جزوه درس برنامه نویسی به زبان C

زمستان ۱۳۹۳ـ دانشگاه قم

ویراست اول: پاییز و زمستان ۹۳

به عنایت الهی نسخه اول جزوه درس برنامه نویسی به زبان C در دی ماه سال ۹۳ آماده شد. هدف از تهیه این جزوه، جمع بندی مطالب تدریس شده در کلاس در قالب مرجعی خلاصه، برای مطالعه دانشجویان عزیز می‌باشد. در برخی از موارد، توضیحاتی علاوه بر مطالب ذکر شده در کلاس در جزوه قرار داده شده است که موجب درک بهتر مطالب می شود.

مرجع اصلی این درس کتاب «برنامه نویسی به زبان C» نوشته آقای جعفر نژاد قمی می‌باشد. دانشجویان عزیز برای مطالعه بیشتر و یادگیری کامل‌تر، می توانند از منابع زیر استفاده نمایند.

- آموزش تصویری C++ توسط آقای کیارش بازرگان – استاد دانشگاه صنعتی اصفهان
- C++ آقای حمیدرضا مقسمی – کنکور کاردانی به کارشناسی
- وب سایت cplusplus.com – قسمت Tutorials

برای دریافت مطالب مربوط به این درس، از جمله نرم افزار Turbo C++، نسخه الکترونیکی جزوه و برخی از مراجع درس به آدرس www.hm-qom.ac.blog.ir مراجعه فرمایید.

در پایان از دانشجویان کارشناسی رشته عمران که به علت تأخیر در آماده شدن این جزوه، مجبور به نوشتن مطالب شدند، عذر خواهی می‌کنم.

و من الله توفیق

هادی مهرجو – زمستان ۱۳۹۳

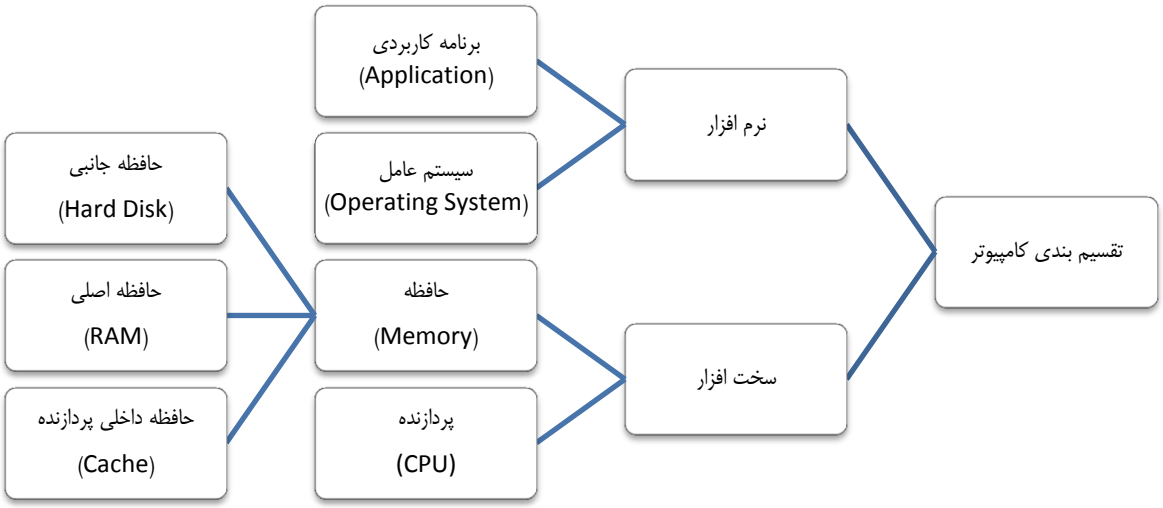
دانشگاه قم

مقدمه

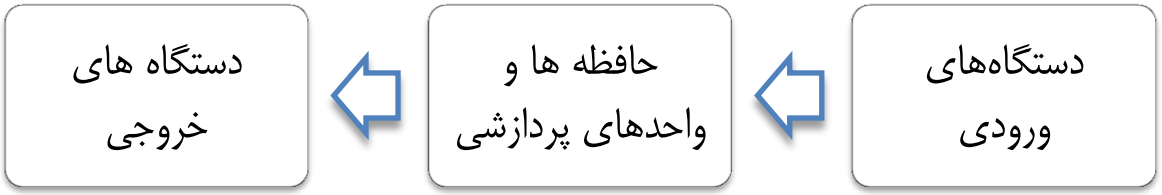
* مفاهیم پایه‌ای کامپیوتر *

در این فصل با برخی از مفاهیم اولیه کامپیوتر به طور خلاصه آشنا می شویم. پرداختن به این مقدمات، ضمن از میان بردن برخی از ابهامات نسبت به ماهیت کامپیوتر و روشن ساختن برخی از ابعاد آن، درک بهتری از برنامه نویسی و اتفاقات پس زمینه آن ایجاد می‌کند.

به طور کلی کامپیوترها شامل دو بخش سخت افزار و نرم افزار می شوند. هر یک از این دو بخش، از قسمت های مختلفی تشکیل می شوند. نمودار زیر، اجمالاً اجزای تشکیل دهنده سخت افزار و نرم افزار را نمایش می دهد.



بخش سخت افزاری کامپیوتر را می توان با زاویه دید دیگری نیز تقسیم بندی نمود. از این منظر، ورودی، خروجی یا داخلی بودن قطعات سازنده مورد بررسی قرار می‌گیرد. نمودار زیر این تقسیم بندی را ترسیم می نماید:



برخی از دستگاه های ورودی: Keyboard, Mouse, Scanner, DVD Rom/While Reading

برخی از دستگاه های خروجی: Monitor, Speakers, Printer, DVD Rom/While Writing

در ادامه به توضیح اجمالی برخی از اجزای تشکیل دهنده کامپیوتر می پردازیم.

۱- بورد مادر (Mainboard یا Motherboard): وظیفه فراهم کردن بستر ارتباطی قطعات مختلف مثل حافظه‌ها و پردازنده را به عهده دارد. به عبارت دیگر Mainboard مانند زمینی است که قطعات مختلف روی آن قرار می گیرند و به هم مرتبط می شوند.

۲- پردازنده (Central Processor Unit): وظیفه پردازش اطلاعات یا به عبارت دیگر انجام محاسبات را بر عهده دارد. این قطعه به منزله مغز کامپیوتر است. تمام محاسبات پردازنده، در مبنای دو یعنی با ارقام ۰ و ۱ انجام می‌گیرند. به این منطق محاسبات، منطق دودویی می‌گویند. در ادامه همین فصل، در بخش زبان‌های برنامه نویسی، اشاره مفصل تری به نحوه کارکرد پردازنده ها کرده‌ایم.

میزان قدرت یک پردازنده توسط فرکانس کاری آن مشخص می شود. تعداد عملیات ممکن در یک ثانیه برای یک پردازنده، همان فرکانس کاری پردازنده است. مثلاً پردازنده ای با فرکانس دو گیگا هرتز (2 GHz)، قادر است تا دو میلیارد عمل در یک ثانیه را انجام دهد. برخی از پردازنده های مشهور عبارتند از: پردازنده های Intel، AMD و IBM.

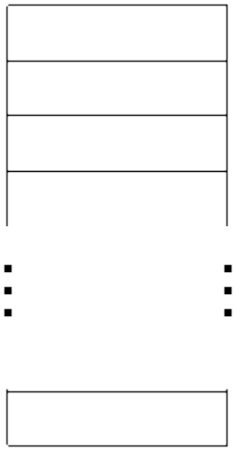
۳- حافظه (Memory): واحده نگه دارنده اطلاعات است. کوچکترین واحد نگه دارنده اطلاعات ثبات یا Register نام دارد که از مجموعه‌ای از ترانزیستورها ساخته می‌شود. یک رجیستر می‌تواند یک بیت از اطلاعات را در خود ذخیره کند. بیت کوچکترین واحد اطلاعات است که می‌تواند مقدار یک یا صفر منطقی را اختیار کند. اعداد صفر و یک، قراردادی هستند که بین بخش سخت افزار و نرم افزار کامپیوتر مقرر شده است و الا ماهیت واقعی ۰ و ۱، ولتاژهای الکتریکی است. به همین خاطر از واژه "منطقی" برای توصیف ۰ و ۱ استفاده می‌کنیم. مثلاً در سیستم‌های قدیمی، مقدار ۵ ولت معادل یک منطقی و ۰ ولت معادل صفر منطقی تلقی می‌شد. به وسیله این قرار داد، زمینه ارتباط برقرار کردن با پردازنده و حافظه با استفاده از اعداد مبنای دو فراهم می‌شود. تمامی اطلاعات تصویری، صوتی، متنی و یا عددی با شیوه‌های گوناگونی به اعداد مبنای دو تبدیل شده و سپس در خانه‌های حافظه یا رجیسترها ذخیره می‌گردند.

خانه های حافظه از کنار هم قرار گرفتن رجیسترها ساخته می‌شوند. در سیستم های قدیمی، رجیسترها، به صورت ۸ بیتی و ۱۶ بیتی ساخته می شدند و از کنار هم قرار گرفتن آنها، بلوک های حافظه ساخته می شد. امروزه با ارتقای سخت افزارها، رجیسترها به صورت ۳۲ بیتی و در سال های اخیر ۶۴ بیتی ساخته می شوند. دلیل این تغییر، بالا بردن سرعت انتقال اطلاعات است. رجیسترهای ۶۴ بیتی قادرند که ۶۴ بیت از اطلاعات را در آن واحد در خود ذخیره کنند و در هنگام نیاز، این ۶۴ بیت را در یک لحظه در اختیار پردازنده قرار دهند. خطوط انتقال اطلاعات که به این رجیسترها منتهی می شوند، ظرفیت جا به جایی ۶۴ بیت از اطلاعات در یک لحظه را دارا هستند. برای درک بهتر تفاوت رجیسترهای ۸ بیتی و ۶۴ بیتی، یک اتوبان فرضی ۸ بانده و ۶۴ بانده را در نظر بگیرید. همچنین فرض کنید ورودی شهرهایی که در دو سر این اتوبان ها قرار دارند، عوارضی های ۸ بانده (برای اتوبان ۸ بانده) و ۶۴ بانده (برای اتوبان ۶۴ بانده) باشند. بدیهی است که ظرفیت ترافیکی جاده ۶۴ بانده ۸ برابر ظرفیت جاده ۸ بانده است. از طرفی ورودی شهرها هم جوابگوی دریافت این حجم بالای ترافیکی هستند.

خطوط انتقال میان رجیسترها مشابه اتوبان های این مثال و خود رجیسترها مشابه شهرهای ورودی هستند. بنابراین سیستم هایی که به صورت ۶۴ بیتی ساخته شده اند، توانایی ذخیره کردن و انتقال حجم بالایی از اطلاعات را دارند.

دقت داشته باشید که حافظه هایی که نام بردیم، همگی حافظه های داخلی پردازنده ها یا کش (Cache) هستند. ساختمان داخلی حافظه‌های اصلی و حافظه‌های جانبی به شکل دیگری پیاده سازی می شوند. به دلیل اهمیت سرعت انتقال داده در پردازنده، یکی از اطلاعاتی که هنگام خرید یک پردازنده مورد توجه قرار می گیرد، ۳۲ بیتی یا ۶۴ بیتی بودن آن است.

در مثال‌هایی که در این فصل یا فصل های آینده می زنیم، فرض بر این است که حافظه از کنار هم قرار گرفتن رجیسترهای ۸ بیتی ساخته شده است. نمای بیرونی این حافظه ها به صورت زیر است.



8 Bit Registers

به ۸ بیت از اطلاعات در کنار هم، اصطلاحاً یک بایت گفته می شود.در این شکل هر کدام از مستطیل ها نمایانگر یک رجیستر ۸ بیتی است که می تواند ۱ بایت یا ۸ بیت از اطلاعات را در خود نگه داری کند. معمولاً برای بیان حجم فایل ها یا فضای خالی و اشغال شده حافظه از واحد بایت استفاده می کنند. مثلاً یک هارد دیسک معمولی امروزی،۵۰۰ گیگا بایت، ظرفیت ذخیره سازی اطلاعات دارد.

واحدهای اندازه گیری حافظه به همراه مقدار دقیق داده ها در هر یک، بر حسب بایت عبارتند از:

1 Byte = 8 Bit

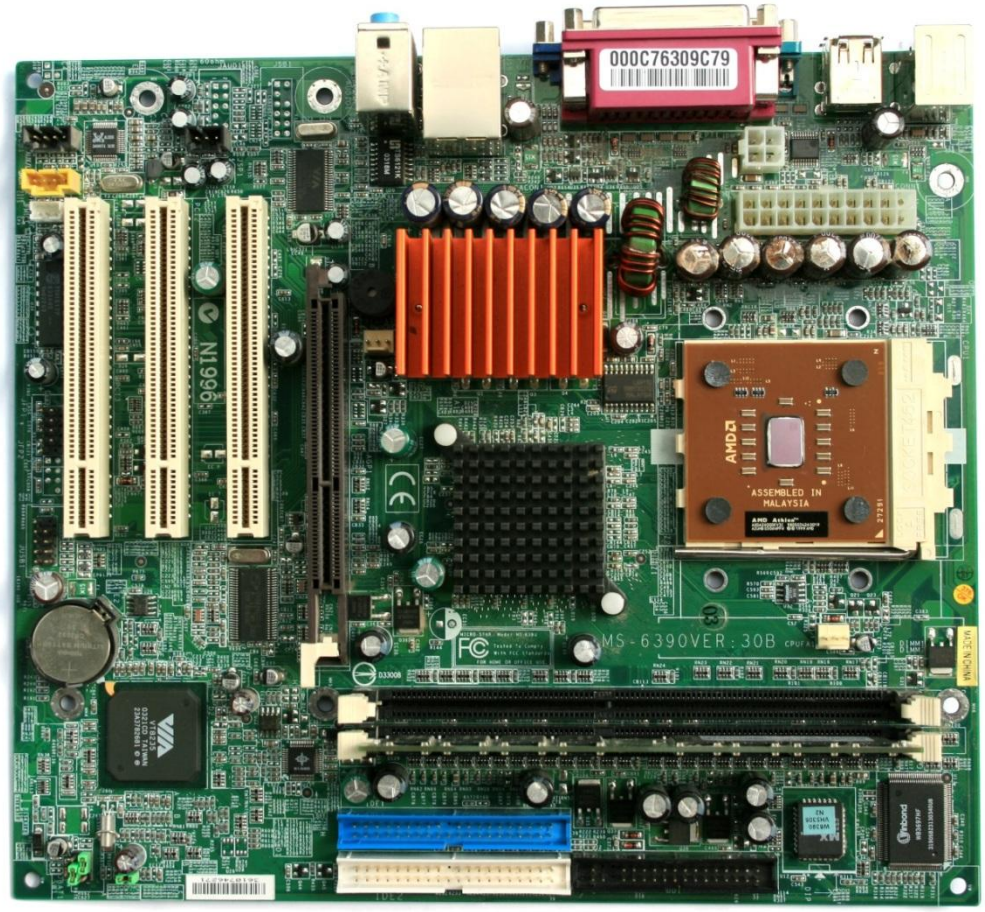
1 Kilo Byte (1 KB) = 2^{10} Byte (1024 Byte) ~ حدوداً ۱۰۰۰ بایت

1 Mega Byte (1 MB) = 2^{10} Kilo Byte (1024 KB) ~ حدوداً ۱۰۰۰ کیلو بایت

1 Giga Byte (1 GB) = 2^{10} Mega Byte (1024 MB) ~ حدوداً ۱۰۰۰ مگا بایت

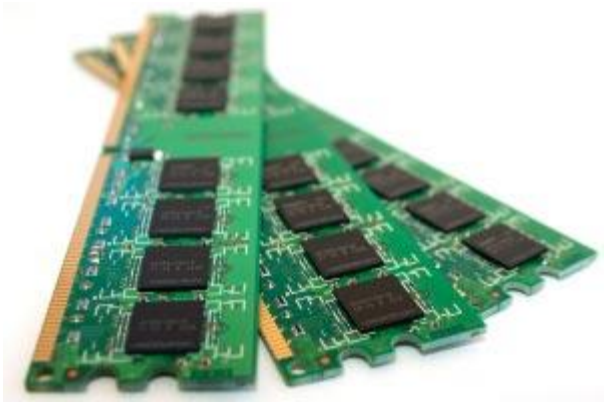
1 Tera Byte (1 TB) = 2^{10} Giga Byte (1024 GB) ~ حدوداً ۱۰۰۰ گیگا بایت

در تصاویر زیر نمونه‌های امروزی قطعات نام برده شده، نمایش داده شده اند.



نمونه امروزی یک Motherboard

پردازنده شرکت Intel



حافظه اصلی یا RAM

حافظه‌های جانبی یا Hard Disk

* زبان‌های برنامه نویسی *

زبان‌های برنامه نویسی (یا زبان‌های برنامه سازی) ابزاری هستند که بین لایه نرم افزار و سخت افزار ارتباط برقرار می‌کنند. به این معنا که توسط این زبان ها برنامه نویس قادر می شود که امکانات سخت افزاری سیستم از جمله حافظه و پردازنده را برای تولید یک نرم افزار یا برنامه کاربردی به کار گیرد. تمامی زبان های برنامه نویسی توسط دستورات به خصوصی به حافظه و پردازنده سیستم فرمان می دهند. بنابراین با تسلط بر این زبان ها، می توان سخت افزار کامپیوتر را در جهت کاربرد مورد نظر به کار گرفت.

مثلاً یک برنامه ماشین حساب ساده را در نظر بگیرید. ضرب اعداد چند رقمی در چند رقمی یا تقسیمات اعشاری طولانی، عملیاتی هستند که برای ما طولانی و خسته کننده است. با استفاده از قدرت محاسباتی بسیار بالای پردازنده‌ها، به راحتی می توان این عملیات را در کسری از ثانیه محاسبه نمود و پاسخ مورد نظر را دریافت کرد. برنامه‌نویس به واسطه آشنایی با چگونگی ارتباط برقرار کردن با سیستم، می تواند از امکانات محاسباتی و حافظه های کامپیوتر، برای نوشتن برنامه ماشین حساب استفاده کند و آن را در اختیار همه قرار دهد.

پردازنده ها به صورت سخت افزاری (فیزیکی) می توانند چندین عمل محاسباتی پایه را انجام دهند. این قابلیت توسط مدارات الکترونیکی سازنده آنها برایشان فراهم شده است. عملیات‌های پیچیده تر، توسط ترکیب همین چند عمل پایه قابل اجرا هستند. مثلاً فرض کنید که پردازنده ای به صورت الکترونیکی قابلیت انجام عمل جمع کردن را داشته باشد. همان طور که می دانید، ضرب دو عدد به معنای جمع زدن یکی از اعداد ضرب با خودش، به تعداد عدد دیگر ضرب است. مثلاً ۵ ضربدر ۷ یعنی این که عدد ۷، پنج بار با خودش جمع شود یا اینکه عدد ۵، هفت مرتبه با خود جمع شود. بنابراین عملیات ضرب به راحتی از روی عملیات جمع قابل پیاده سازی است. همین طور به توان رساندن اعداد و غیره. با استفاده از مدارات سازنده پردازنده ها که قابلیت انجام اعمال پایه ای را به آنها می دهند، اعمال پیچیده گوناگونی را می توان انجام داد.

راه ارتباطی با هر پردازنده، از طریق سیگنال های کنترلی (که از جنس ولتاژ هستند) می باشد. این سیگنال ها از قبل توسط سازنده هر پردازنده، برای آن تعریف شده است. روی پردازنده ها، پایه هایی تعبیه شده است که قابلیت دریافت ولتاژهای الکتریکی را دارند. به این پایه ها، پایه‌های ورودی می گوییم. این پایه های فلزی ظریف، ولتاژهای دریافتی را به داخل پردازنده انتقال می دهند. سیگنال های کنترلی نام برده شده، در واقع همان ولتاژهای الکتریکی هستند. در پردازنده های کامپیوتری، دو سطح متفاوت ولتاژ (مثلاً ۰ ولت و ۵ ولت) برای ارتباط با پردازنده تعریف شده است. بدین ترتیب پردازنده می تواند پیام هایی را توسط ۰ و ۱ های پشت سر هم (ولتاژهای ۰ و ۵ ولت) دریافت کند.

مثلاً فرض کنید که در پردازنده ای کد عمل جمع، ۰۰۱ باشد. با ارسال ولتاژهای صفر، صفر و یک پشت سر هم – که همان سیگنال کنترلی عمل جمع است – پردازنده متوجه می شود که عمل جمع را باید انجام دهد.

اینکه ارسال دو مرتبه صفر پشت سر هم و سپس ارسال یک در چه فاصله زمانی باشد یا اینکه پردازنده چگونه این ولتاژها را تمیز می دهد از موضوع این درس خارج است. مطالب فوق تنها مقدماتی بودند برای آشنایی با نحوه کار با پردازنده ها و ورود به بحث زبان های برنامه نویسی.

بنابر آنچه گفته شد هر پردازنده دارای لیستی از سیگنال های کنترلی از پیش تعریف شده است، که با استفاده از آنها می توان دستورات پردازشی تعبیه شده در آن را اجرا نمود. به مجموعه این لیست، زبان ماشین گفته می شود. به زبان ساده، زبان ماشین، زبان فرمان دادن و ارسال داده به پردازنده با استفاده از کدهای ۰ و ۱ (اعداد مبنای دو) است. این زبان، پایین ترین سطح از زبان های برنامه نویسی است که شامل مجموعه ای از کدهای مبنای دو برای فرمان دادن به پردازنده است.

اما برنامه نویسی توسط این زبان، دشوار و پیچیده است. اولاً برنامه نویس باید به لیست دستورات پردازنده که در قالب اعداد مبنای دو هستند مسلط باشد. دوماً نوشتن برنامه های پیچیده که نیازمند حجم زیادی از برنامه نویسی است، تقریباً غیر ممکن است. چرا که برنامه نویس دائماً نیازمند مرور برنامه خود و کم و زیاد کردن قسمت های مختلف برنامه است اما به دلیل ناخوانایی برنامه، انجام این کار بسیار سخت می باشد. سوماً در صورت بروز خطا در برنامه، بررسی کدهای صفر و یک برای دنبال کردن روال برنامه و پیدا کردن ایراد، بسیار دشوار است.

نمونه ای از کدهای زبان ماشین را در تصویر زیر مشاهده می نمایید.

A *machine code* program for adding 1234 and 4321. This is the lowest level of programming: direct manipulation of the digital electronics. (The right column is a continuation of the left column).

10111001	00000000
11010010	10100001
00000100	00000000
10001001	00000000
00001110	10001011
00000000	00011110
00000000	00000010
10111001	00000000
11100001	00000011
00010000	11000011
10001001	10100011
00001110	00000100
00000010	00000000

به دلیل دشواری های ذکر شده، زبان های سطح بالاتر از جمله زبان اسمبلی و زبان C، نوشته شدند. ویژگی این زبان‌ها این است که در آنها به جای استفاده از کدهای صفر و یک برای فرمان دادن به پردازنده و حافظه، از یک سری کلمات کلیدی استفاده می شود که از طرفی قابل فهم برای انسان است و از طرف دیگر قابل تبدیل به کدهای زبان ماشین می باشد.

این زبان‌ها، عموماً به دو سطح زبان های سطح بالا و سطح پایین تقسیم می شوند. زبان های سطح پایین به زبان‌هایی اتلاق می شوند که به زبان ماشین نزدیک‌تر باشند یا به عبارت دیگر به راحتی قابل ترجمه به کدهای زبان ماشین باشند؛ مانند زبان اسمبلی (Assembly). این زبان ها، به زبان پردازنده نزدیک تر و از زبان انسان دورتر هستند. توضیح بیشتر اینکه در یک زبان سطح پایین مانند زبان اسمبلی، دستورات نوشته شده، مستقیماً به کدهای زبان ماشین تبدیل می شوند. یعنی هر خط از دستورات این زبان، معادل یک خط از دستورات زبان ماشین است. به عبارت دیگر، زبان اسمبلی ترجمه کدهای زبان ماشین به دستورات قابل فهم برای انسان است.

مزیت این زبان ها، قدرت مانور بالای آنها در برنامه نویسی است، به این معنا که تقریباً در میزان دسترسی به حافظه و پردازنده آزادند. اما از طرفی به علت دور بودن از زبان انسان و احتیاج به حجم بالای کد نویسی برای پیاده کردن مقصود مورد نظر برنامه نویس، کار کردن با آنها قدری دشوار و نیازمند حوصله است.

زبان های سطح بالا از جمله زبان C شباهت زیادی با زبان محاوره انسان دارند یا به عبارت دیگر، کلمات کلیدی آنها، اشتراک زیادی با زبان روزمره ما دارند. مثلاً برای نوشتن یک شرط در زبان C از کلمه کلیدی if استفادی می کنیم. حال آنکه نوشتن شرط در زبان اسمبلی به این سادگی نیست. همین موضوع، دلیل سهولت برنامه نویسی با زبان های سطح بالاست.

بر اساس آنچه گفته شد، زبان های سطح بالا به طور مستقیم برای پردازنده قابل فهم نیستند چرا که بعضی از دستورات آنها معادل چندین دستور زبان ماشین است. بنابراین زبان های سطح بالا نیازمند مترجمی هستند که دستورات نوشته شده توسط برنامه نویس را به زبان ماشین تبدیل نماید. این مترجم اصطلاحاً کامپایلر (Compiler) نام دارد. پس از اتمام برنامه نویسی در یک زبان سطح بالا مانند C، برنامه نوشته شده کامپایل می شود و در صورت نداشتن خطا، آماده اجرا می شود.

همان طور که گفته شد، هر زبان برنامه نویسی دارای یک سری کلمات کلیدی است که برنامه نویس توسط آنها برنامه های مورد نظر خود را پیاده سازی می کند. این کلمات کلیدی قبلاً توسط نویسنده زبان، تعریف شده اند. به طور مثال در زبان اسمبلی برای جمع دو مقدار از واژه "ADD" استفاده می کنیم حال که در زبان C برای انجام عملیات جمع از علامت "+" استفاده می کنیم. در ادامه در طول بخش های مختلف این جزوه با برخی از کلمات کلیدی پرکاربرد در زبان C آشنا می شویم.

از آنجایی که محاسبات مبنای دو و منطق دودویی، پایه کار سخت افزار است و یادگیری این محاسبات، موجب فهم بهتر عملکرد سخت افزار و نرم افزار و تعامل این دو با هم می شود، در ادامه مطالب به معرفی اعداد در مبنای دو و چگونگی تبدیل آنها به اعداد مبنای ده و بالعکس می پردازیم.

*** معرفی مبنای دو ***

فصل اول – متغیرها و عملگرها در زبان C

عناوین مطالب:

- معرفی زبان C
- متغیرها در زبان C
- عملگرها و عملوندها در C

* معرفی زبان C *

زبان C یکی از پرکاربردترین و قدیمی ترین زبان های برنامه نویسی کامپیوتر است. با گذشت زمان و تغییر اقتضائات برنامه نویسی، نسخه های اولیه زبان C هم کامل تر شدند. زبان C++ مشهورترین نسخه ارتقا یافته زبان C است که هم اکنون در ساخت بسیاری از نرم افزارها مورد استفاده قرار می‌گیرد. از آخرین نسخه های ارتقا یافته C می توان زبان C# را نام برد که امروزه در تولید نرم افزارهای تحت ویندوز کاربرد فراوانی دارد.

تسلط بر زبان C، به معنای ورود به عرصه برنامه نویسی و ساخت برنامه های کاربردیست.

همان طور که گفته شد، C یکی از زبان های برنامه نویسی سطح بالا محسوب می شود. بنابراین برای اجرای دستورات زبان C بر روی یک کامپیوتر (یا پردازنده) نیازمند کامپایلر این زبان هستیم تا برنامه نوشته شده را به کدهای زبان ماشین تبدیل نماید. بدون این کامپایلر کدهای نوشته شده قابل اجرا بر روی پردازنده های کامپیوتر نمی باشند.

بنابراین برای اجرای یک برنامه C، داشتن دو چیز ضروری است:

۱- محیط ویرایش متن مانند نرم افزار Notepad برای نوشتن کدهای برنامه
۲- کامپایلر زبان C برای ترجمه این کدها به زبان ماشین

نرم افزارهای برنامه نویسی یا اصطلاحاً IDE ها، در قالب یک نرم افزار، این امکانات را در کنار هم جمع کرده اند. IDE مخفف عبارت Integrated Development Environment به معنای محیط ساخت مجتمع شده می‌باشد. منظور از این عبارت، جمع آوری ابزار مورد نیاز برنامه نویس در یک محیط واحد در قالب یک نرم افزار برنامه نویسی است. معمولاً این نرم افزارها، علاوه بر داشتن محیط ویرایش متن و کامپایلر زبان مورد نظر، امکانات مفید دیگری از قبیل Debugger یا خطایاب را در اختیار برنامه نویسان قرار می‌دهند.

نرم افزار Turbo C++ یکی از معروف ترین محیط های برنامه نویسی به زبان C است که در این درس مورد استفاده قرار گرفته است. برنامه های نوشته شده در این جزوه، توسط این برنامه کامپایل و اجرا شده اند. نسخه قابل اجرای این نرم افزار روی نسخه های ۶۴ بیتی ویندوز ۷ و ۸ روی پایگاه اینترنتی این درس قرار گرفته است.

در ادامه این فصل به معرفی مقدمات لازم برای برنامه نویسی به زبان C خواهیم پرداخت.

* متغیرها در زبان C *

متغیرها به مثابه ظرف هایی برای ذخیره داده ها در زبان های برنامه نویسی محسوب می شوند. این ظروف، محل نگه داری داده ها در حافظه کامپیوتر هستند. همان طور که در فصل گذشته اشاره شد، خانه های حافظه به صورت بلوک های هشت بیتی در کنار هم قرار گرفته اند. هر یک از این بلوک ها قادرند که هشت بیت از اطاعات را در خود ذخیره و نگه داری کنند. این بلوک ها، همان ظرف هایی هستند که می توانند انواع اطلاعات را در خود نگه داری کنند. برای درک بهتر مفهوم متغیرها، به مثال زیر توجه فرمایید:

فرض کنید تعدادی ظرف با جنس های متفاوت و چند پارچ از مایعات مختلف که آماده ریخته شدن در این ظروف باشند، در اختیار داشته باشیم. می خواهیم ظرف های هم جنس، حاوی یک نوع از مایعات باشند. مثلاً قرار داد می کنیم که ظرف های بلوری به آب اختصاص داشته باشند؛ ظرف های سفالی برای نگه داری شیر مورد استفاده قرار گیرند و ظرف های چینی مختص نگه داری آب میوه باشند. سپس برای متمایز کردن ظرف ها از هم، آنها را نام گذاری می کنیم. پس از این قرار داد می توانیم به هر میزان که نیاز داریم، ظروف را از مایعات مخصوص خود پر کنیم و از آنها استفاده نماییم. مثلاً اگر بخواهیم از سه مهمان با سه ظرف آب پذیرایی کنیم و از طرفی بدانیم که یک نفر از ایشان به اندازه یک استکان و دو نفر دیگر به اندازه یک لیوان، آب می نوشند، کافیست در ظرف بلوری شماره یک به اندازه یک استکان و در ظروف دو و سه به اندازه یک لیوان، آب بریزیم. سپس از این ظرف ها برای پذیرایی مهمانان استفاده کنیم.

متغیر ها، مشابه مثال فوق، تکلیف ذخیره شدن اطلاعات در خانه های حافظه را مشخص می کنند. با تعریف یک متغیر در دل برنامه، برنامه نویس، نام محل ذخیره اطلاعات و نوع اطلاعات قابل ذخیره در آن محل را معین می کند.

تعریف هر متغیر، شامل دو قسمت است:

۱- نام متغیر: برچسبی است که کاربر برای یک بلوک از حافظه مشخص می کند تا آن بلوک را با آن نام بشناسد (مانند نام گذاری ظروف مایعات).
۲- نوع متغیر: مشخص می کند که چه نوعی از داده‌ها می توانند در این بلوک نگه داری شوند.

برخی از انواع داده‌ها در زبان C عبارتند از:

- کارکتر (Character)
- عدد صحیح بدون علامت (مثبت) (Unsigned Integer)
- عدد صحیح علامت‌دار (Signed Integer)
- عدد اعشاری (Floating Point Number)

شکل تعریف متغیر در زبان C به این صورت است:

نوع متغیر نام متغیر;

برای مشخص کردن نوع یک متغیر از کلمات کلیدی به خصوصی در زبان C استفاده می کنیم. در ادامه لیست برخی از این کلمات به همراه مقدار حافظه تعریف شده برای هر یک ذکر شده است:

char	داده کارکتری	۸ بیت
int	عدد صحیح	۱۶ یا ۳۲ بیت
unsigned int	عدد صحیح بدون علامت	۱۶ یا ۳۲ بیت
long int	عدد صحیح بزرگ	۳۲ بیت
float	عدد اعشاری	۳۲ بیت
double	عدد اعشاری بزرگ	۶۴ بیت

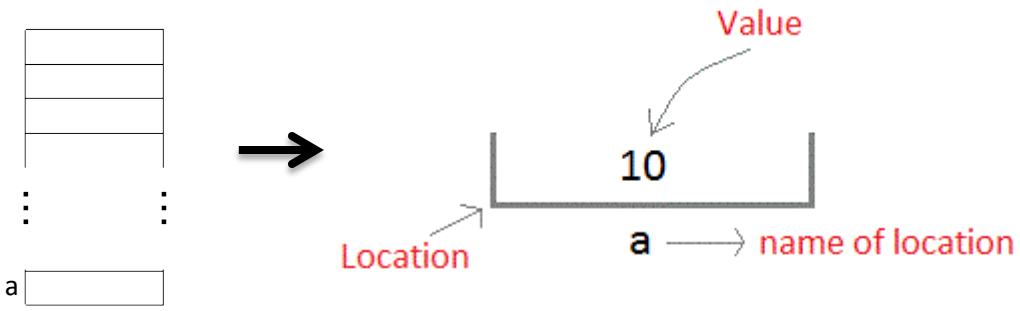
به عنوان نمونه اگر بخواهیم متغیری به نام number از نوع عدد صحیح تعریف کنیم، می نویسیم:

```
int number;
```

مقدار دهی به یک متغیر توسط دستور انتساب (علامت =) صورت می پذیرد. مثلاً اگر بخواهیم عدد ۱۰۰ را در متغیر number قرار دهیم یا اصطلاحاً ۱۰۰ را به number منتسب کنیم می نویسیم:

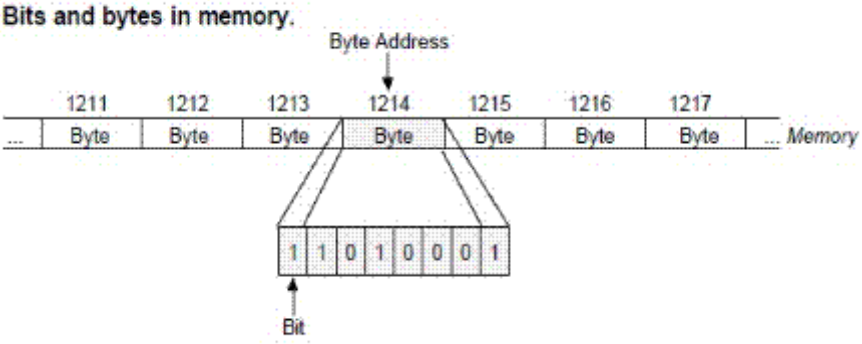
```
number = 100;
```

فرض کنید متغیری با نام a از نوع صحیح تعریف کنیم و سپس عدد ۱۰ را در این متغیر قرار دهیم، نحوه اختصاص یافتن حافظه به این متغیر، در شکل زیر نمایش داده شده است:



در ادامه این فصل، در بخش «فرم نوشتن برنامه در زبان C»، با مثال های دیگری از تعریف متغیرها و محل نوشتن دستورات مربوط به معرفی متغیرها در برنامه آشنا خواهیم شد.

همان طور که در فصل گذشته اشاره شد، تمامی داده ها به صورت اعداد مبنای دو (صفر و یک)، در حافظه ذخیره می شوند و کامپیوتر چیزی به جز این اعداد که در واقع دو ولتاژ الکتریکی متفاوت هستند نمی شناسد. بنابراین برای ذخیره شدن کارکترها، اعداد اعشاری یا اعداد علامت دار در حافظه، می بایست مکانیزم به خصوصی وجود داشته باشد که با استفاده از آن، این نوع داده‌ها ابتدا به مقادیر دودویی (صفر و یک) تبدیل شوند و سپس در حافظه ذخیره گردند. مثلاً برای ذخیره یک عدد علامت دار در حافظه، کامپایلر C، پر ارزش ترین بیت اختصاص یافته به این عدد (بیت سمت چپ) را برای ذخیره علامت عدد در نظر می گیرد. به این معنا که اگر بیت آخر عدد معادل یک شد، عدد به صورت منفی و اگر این بیت برابر با صفر شد عدد به صورت مثبت در حافظه ذخیره می گردد.



در شکل فوق، قرار گرفتن بلوک های حافظه در کنار هم به صورت افقی نمایش داده شده است.

پس از معرفی متغیرها به برنامه، برنامه نویس با نوشتن دستوراتی ، اعمالی را روی این متغیرها انجام می دهد تا نتیجه مطلوب خود را به دست بیاورد. غالب این اعمال، توسط عملگرهای زبان C انجام می شوند که در ادامه این فصل به آنها می پردازیم.

❋ عملگرها و عملوندها در C ❋

عملگرها، علائمی هستند که انجام عملیات‌های گوناگون را به عهده دارند. مثلاً جمع مقدار دو متغیر توسط عملگر "+" و انتساب یک مقدار به یک متغیر، توسط عملگر "=" انجام می پذیرد. هر عملگر، یک یا تعدادی عملوند دارد که عملیات مربوطه را روی آنها اعمال می کند. در واقع عملوندها، متغیرها یا مقادیری هستند که توسط عملگرها برای انجام عملیات‌های مختلف مورد استفاده قرار می‌گیرند.

مثلاً دستور x + 2، عملیات جمع را روی دو عملوند x و 2 پیاده می کند یا در دستور a = 6، عمل انتساب، روی دو عملوند ۶ و a صورت می پذیرد یا عدد ۶ به متغیر a انتساب پیدا می کند (عدد ۶ در متغیر a قرار می گیرد).

لیست برخی از عملگرهای پرکاربرد در زبان C به همراه کارایی هر یک در ادامه آمده است:

عملگرهای محاسباتی:

- عملگر = کاربرد: انتساب یک مقدار به یک متغیر مثال: a=2 یا ch='B'
- عملگرهای + − ❋ کاربرد: جمع، تفریق و ضرب مقادیر مثال: a+b یا a*b+c
- عملگر / کاربرد: انجام تقسیم صحیح مثال: 20/3 که برابر 6 می‌باشد
- عملگر ٪ کاربرد: محاسبه باقی مانده تقسیم مثال: 2%5 که معادل 1 است
- عملگرهای ++ و — کاربرد: افزایش یا کاهش متغیر به قدر ۱ واحد مثال: ++x یا x++
- عملگرهای =+، =-، =-* کاربرد: انجام عملیات مربوط روی طرفین تساوی و انتساب نتیجه به عملوند چپی

همان طور که در مثال اول پیداست، انتساب یک کارکتر به یک متغیر، شکل خاصی دارد. به این صورت که کارکتر می‌بایست داخل دو علامت «'» (single quotation) قرار گیرد. همچنین در عملیات انتساب، طرفین تساوی می‌توانند دو متغیر باشند؛ مثلاً a=b. در این دستور مقدار موجود در متغیر b به متغیر a منتقل می شود.

همچنین توجه داشته باشید که عملگر «/»، تقسیم صحیح دو عدد بر یکدیگر را انجام می دهد. یعنی در صورت بخش پذیر نبودن دو عدد بر هم، تنها قسمت صحیح خارج قسمت، حاصل استفاده از این عملگر است.

عملگر =+، مقادیر سمت چپ و راست تساوی را با هم جمع کرده و نتیجه را در متغیر سمت چپ، قرار می‌دهد. مثلاً دستور sum+=num، معادل دستور sum=sum+num می‌باشد. به همین ترتیب عملگرهای =- و =* به صورت مشابه عمل می کنند.

عملگرهای مقایسه ای

- عملگر < کاربرد: مقایسه بزرگتری مثال: a>2
- عملگر > کاربرد: مقایسه کوچکتری مثال: y<x
- عملگر =< کاربرد: مقایسه کوچکتر مساوی بودن مثال: i<=10
- عملگر =< کاربرد: مقایسه بزرگتر مساوی بودن مثال: i>=n
- عملگر == کاربرد: مقایسه تساوی دو متغیر مثال: a==10
- عملگر =! کاربرد: مقایسه تضاد دو متغیر مثال: x!='.'

دقت کنید که عملگر «==» به معنای انتساب نیست. مثلاً در مثال a==10، عدد ۱۰ در متغیر a ریخته نمی شود. بلکه مقدار متغیر a با عدد ۱۰ مقایسه می شود تا مساوی بودن این دو مقدار معلوم شود.

عملگر =! به معنای مقایسه تناقض دو متغیر یا تناقض یک متغیر با یک مقدار است. در مثال فوق، عبارت x!='.' به این معناست که کامپایلر، متغیر x (که از نوع کارکتری است) را با کارکتر نقطه (.) مقایسه می‌کند تا معلوم شود که آیا این دو متناقضند یا خیر.

اولویت بندی عملگرها:

اولویت اجرای عملگرها در یک عبارت محاسباتی به این معناست که کدام یک از عملگرها زودتر و کدام یک دیرتر اجرا می شوند. فرض کنید عبارتی به صورت مقابل داشته باشیم: « a=z+x*y/2-w*3 » ظاهراً ترتیب محاسبه عبارت به این صورت است که در ابتدا متغیر z با x جمع شده و حاصل در y ضرب می شود. سپس مقدار به دست آمده بر ۲ تقسیم شده و منهای مقدار متغیر w شده و حاصل در عدد ۳ ضرب می شود. در پایان هم این مقدار در متغیر a قرار می گیرد. اما کامپایلر زبان C به این صورت عمل نمی کند! بین چهار عمل اصلی، اولویت بالاتر با اعمال ضرب و تقسیم است. پس از آن اعمال جمع و تفریق، انجام می گیرند و حاصل محاسبه می گردد. اگر در عبارتی چند عملگر با اولیت یکسان به کار رفته باشد، ترتیب اجرای آنها معادل ترتیب نوشته شدنشان از چپ به راست است. به این ترتیب شکل صحیح اجرای عبارت فوق در برنامه C از این قرار است:

1- x*y
2- حاصل ضرب به دست آمده/2
3- w*3
4- نتیجه مقدار به دست آمده در مرحله دوم+z
5- عبارت به دست آمده در مرحله چهار منهای عبارت به دست آمده در مرحله سه
6- انتساب عبارت محاسبه شده به متغیر سمت چپ تساوی

بهترین کار برای به خطا نیفتادن و اطمینان از نحوه انجام شدن محاسبات، استفاده از علامت پرانتز است. با استفاده از پرانتز، می توان اولویت اجرای دستورات را تغییر داد. به این صورت که دستورات داخل پرانتز به صورت مستقل محاسبه می گردند و نتیجه حاصل، وارد باقی محاسبات می شود. مثلاً اگر عبارت مثال گذشته را به صورت: a=(z+x)*y/(2-w)*3 پرانتز گذاری کنیم، نحوه محاسبه این عبارت به صورت زیر تغییر می‌کند:

1- z+x
2- 2-w
3- y*حاصل ضرب عبارت محاسبه شده در مرحله یک
4- تقسیم عبارت محاسبه شده در مرحله ۳ بر حاصل مرحله ۲
5- ضرب حاصل مرحله ۴ در عدد ۳
6- انتساب قسمت سمت راست تساوی به قسمت سمت چپ

نکته پایانی اینکه دو دستور a=x++ و a=++x اگر چه ظاهراً دو دستور معادل یکدیگرند اما از لحاظ نتیجه تفاوت دارند. دلیل این مسأله این است که هنگام اجرای دستور a=x++، اولویت با انتساب است. یعنی ابتداً مقدار متغیر x در متغیر a قرار می گیرد، سپس یک واحد به x افزوده می شود. اما در دستور a=++x، ابتدا یک واحد به مقدار x افزوده می‌شود، سپس این مقدار در متغیر a ریخته می‌شود.

در ادامه این فصل، ساختار نوشتن یک برنامه در زبان C را بررسی می نماییم

* فرم نوشتن برنامه در زبان C *

یک برنامه زبان C شامل قسمت‌های مختلفیست که در ادامه به معرفی آنها می پردازیم.

اصلی ترین قسمت برنامه، بدنه اصلی برنامه یا اصطلاحاً قسمت main برنامه است. main، تابعی است که دستورات اصلی برنامه در آن قرار می گیرند. علت به کار بردن کلمه تابع برای main را در فصل چهارم توضیح می دهیم. در مورد سایر دستوراتی که خارج از main نوشته می شوند هم در آینده بحث خواهیم کرد. فرم کلی نوشتن main به صورت زیر است:

```
void main()
 {
 دستورات برنامه
 }
```

تمامی برنامه های زبان C، از قالب فوق پیروی می کنند. همان طور که می بینید پس از نوشتن کلمه main، یک پرانتز باز و بسته شده است و سپس در خط بعدی با باز شدن آکولاد، نوشتن دستورات برنامه آغاز می شود. در انتهای برنامه، پس از اتمام دستورات، بسته شدن آکولاد، نشانه خاتمه برنامه است.

دقت داشته باشید که در زبان C، بعضی فاصله ها ضروری و بعضی غیر ضروری هستند. وجود فاصله بین کلمات کلیدی، ضروری است چرا که کامپایلر باید بتواند دستورات مستقل را شناسایی کند. اما بعضی از فاصله ها غیر ضروری هستند. مثلاً دستورات زیر با یکدیگر معادلند و کامپایلر در زمان ترجمه، تمام فواصل اضافی را حذف می نماید. دستوراتی که با علامت تیک متمایز شده اند، همان حالت‌های اصلی هستند که باقی دستورات مشابه، در هنگام ترجمه به شکل آنها تغییر پیدا می کنند.

```
a = 2
a  = 2
a=   2
```
✓ `a=2`
```
----
void main ()
```
✓ `void main()`

در مثال های فوق، کامپایلر زبان C فاصله های غیر ضروری را به صورت خودکار حذف می نماید.

حال می پردازیم به نحوه نوشتن دستورات اصلی. همان طور که در اوایل این فصل اشاره شد، متغیرها ظرف هایی برای نگه داری اطلاعات در حافظه اند. در واقع خانه های خالی حافظه پس از معرفی متغیر، برچسب گذاری و دارای هویت می شوند. مثلاً مشخص می شود که دهمین تا سیزدهمین رجیسترها از بلوک اول حافظه به یک مقدار اعشاری اختصاص یابند و نام آنها f قرار گیرد. بدین ترتیب، با تعریف متغیرها خانه های خالی حافظه نام گذاری شده و نوع مقادیری که قرار است در این خانه ها ریخته شوند، مشخص می گردد. برای فهم بهتر این موضوع یک کتابخانه خالی که از چندین قفسه تشکیل شده است را در نظر بگیرید. قبل از قرار دادن کتاب ها در قفسه ها بهتر است برای هر قفسه برچسبی در نظر بگیریم و نوع کتاب هایی که قرار است در آن قفسه قرار بگیرند را در آن برچسب ذکر کنیم؛ مثلاً قفسه کتاب های اخلاقی، قفسه کتب دانشگاهی، قفسه کتب ادبی و مشابه آن. تعریف متغیر هم دقیقاً برچسب گذاری خانه های خالی حافظه و تعیین نوع مقداریست که در آن خانه ها قرار می گیرد. این مقادیر شامل اعداد صحیح بدون علامت، اعداد صحیح علامت دار، اعداد صحیح کوتاه، اعداد صحیح طویل، اعداد اعشاری، مقادیر کارکتری و چند مورد دیگر هستند.

همان طور که در اوایل فصل گفتیم، شکل تعریف متغیر در زبان C به این صورت است:

```
نوع متغیر   نام متغیر;
```

در قطعه کد زیر، چند متغیر به برنامه معرفی شده اند:

```
int x;
int y,w,z;
float a,b,c;
```

توجه داشته باشید که مکان تعریف متغیرها در برنامه، همان ابتدای برنامه پس از آکولاد main است.

سابق بر این دیدیم که مقدار دهی به یک متغیر با استفاده از دستور انتساب ("=") انجام می گیرد. می توانیم در هنگام تعریف متغیر، مقدار دهی آن را نیز انجام دهیم. مثلاً float a=2.2;

زبان C یک زبان حساس به کوچکی و بزرگی حروف یا اصطلاحاً Case Sensitive است. به این معنا که کامپایلر C بین حروف a و A تفاوت قائل می شود. بنابراین در قسمت های مختلف برنامه به خصوص در زمان تعریف متغیرها به نام گذاری توجه کنید. به طور مثال اگر متغیری در زمان تعریف با حروف کوچک معرفی شود و در دل برنامه با حروف بزرگ مورد استفاده قرار بگیرد، کامپایلر از برنامه نوشته شده خطا می گیرد و برنامه اجرا نمی شود.

فصل دوم: ورودی و خروجی

عناوین مطالب:

- استفاده از سرآیندها برای فراخوانی توابع
- ورودی‌ها و خروجی‌های برنامه
- نحوه دریافت ورودی
- نحوه چاپ خروجی

فصل سوم: ساختارهای تکرار و تصمیم

عناوین مطالب:

- ساختارهای تکرار در زبان C
- حلقه for
- حلقه بی‌نهایت
- حلقه while و do while
- ساختارهای تصمیم
- دستور if
- دستورات break و continue

* ساختارهای تکرار در زبان C *

ساختارهای تکرار در زبان C شامل دستوراتی هستند که موجب تکرار قسمتی از برنامه می‌شوند. به کمک این دستورات، برنامه نویس از نوشتن کدهای تکراری برای انجام برخی از عملیات‌های مورد نیاز خویش، رهایی می‌یابد.

به طور مثال می‌خواهیم برنامه‌ای بنویسیم که ده عدد را از ورودی خوانده و با یکدیگر جمع کند؛ سپس نتیجه را در خروجی نمایش دهد. بدون استفاده از ساختارهای تکرار ناگزیریم یکی از دو راه حل زیر را انتخاب کنیم:

- ده مرتبه از دستور scanf برای گرفتن ورودی از کاربر و جمع زدن آن با مقادیر قبلی استفاده کنیم:

```
sum=0;
scanf("%d",&x);
sum=sum+x;
scanf("%d",&x);
sum=sum+x;
... تا ده مرتبه
```

- ده متغیر تعریف کنیم و با استفاده از دستور scanf، مقادیر وارد شده توسط کاربر را در آنها ذخیره کنیم:

```
int a,b,c,d,e,f,g,h,i,j,sum;
scanf("%d%d%d...تا ده مرتبه...%d",&a,&b,&c,...,&j);
sum=a+b+c+...+j;
```

در روش اول، برنامه نویس ناچار است که چندین بار از دستور scanf برای خواندن ورودی از کاربر استفاده نماید.
در روش دوم، قسمت زیادی از حافظه به متغیرهای تعریف شده اختصاص می‌یابد. ضمن اینکه این روش باعث اشغال حجم زیادی از حافظه می‌شود، برای تعداد نامشخصی از ورودی‌ها جوابگو نیست.

با استفاده از ساختارهای تکرار، می‌توان به گونه‌ای برنامه نویسی نمود که بدون نیاز به نوشتن قطعه کدهای تکراری و اشغال حجم زیادی از حافظه، بتوان مجموعه‌ای از دستورات را بارها اجرا نمود. برای این کار کافیست با استفاده از دستورهای مربوط به ساختارهای تکرار، حلقه‌هایی در برنامه ایجاد کرد و به تعداد دلخواه دستورات داخل این حلقه‌ها را اجرا نمود.

ساختارهای تکرار در زبان C عبارتند از:

- دستور for
- دستور while
- دستور do while
- دستور goto

❊ حلقه for ❊

فرم کلی این دستور به صورت مقابل است:

for (معرفی و مقداردهی شمارنده حلقه ;شرط حلقه ;گام حلقه)

{

دستورات داخل حلقه

}

با استفاده از این دستور، می‌توان دستورات نوشته شده داخل آکولاد (حلقه) را به تعداد دلخواه اجرا نمود.

دستور for نیازمند سه قسمت شمارنده، شرط و گام حلقه است که در ادامه به توضیح هر یک پرداخته شده است.

شمارنده حلقه: شمارنده، وظیفه شمارش تعداد اجرا شدن دستورات داخل حلقه را به عهده دارد. پس از هر بار اتمام حلقه، به اندازه گام حلقه به شمارنده افزوده و یا از آن کاسته می‌شود. معمولاً در حلقه‌های for پس از هر بار اجرای حلقه یک واحد به شمارنده افزوده می‌شود و یا یکی از آن کاسته می‌شود.

گام حلقه: گام حلقه مشخص کننده این است که بار هر بار اجرا شدن دستورات داخل حلقه، به چه میزان به شمارنده افزوده یا از آن کاسته شود.

شرط حلقه: معین می‌کند که حلقه تا چه زمان ادامه داشته باشد. به طور مثال اگر بخواهیم توسط یک حلقه ده مرتبه از کاربر ورودی دریافت کنیم، در قسمت شرط، معین می‌کنیم که این حلقه ده مرتبه تکرار شود.

مثالی ساده برای دستور for:

می‌خواهیم برنامه‌ای بنویسیم که ده عدد را از کاربر گرفته و میانگین آنها را محاسبه نماید. توسط قطعه کد زیر که با استفاده از دستور for نوشته شده می‌توانیم بدون نیاز به نوشتن‌های مکرر، ده عدد را از ورودی دریافت کنیم و به سادگی میانگین آنها را محاسبه نماییم.

```
int i;
int sum=0;
float ave;
for (i=1; i<=10; i++)
        {
        scanf("%d",&x);
        sum=sum+x;
        }
ave=(float)sum/10;
```

با رسیدن به حلقه for در قطعه کد فوق، ده عدد به عنوان ورودی از کاربر دریافت می‌شود و پس از هر بار دریافت، مقدار آن با مجموع مقادیر قبلی جمع می‌شود. پس از اتمام حلقه، مجموع کل ورودی‌ها در sum ذخیره شده و با یک تقسیم ساده میانگین این اعداد محاسبه گردیده است. دقت کنید که تنها با استفاده از یک متغیر (متغیر x)، ده عدد از ورودی دریافت شد و مجموع آنها در متغیر دیگری به نام sum ذخیره شد.

در این قطعه کد، متغیر i به عنوان شمارنده حلقه معرفی شده است. همان طور که می‌بینید در قسمت ابتدایی پرانتز حلقه for، ضمن معرفی i به عنوان شمارنده، مقدار آن نیز معین شده است.

قسمت انتهایی پرانتز اختصاص به گام حلقه دارد. پس از هر بار اجرای دستورات داخل حلقه (دستورات داخل آکولاد)، گام حلقه اعمال می‌شود. از آنجایی که در این مثال، گام حلقه ما ++i است، با اجرای دستورات حلقه، یک واحد به i یا همان شمارنده حلقه اضافه می‌شود. قسمت

قسمت میانی پرانتز که بین دو علامت ; قرار دارد مربوط به شرط حلقه ماست. از آنجایی که می‌خواهیم ده مرتبه اعداد مورد نظر کاربر را دریافت نماییم، نیاز داریم که این حلقه، ده مرتبه تکرار شود. مقدار اولیه شمارنده ما ۱ است و پس از هر بار اجرا شدن حلقه یک واحد افزوده می‌شود. بنابراین کافیست که تا رسیدن i به عدد ۱۰، این حلقه ادامه پیدا کند.

به بیانی دیگر، دستور for در قطعه کد بالا می‌گوید:

- شمارنده این حلقه i نام دارد و مقدار اولیه آن ۱ است. (معرفی شمارنده حلقه و مقدار دهی آن)
- پس از هر بار اجرای حلقه یک واحد به شمارنده افزوده شود. (گام حلقه)
- تا زمانی که i کوچکتر مساوی ده است، این حلقه ادامه پیدا کند.

در ادامه برنامه‌های FOR-1.C، FOR-2.C و FOR-3.C به عنوان نمونه برنامه‌های کاملی که با استفاده از دستور for پیاده سازی شده‌اند، آورده شده است.

❋ برنامه FOR-1.C ❋

```
//This program will take 10 numbers and print their sum and average in the output.
#include<stdio.h>
#include<conio.h>
void main()
{
int i,num,sum;
float ave;
sum=0;
//************
clrscr();
printf("enter your numbers (10 times):\n");
for(i=1;i<11;i++)
        {
        scanf("%d",&num);
        sum=num+sum; //sum+=num can also be used.
        }
printf("Your sum is: %d",sum);
ave=(float)sum/10;
printf("\nYour average is %f\n", ave);
getch();
}
```

برنامه فوق با استفاده از حلقه for، ده عدد را از کاربر گرفته و میانگین آنها را محاسبه و چاپ می‌نماید.

❊ برنامه FOR-2.C ❊

```
//This program will take some numbers (the user decides how many numbers to enter) and calculate their sum and average.
#include<stdio.h>
#include<conio.h>
void main()
{
int n;
int i,num,sum;
sum=0;
float ave;
//If compile this program with .C format, the compiler will given an error for declaring "ave" after assigning a value for "sum".
//This is the mentioned error: "Declaration is not allowed here"
ave=0;
//************
clrscr();
printf ("How many numbers do you want to enter?\n");
scanf("%d",&n);
printf("Enter your numbers (%d times):\n",n );
for(i=1;i<=n;i++)
        {
        scanf("%d",&num);
        sum= num+sum;
        }
printf("Your sum is: %d",sum);
ave=(float)sum/n;
printf("\nYour average is %f\n", ave);
getch();
}
```

برنامه فوق، مشابه برنامه قبلی تعدادی عدد را از کاربر دریافت می‌کند و میانگین آنها را محاسبه می‌کند با این تفاوت که تعداد اعداد ورودی به دلخواه کاربر مشخص می‌شود. در ابتدای برنامه تعداد اعداد ورودی از کاربر خوانده می‌شود و در متغیر n ذخیره می‌گردد. سپس حلقه for، n مرتبه تکرار می‌شود تا n عدد مورد نظر کاربر دریافت شده و مجموع آنها محاسبه گردد. در انتهای برنامه هم مجموع محاسبه شده، تقسیم بر n شده و میانگین اعداد بر روی خروجی چاپ می‌شود.

❊ برنامه FOR-3.C ❊

```
//This program prints "Multiplication Table" on the screen.
//The program contains a for loop inside another for loop.
#include <stdio.h>
#include <conio.h>
void main()
 {
 int i,j,x;
 i=1;
 j=1;
 // *** The Main Body *** //
 clrscr();
 printf("This program prints the \"Multiplication Table\" on the screen.\nPress
any key to continue:\n\n");
 getch();
 for (i=1;i<=10;i++)
        {
        for (j=1;j<=10;j++)
                {
                x=i*j;
                printf("%d\t",x);
                }
        }
 printf("\n");
 getch();
 }
```

این برنامه با استفاده از دو حلقه تو در تو، جدول ضرب را در خروجی چاپ می‌نماید.

روال کار برنامه به این صورت است که پس از ورود به حلقه اصلی (حلقه بیرونی) برای اولین بار، مقدار شمارنده این حلقه که **i** نام دارد، برابر با ۱ می‌شود. سپس بلافاصله برنامه وارد حلقه درونی می‌شود. در این حلقه، شمارنده حلقه که **j** نام دارد از ۱ تا ۱۰ افزایش می‌یابد و در هر مرتبه اجرای حلقه، حاصل ضرب **i*j** پس از محاسبه در خروجی چاپ می‌شود. بنابراین در اولین مرتبه اجرای کامل حلقه درونی (پایان ده مرتبه اجرا)، حاصل ضرب اعداد ۱ تا ۱۰ در عدد ۱ در خروجی چاپ می‌شود.

برای فهم بهتر عملکرد برنامه، روال چاپ خروجی را دنبال می‌کنیم:

دور اول اجرای حلقه درونی (i=1):

i=1 j=1 i*j=1 i=1 j=2 i*j=2 ... i=1 j=10 i*j=10

دور دوم اجرای حلقه درونی (i=2):

i=2 j=1 i*j=2 i=2 j=2 i*j=4 ... i=2 j=10 i*j=20

دور سوم اجرای حلقه درونی (i=3):

i=3 j=1 i*j=3 i=3 j=2 i*j=6 ... i=3 j=10 i*j=30

...

دور دهم اجرای حلقه درونی (i=10):

i=10 j=1 i*j=10 i=3 j=2 i*j=6 ... i=3 j=10 i*j=30

در نهایت خروجی برنامه به صورت زیر خواهد بود:

```
This program prints the "Multiplication Table" on the screen.
Press any key to continue:

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

_
```

❋ حلقه بی‌نهایت ❋

می‌توان حلقه را به گونه‌ای تعریف کرد که برنامه هیچ گاه از این حلقه خارج نشود. چنین برنامه‌ای تا زمانی که توسط کلیدهای خاصی خاتمه پیدا نکند، ادامه خواهد داشت. یکی از کاربردهای حلقه‌های بی‌نهایت نوشتن سیستم عامل است. سیستم عامل در واقع برنامه بزرگی است که با روشن شدن سیستم شروع به کار می‌کند و تا خاموش نشدن سیستم ادامه دارد. همه برنامه‌های کاربردی در دل این برنامه بزرگ اجرا می‌شوند.

برای شکستن حلقه‌های بی‌نهایت در زمان اجرای برنامه، از ترکیب کلیدهای ctrl و break باید استفاده نمود. البته در بعضی از نسخه‌های C++، ctrl+c به جای ctrl+break مورد استفاده قرار می‌گیرد.

نحوه نوشتن حلقه بی‌نهایت برای دستورات for و while به این صورت است:

```
for (;;)                    while (1)
    {                       {
    دستورات حلقه            دستورات حلقه
    }                       }
```

در برنامه زیر یک حلقه بی‌نهایت با استفاده از دستور for پیاده سازی شده است:

❋ برنامه LOOP.C ❋

```
//Infinite Loop
#include <stdio.h>
void main()
 {
 int i=0;
 for(;;)
        {
        printf("%d\n",i);
        i++;
        }
 }
```

در این برنامه، متغیر i از ۰ تا حداکثر ظرفیت خود – که در نوع int عدد ۳۲۷۶۷ است – یکی یکی اضافه می‌شود و در هر مرتبه چاپ می‌شود. پس از آن اعداد ۳۲۷۶۸- تا ۱- چاپ شده و دوباره مقدار متغیر i صفر می‌شود. دلیل این مسأله این است که اگر عدد ۱۱۱۱۱۱۱۱۱۱۱۱۱۱۱ در مبنای دو که برابر با ۳۲۷۶۷- در مبنای ده است (دقت کنید که بیت سمت چپ، بر اساس قرارداد متغیر int در زبان C، بیت علامت است. یعنی بر طبق قرارداد، یک شدن این بیت معادل با منفی بودن عدد در مبنای ده است) را با یک جمع کنیم، حاصل صفر خواهد بود. چون بیت هفدهمی برای ذخیره رقم یکی که در نتیجه این جمع ظاهر می‌شود، نخواهیم داشت. روند افزوده شدن به i و چاپ آن، دائماً در این برنامه تکرار می‌شود.

❋ معرفی دستورات while و do while ❋

یکی دیگر از دستورات پیاده سازی حلقه‌ها در زبان C، دستور while است. ساختار این دستور به صورت زیر است:

```
while (شرط حلقه)
        {
        دستورات حلقه
        }
```

مشابه حلقه for، مجموعه دستورات داخل آکولاد تا نقض نشدن شرط حلقه، تکرار می‌شوند. حلقه while – همان طور که از ترجمه کلمه while پیداست – تا زمانی که شرط حلقه برقرار باشد ادامه می‌یابد. تفاوت این دستور با حلقه for در این است که در ساختار while متغیری به عنوان شمارنده حلقه، تعریف نشده است. برنامه نویس می‌تواند در صورت نیاز به شمارش دفعات تکرار حلقه، به صورت دستی در دل حلقه، شمارنده‌ای را کم یا زیاد کند. برای این منظور، حلقه را به این نحو می‌نویسیم:

```
while (شرط حلقه)
        {
        دستورات حلقه
        i++;
        }
```

قطعه کد زیر نمونه‌ای از ایجاد حلقه به کمک این دستور است:

```
int sum=0;
scanf("%d",&x);
while (x!=0)
        {
        scanf("%d",&x);
        sum=sum+x;
        }
printf("%d",sum);
```

در این قطعه کد، ابتداً عددی از ورودی خوانده می‌شود و در متغیر x ذخیره می‌گردد. سپس تا زمانی که x مخالف صفر باشد، کاربر می‌تواند اعداد مورد نظر خود را وارد نماید. به محض وارد شدن عدد صفر توسط کاربر، شرط حلقه نقض شده و برنامه از حلقه خارج می‌گردد. سپس مجموع اعداد وارد شده برای کاربر نمایش داده می‌شود. در واقع x=0 شرط خاتمه حلقه فوق است.

دستور do while هم دقیقاً مشابه دستور while عمل می‌کند با این تفاوت که شرط حلقه در انتهای آن بررسی می‌شود. بنابراین تنها تفاوت دستور do while با while در اینجاست که حلقه‌های پیاده شده با دستور do while حداقل یک مرتبه اجرا می‌شوند چرا که شرط حلقه در انتهای آن چک می‌شود.

فرم کلی دستور do while به صورت زیر است:

```
do
        {
        دستورات حلقه
        } while (شرط حلقه )
```

قطعه کد زیر معادل قطعه کد صفحه گذشته است که با استفاده از دستور do while پیاده شده است.

```
int sum=0;
scanf("%d",&x);
do
        {
        scanf("%d",&x);
        sum=sum+x;
        } while (x!=0)

printf("%d",sum);
```

تفاوت این قطعه کد با مثال قبلی در اینجاست که اگر اولین عدد وارد شده توسط کاربر صفر باشد، پس از ورود برنامه به حلقه و اجرای دستورات داخل آن، نقض شدن شرط حلقه معلوم شده و برنامه از حلقه خارج می‌گردد؛ بر خلاف قطعه کد مثال گذشته که شرط حلقه در ابتدای ورود به آن بررسی می‌گردید و در صورت برقرار نبودن شرط، برنامه وارد حلقه نمی‌شد.

در ادامه دو نمونه از برنامه‌های نوشته شده با دستور while و do while آمده‌اند.

❋ برنامه WHILE-1.C ❋

```
//This program calculates the square of the given numbers (for 10 numbers) and add them with each other.
//It can easily be converted to a series that calculates the sum of squares for natural numbers.
#include <stdio.h>
#include <conio.h>
void main()
 {
 // *** Variable Declaration ***
 int x,i;
 long int y,sum;
 i=10;
 sum=0;
 // *** The Main Body ***
 clrscr();
 printf("This program takes 10 numbers and calculates the square of them. Each square will be added with the previos ones and the result will be shown.\n");
 while(i) //While(i) means this loop will continue until 'i' reaches 0. In other word this loop will continue until 'i' is true.
    {
    printf("\nEnter a number please: ");
    scanf("%d",&x);
    y=x*x;
    sum=sum+y;
    printf("\nThe square of your number is: %d\n",y);
    printf("Total Sum = %d\n",sum);
    i--;
    }
 printf("\n*** This is the end of the program ***");
 getch();
 }
```

این برنامه ۱۰ عدد را به عنوان ورودی از کاربر دریافت می‌کند و ضمن هر بار دریافت، مجذور عدد ورودی را محاسبه کرده و حاصل جمع آن با مجذور ورودی‌های گذشته را در متغیر sum ذخیره و نتیجه را چاپ می‌نماید. نکته قابل توجه در این مثال شرط حلقه می‌باشد. while(i) به این معناست که تا زمانی که i برقرار است (یعنی i مساوی با صفر نشده است) حلقه ادامه پیدا می‌کند.

❊ برنامه WHILE-2.C ❊

```
//This program contains a new command for the Printf function. More Explanation is given in line 12.
#include <stdio.h>
#include <conio.h>
void main( )
 {
 //*** Variable Declaration ***
 int count=0;
 char x,y;
 //*** The Main Body ***
 clrscr();
 printf("This program will count the number of characters in a given statement.\nNote that the program will continue until you enter \"0\".\n");
 //The \" command in the previous line, will print the quotation mark on the screen.
 printf("\nenter your statement please:\n");
 while (x!='0')
        {
        scanf("%c",&x);
        count++;
        }
 printf("\nThe number of characters in your statement is: %d",count);
 getch();
 /*
 The following lines will check the value of x after the end of the program.
 This test shows the function of scanf, while we enter a string instead of a single character. By entering a string, all the characters will be scanned and saved in the defined variable respectively (x in this program).
 But each time, the scanned character will be replaced by the next character of the string.
 */
 printf("\n\nChecking the value of variable x:\n");
 printf("%c",x);
 getch();
 }
```

این برنامه متنی را از کاربر دریافت می‌کند و تعداد کارکترهای آن را شمارش می‌نماید. شرط خاتمه حلقه دریافت کردن کارکتر "0" از ورودی است.

* ساختارهای تصمیم *

ساختارهای تصمیم در زبان C به برنامه نویس امکان قرار دادن شرط برای اجرا شدن بخش‌های دلخواهی از برنامه را می‌دهد. به عبارت دیگر، برنامه نویس با استفاده از این ساختارها، اجرای قسمت‌هایی از برنامه را منوط به برقراری شرایط خاصی می‌نماید. مثلاً می‌خواهیم برنامه‌ای بنویسیم که زوج یا فرد بودن عدد دریافت شده از کاربر را تشخیص دهد. برای این کار، باقی مانده تقسیم عدد دریافت شده بر ۲ را محاسبه می‌کنیم. اگر باقی مانده برابر با ۱ بود، عدد ورودی فرد بوده و الا این عدد زوج است. الگوریتم نوشتن این برنامه به صورت زیر است:

دریافت عدد ورودی از کاربر
محاسبه باقی مانده تقسیم این عدد بر ۲
اگر باقی مانده برابر با یک شد:
    چاپ شود "این عدد فرد می‌باشد"
والا چاپ شود:
    "این عدد زوج می‌باشد"

در این الگوریتم، کلمات "اگر" و "والا" همان دستوراتی هستند که شروطی را برای اجرای دستورات ذیل خود معین می‌کنند.

ساختارهای تصمیم در زبان C با دستورات زیر قابل پیاده سازی هستند:

- دستور if
- دستور switch

* دستور if *

فرم کلی این دستور به صورت زیر است:

```
if (شرط)
        {
        دستورات مربوط به شرط
        }
else
        {
        سایر دستورات
        }
```

کلمه if به معنای "اگر" و کلمه "else" معادل عبارت "در غیر این صورت" است. توجه شود که نوشتن دستور else ضروری نیست و می‌توان بدون استفاده از آن، شرط مورد نظر را برای برنامه نوشت.

اگر بخواهیم یک مسأله چند حالته را به صورت شرطی بنویسیم، می‌توانیم از if های پشت سر هم یا از دستور else if استفاده کنیم. شکل کلی استفاده از دستور else if به صورت زیر می‌باشد:

```
if (حالت اول رخ داد)
        {
        دستورات مربوط به حالت اول
        }
else if (حالت دوم رخ داد)
        {
        دستورات مربوط به حالت دوم
        }
else if (حالت سوم رخ داد)
        {
        دستورات مربوط به حالت سوم
        }
        ...
else
        {
        دستورات مربوط به زمانی که هیچ یک از حالت‌های فوق رخ ندهد
        }
```

در برنامه IF-1.C که در ادامه این بخش آمده است، با استفاده از دستور else if، حالت‌های مختلف یک عدد از لحاظ تعداد ارقام، بررسی شده است.

برای پیاده کردن شرط‌های وابسته به یکدیگر، می‌توان از if های تو در تو استفاده نمود. به مثال زیر توجه فرمایید:

می‌خواهیم برنامه‌ای بنویسیم که با استفاده از if های تو در تو، اعدد بخش پذیر بر ۶ را تشخیص دهد. می‌دانیم که اعداد بخش پذیر بر ۶ هم بر ۲ و هم بر ۳ بخش پذیر هستند. می‌توانیم با اضافه کردن یک شرط به شرط مثال قبل (تشخیص زوج یا فرد بودن اعداد)، به مقصود خود دست یابیم. به این ترتیب که اگر عدد وارد شده زوج باشد، باقیمانده آن بر عدد ۳ محاسبه گردد. اگر این مقدار برابر با صفر بود، عدد ورودی هم زوج است و هم بر سه بخش پذیر. به عبارت دیگر عدد وارد شده بر ۶ بخش پذیر است. شکل کلی پیاده سازی این دو شرط تو در تو در صفحه آینده آمده است:

```
if (باقی مانده عدد ورودی بر ۲ برابر با صفر شد)
        {
        if (باقی مانده عدد ورودی بر ۳ برابر صفر شد)
                چاپ شود "این عدد بر ۶ بخش پذیر است"
        }
```

برنامه‌های IF-1.C و IF-2.C که در ادامه می‌آیند، نمونه برنامه‌هایی هستند که با استفاده از دستور if نوشته شده‌اند.

*** برنامه IF-1.C ***

```
//This is a simple program to see the function of if and else command.
//The program takes a number and tells how many digits it has.
#include <stdio.h>
#include <conio.h>
void main()
 {
 long int x;
 x=0;
 // *** The Main Body *** //
 clrscr();
 printf("Enter a number between 0 and 30000 please:\n");
 scanf("%d",&x);
 if(x>=0 && x<10)
        printf("Your number has 1 digit.");
 else if(x>=10 && x<100)
        printf("Your number has 2 digits.");
 else if(x>=100 && x<1000)
        printf("Your number has 3 digits.");
 else //This else relates to the last written if  (if x>=100 && x<1000)
        printf("Your number has more than 3 digits.");
 getch();
 }
```

این برنامه ابتداً عددی را از کاربر دریافت می‌کند. سپس با مقایسه این عدد با بازه اعداد یک رقمی (اعداد بین صفر تا ۱۰)، دو رقمی (اعداد بین ۱۰ تا ۱۰۰)، سه رقمی (اهداد بین ۱۰۰ تا ۱۰۰۰) و بیشتر (اعداد بزرگتر مساوی ۱۰۰۰)، تعداد ارقام این عدد را در خروجی نمایش می‌دهد.

دقت کنید که دستور else انتهایی مربوط به else if قبل از خود است و ارتباطی با else if های پیشین یا if ابتدایی ندارد.

*** برنامه IF-2.C ***

```
//This program counts the number of words and characters with and without spaces.
//This program contains a new command for the Printf function. More Explanation is given in line 14.
#include <stdio.h>
#include <conio.h>
void main( )
 {
 //*** Variable Declaration ***
 int count=0;
 int word=1;
 char x,y;
 //*** The Main Body ***
 clrscr();
 printf("This program will count the number of characters in a given statement.\nNote that the program will continue until you enter \"0\".\n");
 //The \" command in the previous line, will print the quotation mark on the screen.
 printf("\nenter your statement please:\n");
 while (x!='0')
        {
        scanf("%c",&x);
        count++;
        if (x==' ')
                {
                ++word;
                }
        }
 printf("\nThe number of words in your statement is: %d",word);
 printf("\nThe number of characters (with spaces) in your statement is: %d",count);
 printf("\nThe number of characters (without spaces) in your statement is: %d",count-word+1);
 getch();
 /*
 The following lines will check the value of x after the end of the program.
 This test shows the function of scanf, while we enter a string instead of a single character.
```

```
By entering a string, all the characters will be scanned and saved in the defined variable respectively (x in this program).
But each time, the scanned character will be replaced by the next character of the string.
*/
printf("\n\nChecking the value of variable x:\n");
printf("%c",x);
getch();
}
```

این برنامه پس از دریافت یک متن از کاربر، تعداد کارکترهای داخل متن با احتساب فاصله‌ها، تعداد کارکترها بدون احتساب فاصله‌ها و تعداد کلمات موجود در متن را شمارش می‌نماید و روی خروجی نمایش می‌دهد. این عملیات، معادل دستور word count در نرم افزار Microsoft Word است. شرط خاتمه این برنامه هم ورود کارکتر "0" است.

نحوه شمارش کلمات با استفاده از دستور if پیاده سازی شده است. به این صورت که در هر بار فشار دادن کلید توسط کاربر، کارکتر ورودی با کارکتر "space" (فاصله) مقایسه می‌شود. در صورت مساوی بودن این دو، یک واحد به شمارنده تعریف شده افزوده می‌شود. مساوی بودن کارکتر ورودی با کارکتر فاصله به این معناست که کاربر، کارکتر فاصله را وارد کرده است. به عبارت دیگر، نوشتن کلمه قبلی به اتمام رسیده و کاربر می‌خواهد کلمه جدیدی را وارد نماید. پس از این که کاربر متن مورد نظر خود را وارد نمود، این شمارنده حاوی «تعداد کارکترهای فاصله وارد شده در متن» یا به عبارت دیگر «تعداد کلمات متن منهای یک» است.

البته این برنامه دارای اشکالات اندکی نیز می‌باشد. از جمله اینکه رفتن به سطر بعدی با استفاده از کلید "enter"، به معنای خاتمه کلمه قبلی محسوب نمی‌شود یا اگر کاربر بیش از یک فاصله بین دو کلمه وارد نماید، تعداد کلمات متن، به تعداد فاصله‌های اضافی، افزایش می‌یابد. اما از آنجا که غرض از نوشتن این برنامه، استفاده از دستور if بوده است، این جزئیات در بهینه کردن خروجی اعمال نشده است. می‌توان با اضافه کردن چند دستور ساده، برنامه‌ای بدون ایراد یا اصطلاحاً باگ (bug) ارائه نمود.

درادامه، برنامه‌های کاربردی FACT.C و SERIES.C به عنوان نمونه برنامه‌هایی که با استفاده از ساختارهای تکرار و تصمیم نوشته شده‌اند، مورد بررسی قرار گرفته‌اند.

❊ برنامه SERIES.C ❊

```
/*
This program will calculate the following series:
1+(1/2)+(1/4)+(1/8)+(1/16)+...
Also the program contains a new command which determines the number of
digits to be printed in the output in a float variable.
More explanation in line 16
*/
#include <stdio.h>
#include <conio.h>
void main()
 {
 int count;
 int num=6;
 //num tells how many terms you want to calculate in this series.
 //num can be entered by user by using scanf command.
 float sum, x;
 clrscr();
 for (sum=0, x=1.0, count=1; count<=num; count++, x*=2)
        {
        sum+=1/x;
        printf("sum=%7.4f when count=%d\n", sum, count);
        /* %7.4f means the output will contain 6 digits plus '.' (point).
        Four of this digits are reserved for the right side of the point.
        The other two, will be used before the point.
        Example: 46.5676
        There are 7 characters in this example: 2 (right side digits) + point + 4
(left side digits)
        When we type %7.4f, if the value of our variable contains more digits,
only the first four digits in the right side of the point will be shown.
        All of the digits before the point will be shown in the output.
        Example: sum=121.123456; By writing printf("%7.4f", sum), 121.1234
will be printed in the output.
        */
        }
 getch();
 }
```

این برنامه، ۶ جمله اول از سری ...+(1/8)+(1/4)+(1/2)+1 را محاسبه می‌نماید.

نحوه عملکرد برنامه به این صورت است که .... (در حال تکمیل است)

با اندک تغییری در این برنامه، می‌توان مجموع تعداد دلخواهی از جملات این سری را محاسبه نمود. کافیست عددی را به عنوان تعداد جملات مورد نیاز کاربر، از ورودی خوانده و آن را در متغیری ذخیره نماییم. سپس شرط حلقه for را تا رسیدن مقدار شمارنده حلقه به آن عدد، تعیین نماییم.

❊ برنامه FACT.C ❊

```
//This program takes a number and calculate its factorial.
#include <stdio.h>
#include <conio.h>
void main()
 {
 int n;
 int i;
 int x=1;
 // *** The Main Body *** //
 clrscr();
 printf("Enter a number please, the program will give back its factorial.\n");
 scanf("%d",&n);
 if(n==0)       //There is no 0! so the program gives back an error message
        {
        printf("0! : error");
        goto end;
        }
 for(i=1; i<=n; i++)
        {
        x=i*x;
        }
 printf("%d! = %d",n,x);
 end: getch();
 }
```

* دستور break *

کلمه break به معنای شکستن است. با استفاده از این دستور می‌توان از یک حلقه خارج شد یا اصطلاحاً آن حلقه را شکست.

قالب استفاده از این دستور به شکل زیر است:

```
while (شرط)
        {
        دستورات داخل حلقه
        if (شرط خاصی بر قرار بود)
                break;
        ادامه دستورات حلقه (در صورت وجود)
        }
```

توجه داشته باشید که دستور break برای تمامی حلقه‌ها قابل استفاده است و اختصاصی به دستور while ندارد. همچنین این دستور باید داخل حلقه نوشته شود و خارج از حلقه معنایی ندارد.

با استفاده از این دستور می‌توان از حلقه‌های بی‌نهایت هم خارج شد. به این ترتیب حلقه‌های بی‌نهایت کاربرد عملیاتی پیدا می‌کنند. مثلاً برای خروج از سیستم عامل که یک حلقه بی نهایت است، می‌توان طوری برنامه نویسی نمود که با نوشتن یک دستور به خصوص یا انتخاب گزینه خروج در محیط سیستم عامل، برنامه سیستم عامل خاتمه یابد. نحوه پیاده سازی این عملیات اجمالاً به صورت زیر است:

```
شروع برنامه سیستم عامل
while (1)
        {
        دستورات سیستم عامل
        if  (گزینه خروج انتخاب شد)
                break;
        }
دستورات خاتمه سیستم عامل
```

در مثال فوق، تا زمانی که گزینه خروج فعال نشده است، دستورات سیستم عامل مداوماً اجرا می‌شوند. به محض انتخاب گزینه خروج، دستور break، برنامه را از حلقه بی‌نهایت سیستم عامل خارج می نماید و دستورات خاتمه سیستم عامل اجرا می شوند.

قطعه کد زیر، کاربرد این دستور را به صورت عملی نمایش می دهد:

```
while (1)
        {
        x=getche();
        if (x=='#')
                {
                printf("\n---The program will end now---");
                break;
                }
        }
getch();
```

این قطعه کد یک محیط تایپ را برای کاربر فراهم می نماید. دستور getche() ضمن اینکه کارکتری را از ورودی می خواند، آن را روی صفحه نمایش نشان می دهد. از آنجایی که در یک حلقه بی نهایت که با استفاده از دستور while (1) پیاده شده است، دستور getche() مداوماً اجرا می شود، یک محیط تایپ برای کاربر فراهم می گردد. با استفاده از دستور break، هنگامی که کاربر کلید # را وارد نماید، حلقه بی نهایت while شکسته شده و برنامه خاتمه می یابد.

❊ دستور continue ❊

مشابه دستور break، این دستور هم مربوط به حلقه‌های تکرار است. هنگامی که برنامه نویس از این دستور در حلقه استفاده می نماید، برنامه دستورات بعدی حلقه را نادیده گرفته و به اولین خط حلقه منتقل می شود.

فرض کنیم می خواهیم تعداد کارکترهای یک متن را بدون در نظر گرفتن کارکترهای "space" بشماریم. قطعه کد زیر با استفاده از دستور continue، این عملیات را پیاده سازی می‌کند.

```
while (x!='#')
        {
        x=getche();
        if (x==' ')
                continue;
        count++;
        }
```

فصل چهارم – توابع

عناوین مطالب:

- برنامه نویسی ساخت یافته
- توابع در زبان C

* برنامه نویسی ساخت یافته *

به زبان ساده ساخت یافتگی در برنامه نویسی به معنای تقسیم کردن برنامه به بلوک های مجزا از هم است. برای فهم بهتر این موضوع بنایی را در نظر بگیرید که اجزای سازنده آن جداگانه آماده شده اند. مثلاً یک خانه اسکیمویی! قطعات یخی سازنده این خانه از پیش ساخته شده اند. هر یک از بلوک ها وظیفه مربوط به خود را انجام می دهند.

۱- نوشتن برنامه به صورت ساخت یافته مراحل مختلفی دارد. در مرحله اول، پس از اینکه برنامه نویس الگوریتم کلی برنامه خود را تصور نمود، آن را به بخش های کوچکتری تقسیم می کند. لازم نیست که در همان ابتدا تمام این بخش ها و جزئیات برنامه را معلوم کند. همین که برنامه نویس اجمالاً بعضی از دستورات و ساختارهای مورد نیاز خود برای نوشتن برنامه اصلی را تصور می کند، کفایت می کند. مثلاً برای نوشتن برنامه محاسبه فاکتوریل اعداد به یکی از ساختارهای تکرار (حلقه‌ها) احتیاج داریم تا بتوانیم ضرب اعداد متوالی را پیاده سازی کنیم.

۲- پس از این تقسیم بندی، در مرحله دوم، برنامه نویس به راحتی تمرکز خود را معطوف نوشتن قسمت های کوچک‌تر برنامه می کند. به عنوان نمونه در مثال محاسبه فاکتوریل، در قسمت تعریف متغیرها، متغیرهای مورد نیازی که فعلاً به ذهنش می رسد را تعریف می کند؛ سپس کدهای مربوط به برقراری ارتباط با کاربر و گرفتن ورودی را می نویسد؛ بعد از آن تمرکز خود را به نوشتن حلقه محاسبه کننده فاکتوریل اختصاص می دهد؛ در پایان هم قطعه کد مربوط به چاپ مقدار محاسبه شده را پیاده سازی می کند.

لزومی ندارد که مراحل نوشتن به ترتیب فوق باشند. ممکن است برنامه نویس ترجیح دهد همان ابتدا تکلیف قسمت اصلی برنامه خود – که در مثال بالا نوشتن حلقه مربوط به محسابه فاکتوریل بود – را مشخص کند.

۳- بخش اصلی رعایت ساخت یافتگی در نوشتن برنامه ها، مربوط به استفاده از توابع است... (در حال تکمیل)

۴- همان طورکه برنامه نویس در طول پیاده سازی برنامه، ساخت یافتگی را رعایت می کند، بهتر است که در هنگام نوشتن کدهای مربوط، با رعایت برخی از نکات، نوشتار خود را هم ساخت یافته کند. این مطلب کمک فراوانی به منظم شدن و خوانایی برنامه (فهم بهتر برنامه) می کند.
برخی از این نکات در ادامه ذکر شده اند.

جلوتر بودن نقطه شروع مجموعه دستورات داخل حلقه ها یا ساختارهای تصمیم مانند مثال زیر:

```
if (i<n)
        {
        sum+=num;
        i++;
        printf("%d\n",i);
        }
printf("final sum = %d",sum);
```

جلوتر بودن دستورات داخل توابع از عنوان اصلی تابع، مانند نمونه زیر:

```
void main()
 {
 int x;
 char y;
 x=0;
 y=getch();
 …
 }
```

استفاده از کامنت برای توضیح برنامه

تعریف و مقدار دهی متغیرها در ابتدای بدنه توابع

الگوی زیر نمونه ای از ساخت یافتگی در نوشتن برنامه است:

فراخوانی سرآیندهای مورد نیاز //

دستورات مربوط به فراخوانی سرآیندها

تعریف توابع //

دستورات مربوط به تعریف توابع

تعریف متغیرها //

دستورات مربوط به تعریف متغیر ها و مقدار دهی اولیه

بدنه اصلی برنامه //

دستورات مربوط به برقراری ارتباط با کاربر برنامه و گرفتن ورودی های لازم (ازجمله printf و scanf)

اعمال عملیات های لازم روی ورودی ها (مثلاً محاسبه فاکتوریل، محاسبه مجموع و میانگین گیری)
یا
استفاده از ساختارهای تکرار و تصمیم (مثل حلقه for یا دستور if)
یا
فراخوانی توابع مورد نیاز

چاپ خروجی

لازم به ذکر است که برنامه نویس می تواند هیچ یک از مراحل فوق را طی نکند. رعایت ساخت یافتگی باعث نظم ذهنی و راحتی کار خود برنامه نویس است، به خصوص در برنامه های بزرگ.