ترسیم می‌کند.

۱۸) void rectangle(int x1, int y1, int x2, int y2);

این دستور برای رسم یک مستطیل تو خالی که قطر آن پاره خطی است که دو سر آن (x1,y1) و (x2,y2) هستند به کار می‌رود.



۱۹) void setcolor(int color);

این دستور رنگ ترسیم کلیه‌ی خطوط را برابر color تنظیم می‌کند.

۲۰) void setfillstyle(int pattern, int color);

این دستور رنگ ترسیم کلیه‌ی اشیاء تو پر را برابر color و الگوی پر شدن آن‌ها را برابر pattern تنظیم می‌کند.

لوح

لیست کامل دستورات گرافیکی با توضیحات در آدرس زیر آمده است:

CDROM:\\IDE\Dev C++\DEV Cpp - BGI

\DevCpp_winBGI_Manual\index.html

مراجع

[1] Brian W. Kernighan, Dennis M. Ritchie, "The C Programming Language, 2nd Ed.", Prentice Hall, New Jersey, 1988

[2] Herbert Schildt, "C++: The Complete Reference, 3rd Edition", Osborne McGraw-Hill, New York, 1998

[3] Bjarne Stroustrup, "The C+ + Programming Language, 3rd Edition", an Imprint of Addison Wesley Longman, Inc., New Jersey, 1997

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, "Introduction to Algorithms, 2nd Edition", The MIT Press, Cambridge, Massachusetts, 2001

[۵] دکتر محمدحسن شیرعلی شهرضا، محمد شیرعلی شهرضا، "آموزش سریع الگوریتم‌ها"، نشر زمان، تهران، ۱۳۸۶

[۶] یحیی تابش، "آشنایی با الگوریتم‌ها"، انتشارات فاطمی، تهران، ۱۳۸۹

Programming with the C language, using windows tools

Abdollah Arasteh