

بامتعال



دانشگاه صنعتی شریف

دانشکده‌ی مهندسی کامپیوتر

آموزش مبانی کامپیوتر و برنامه‌سازی به زبان C

تهریه و تنظیم:

یحیی پورسلطانی

(دانشجوی دکترای تخصصی رشته‌ی مهندسی کامپیوتر – نرم افزار)

۱۴۰۲ پاییز

اللّٰهُ الرَّحْمٰنُ الرَّحِيْمُ

به یاد دوست فقیدم

مهندس محمد میلاد ناصری

فارغ التحصیل دانشکده‌ی مهندسی کامپیوتر دانشگاه صنعتی شریف

فهرست مطالب

۱	درس ۱ : مفاهیم اولیه.....
۱	۱-۱. مقدمه
۲	۱-۲. وجود رایانه‌ها، چه مزایایی را دارد؟.....
۲	۱-۳. ساختار یک کامپیوتر
۳	۱-۳-۱. نرم‌افزار
۴	۱-۳-۲. سخت‌افزار
۵	۱-۴. مقدمه‌ای بر برنامه‌نویسی
۶	۱-۴-۱. چرا لازم است برنامه‌نویسی را بیاموزیم؟.....
۶	۱-۴-۲. چگونه یک برنامه را بنویسیم؟.....
۶	۱-۴-۳. فقط علوم کامپیوتری‌ها باید برنامه‌نویسی یاد بگیرن؟.....
۷	۱-۵. انواع زبان‌های برنامه‌نویسی
۷	۱-۵-۱. زبان ماشین
۸	۱-۵-۲. زبان اسembلی
۹	۱-۵-۳. زبان‌های برنامه نویسی سطح بالا
۹	۱-۶. ترجمه‌ی زبان‌های برنامه نویسی سطح بالا به زبان ماشین
۱۰	۱-۶-۱. ترجمه‌ی آنلاین (تفسیر)
۱۰	۱-۶-۲. ترجمه آفلاین (کامپایل)
۱۰	۱-۶-۳. ترجمه ترکیبی
۱۱	۱-۷. سلسله مراتب ذخیره‌سازی داده‌ها
۱۲	۱-۸. حساب رایانه‌ای
۱۳	۱-۸-۱. ارزش مکانی
۱۴	۱-۸-۲. تبدیل مبناهای
۱۶	۱-۸-۳. نمایش اعداد در مبناهای دیگر
۱۶	۱-۸-۴. آشنایی با اعداد مبنای ۱۶ و روش‌های تبدیل آن
۱۹	۱-۹. مراحل نوشتن و اجرای یک برنامه
۱۹	۱-۹-۱. محیط ایجاد یکپارچه (IDE)
۱۹	۱-۹-۲. مراحل ترجمه و اجرای یک برنامه‌ی نوشته شده به زبان C
۲۱	۱-۹-۳. ساخت یک پروژه ساده به زبان C

۱۰-۱. مراحل ایجاد یک پروژه به زبان C	۲۱
۱۰-۱. اجزای یک برنامه‌ی نوشته شده به زبان C	۲۵
۱۰-۱. کامپایل و اجرای برنامه در ابزار Code Blocks	۲۶
درس ۲ : آغاز برنامه‌نویسی	۲۹
۱-۲. عملیات ورودی و خروجی در زبان C	۲۹
۱-۲. چاپ خروجی	۲۹
۱-۲. دریافت ورودی	۳۰
۲-۲. متغیرها	۳۲
۲-۲-۲. تعریف کردن متغیر	۳۲
۲-۲-۲. مقداردهی به متغیرها	۳۳
۲-۲-۲. تبدیل نوع (Type Cast)	۳۴
۲-۲-۲. نکاتی در خصوص متغیرها	۳۵
درس ۳ : الگوریتم‌ها و محاسبات منطقی	۳۷
۳-۳. مقدمه حل مسئله با استفاده از الگوریتم‌ها	۳۷
۳-۳. ویژگی‌های الگوریتم‌ها	۳۸
۳-۳. انواع دستورات در الگوریتم‌ها	۳۸
۳-۳. روش‌های بیان الگوریتم‌ها	۳۸
۴-۳. روش اول : شیه کد	۳۸
۴-۳. روش دوم : فلوچارت (روندنما)	۳۹
۴-۳. مروری بر محاسبات منطقی	۴۲
درس ۴ : ساختارهای کنترلی	۴۳
۴-۴. تصمیم‌گیری و دستورات شرطی	۴۴
۴-۴-۱. ساختار تصمیم‌گیری if	۴۴
۴-۴-۲. ساختار کنترلی switch – case	۴۶
۴-۴. ساختارهای تکراری	۴۷
۴-۴-۱. تکرار با تعداد دفعات نامشخص (حلقه‌ی while)	۴۷
۴-۴-۲. تکرار با تعداد دفعات مشخص	۴۸
۴-۴-۳. ساختاری تکراری do-while	۵۰
۴-۴-۳. دستورات break و continue	۵۱
درس ۵ : آرایه‌ها	۵۲
۵-۵. معرفی آرایه‌ها	۵۳
۵-۵. تعریف آرایه‌ها در زبان C	۵۴
۵-۵. مقداردهی اولیه به آرایه‌ها	۵۵

۴-۵	رشته‌ها، آرایه‌ای از کاراکترها	۵۸
۵-۵	آرایه‌های دوبعدی و چند بعدی	۵۸
۵-۵-۱.	نحوه دسترسی به عناصر آرایه‌ی دوبعدی	۵۹
درس ۶ : توابع		۶۱
۶-۶	انواع دستورات در کامپیوتر	۶۱
۶-۶-۲.	مزایای استفاده از توابع	۶۲
۶-۶-۳.	تعریف و به کارگیری تابع در زبان C	۶۲
۶-۶-۳-۱.	گام اول: معرفی الگوی تابع (Function Prototype)	۶۳
۶-۶-۳-۲.	گام دوم: تعریف دستورات تابع	۶۴
۶-۶-۳-۳.	گام سوم: فراخوانی تابع	۶۵
۶-۶-۴.	میدان دید (Scope)	۶۶
۶-۶-۵.	پشته‌ی فراخوانی (Call Stack)	۶۷
۶-۶-۶.	فراخوانی با ارجاع و فراخوانی با مقدار	۶۹
۶-۶-۷.	بازگشت (Recursion)	۷۰
۶-۶-۷-۱.	بازگشت یا تکرار؟	۷۱
درس ۷ : اشاره‌گرها		۷۲
۷-۷	مقدمه‌ای بر اشاره‌گرها	۷۳
۷-۷-۱.	عملگرهای & و *	۷۵
۷-۷-۲.	کاربردهایی از اشاره‌گرها	۷۶
۷-۷-۳.	چند نکته در خصوص تعریف اشاره‌گرها	۷۶
۷-۷-۴.	مثال‌های بیشتر	۷۶
۷-۷-۴-۱.	طرز کار مرتب‌سازی حبابی	۷۷
۷-۷-۵.	ثبت کردن اشاره‌گر و مقادیر آن	۷۹
۷-۷-۶.	اشارة‌گرها و ارتباط آن‌ها با آرایه‌ها	۸۰
۷-۷-۷.	اعمال حسابی بر روی اشاره‌گرها	۸۰
درس ۸ : کاراکترها و رشته‌ها		۸۲
۸-۸	مقدمه‌ای بر رشته‌ها و کاراکترها	۸۳
۸-۸-۲.	کتابخانه‌هایی برای کار با رشته‌ها و کاراکترها	۸۴
۸-۸-۱.	پردازش ورودی و خروجی با استفاده از کتابخانه stdio	۸۴
۸-۸-۲.	پردازش کاراکترها با کتابخانه ctype	۸۵
۸-۸-۳.	کتابخانه String	۸۵
درس ۹ : ساختارها		۸۹
۹-۹	ساختار چیست؟	۸۹

۹۰	۲-۹. کاربرد ساختارها
۹۰	۳-۹. تعریف یک ساختار
۹۱	۴-۹. مقداردهی و به کارگیری ساختارها
۹۲	۵-۹. ارسال ساختار بهتابع
۹۲	۶-۹. آرایه‌ای از ساختارها
۹۳	۷-۹. تعریف نوع داده جدید با دستور <code>typedef</code>
۹۵	درس ۱۰ : ذخیره و بازیابی در فایل‌ها.....
۹۵	۱۰-۱. فایل چیست و چه ضرورتی دارد که از آن استفاده کنیم؟.....
۹۵	۱۰-۲. انواع فایل‌ها.....
۹۶	۱۰-۳. بازکردن فایل متنی و اشاره به آن
۹۷	۱۰-۴. نوشتن در فایل متنی.....
۹۷	۱۰-۵. خواندن از فایل متنی.....
۹۷	۱۰-۶. دیگر توابع کاربردی کار با فایل‌ها
۹۹	مراجع.....

پیش‌گفتار

امروزه به کارگیری رایانه‌ها در تمامی صنایع و رشته‌ها فراغیر شده است؛ بنابراین شاهد گسترش و رشد روز افزون نرم‌افزارهای کاربردی در حوزه‌های مختلف فنی مهندسی و علوم پایه هستیم؛ اگرچه این نرم‌افزارها می‌توانند نیازمندی‌های معمول مهندسین را برطرف کنند، اما ممکن است نتوانند راه حل کارآمدی را برای حل مسائل جدیدی که پیش‌تر برای آن‌ها راهکار رایانه‌ای ارائه نشده است، ارائه دهند؛ از سوی دیگر، محاسبات حوزه‌ی مهندسی و علوم پایه می‌بایست با سرعت و دقت بالاتری انجام گردد و این امروزه این امر بدون استفاده از رایانه‌ها مقدور نیست.

یادگیری اصول و مفاهیم برنامه‌نویسی مقدماتی کامپیوتر می‌تواند مهندسین و دانشمندان آینده را در ساخت نرم‌افزارهای مورد نیازشان و برای حل مسائل مورد نظرشان توانمند سازد و حتی این امکان را به ایشان بدهد که بتوانند قابلیت‌هایی جدید را به نرم‌افزارهای پیشین به منظور حل مسائل جدیدتر اضافه نمایند و در تکامل آن‌ها موثر باشند.

هدف درس مبانی برنامه‌نویسی، صرفاً آشنایی دانشجویان با یک زبان برنامه‌نویسی جدید نیست؛ بلکه هدف آن آموزش مفاهیم برنامه‌نویسی مقدماتی و مستقل از زبان برنامه‌نویسی است؛ بنابراین زبان برنامه‌نویسی C صرفاً به عنوان ابزاری به منظور تحقق بخشیدن به این مفاهیم است و مفاهیم قابل تعمیم به بسیاری از زبان‌های برنامه‌نویسی مرسوم دیگر نیز می‌باشد؛ بنابراین برنامه‌ی آموزشی این درس به گونه‌ای برنامه‌ریزی شده است که دانشجویان با فراگیری کامل مطالب آن، بتوانند دانشی را کسب نمایند که قابل تعمیم به سایر مسائل و زبان‌های برنامه‌سازی باشد.

این جزو، حاصل تدریس و بحث و بررسی درس مبانی کامپیوتر به زبان C در کلاس‌های حل تمرین این درس در دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران) و تدریس آن در کلاس‌های درس دانشگاه صنعتی شریف بوده و به تدریج تکمیل گشته است و به عنوان منبع کمکی در کنار کتاب درسی (کتاب دایتل) مورد استفاده قرار می‌گیرد.

در پایان، از تمامی اساتید ارجمند که در تمام این سال‌ها من را در آموزش مبانی برنامه‌نویسی راهنمایی کردند و تجربیات ارزشمند خود را قرار دادند سپاس گزارم؛ از جمله آقای دکتر محمد حسن شیرعلی شهرضا و آقای دکتر زاهد رحمتی از دانشگاه صنعتی امیرکبیر. همچنین از اساتیدی که محتواهای ارزشمندشان در شکل‌گیری محتواهای این نوشتار الهام‌بخش بوده سپاس گزارم؛ خصوصاً آقای دکتر کلامی هریس، آقای مهندس منوچهر بابایی و آقای مهندس وحید باقی از مجموعه‌ی فرادرس. همچنین از اساتید ارجمند دانشکده‌ی مهندسی کامپیوتر به خصوص آقای دکتر آیام و آقای دکتر فضلی به خاطر حمایت‌های ارزشمندشان سپاس گزارم از آقای دکتر جعفری رامیانی بابت طراحی پایه‌ی قالب این مستند و انتشار آن در سایت اداره ارتباط با صنعت دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران) سپاس گزارم و در نهایت، از پدر عزیزم که در میانه‌ی این مسیر، تجربیات ارزشمندشان را در اختیارم گذاشتند بسیار ممنونم. امیدوارم این نوشتار برای دانشجویان عزیز درس مبانی برنامه‌نویسی مفید باشد و مشتاقانه پذیرای نظرات ارزشمند ایشان و دیگر صاحب‌نظران هستم.

پیروز، شاد و تن درست باشید

یحیی پورسلطانی

۱۴۰۲ پاییز

درس ۱ : مفاهیم اولیه

در پایان این فصل از فرآگیران انتظار می‌رود با مفاهیم زیر آشنا شده باشند:

- با اجزای کامپیوتر (سخت‌افزار و نرم‌افزار) آشنا شود.
- ضرورت برنامه‌نویسی را درک کند.
- انواع زبان‌های برنامه‌نویسی را بشناسد.
- روش‌های ترجمه و تفسیر را درک کند.
- با مراحل اجرا و ترجمه‌ی یک برنامه آشنا شود.
- با دستگاه اعداد مبنای ۲ و ۱۶ آشنا شوند.
- با IDE ها و خصوصا Code Blocks IDE آشنا شوند.

۱-۱. مقدمه

در عصر جدید، کمتر کسی را می‌توان پیدا کرد که با رایانه، آشنایی نداشته باشد و یا حداقل یک مرتبه از این وسیله، استفاده نکرده باشد. رایانه‌ها، منحصر به کامپیوترهای شخصی نیستند و امروزه، تمام وسایل برقی، به نوعی به رایانه‌ها مرتبط هستند و به وسیله‌ی آن‌ها کنترل می‌شوند. این دستگاه‌ها، طیف وسیعی از کوچکترین دستگاه‌ها (میکروکامپیوترها) تا ابرکامپیوترها را دربر می‌گیرند و در کاربردهای مختلفی، ظاهر می‌شوند. به طور خلاصه، رایانه دستگاهی الکترونیکی است که داده‌هایی را از محیط بیرون دریافت کرده و بر مبنای دستورهای دریافتی، یک سری عملیات محاسباتی، مقایسه‌ای، منطقی، انتقالی، جابجایی و ... را انجام دهد. در طول درس، از واژه‌ی متدال کامپیوتر، بجای رایانه، استفاده خواهیم کرد.

۱-۲. وجود رایانه‌ها، چه مزایایی را دارد؟

قبل از ورود به دنیای رایانه‌ها، لازم است این سؤال را از خود بپرسیم: چرا رایانه‌ها ساخته شدند و وجود آن‌ها، چه مشکلاتی را حل کرده است؟ آیا می‌توان قابلیت‌های آن

مهم‌ترین ویژگی‌های یک کامپیوتر:

۱. سرعت
 ۲. دقت
 ۳. تکرار بدون نقص
 ۴. عدم فراموشی
- ها را تا حدی گسترش داد که بتوانند جایگزین انسان‌ها شوند؟ انسان‌ها، در طول یک روز، با کارهای تکراری و خسته‌کننده‌ای رویرو می‌شوند و گاهی اوقات، این خستگی می‌تواند منجر به بی‌دقیقی‌های متعددی شود. رایانه‌ها، مثل انسان‌ها دچار خستگی نمی‌شوند و در اثر خستگی، دچار خطأ نمی‌شوند. برای کارهای تکراری و در شرایط یکسان بسیار مناسب هستند. هم چنین، فرآخوانی و استفاده‌ی مجدد از اطلاعات ذخیره شده، با استفاده از کامپیوترها به مراتب، ساده‌تر خواهد بود.

مثال ۱: در خصوص تست تورینگ، جستجو و تحقیق کنید و شرحی مختصر از آن را بیان کنید. بر اساس این آزمایش، به کدام یک از محدودیت‌های یک کامپیوتر می‌توان پی برد؟

۱-۳. ساختار یک کامپیوتر

کامپیوترها، اجتماعی هستند از سخت‌افزار و نرم‌افزار. این دو بخش را می‌توان به مثابه جسم و روح یک کامپیوتر در نظر گرفت. همان‌طور که اجتماع روح و جسم، یک انسان زنده را می‌سازد، اجتماع سخت‌افزار و نرم‌افزار، یک کامپیوتر قابل استفاده را برای ما می‌سازد. بنابراین :

$$\text{کامپیوتر} = \dots + \dots$$

منظور از نرم افزار، همان برنامه‌ای است که از مجموعه‌ای از دستورالعمل‌ها تشکیل شده است و با استفاده از آن‌ها، می‌توان داده‌ها را پردازش نمود.

مثال ۲: با مراجعه به کتاب «مهندسی نرم افزار» و یا با جستجوی در اینترنت، تعریف دقیقی از نرم افزار ارائه دهید. (با ذکر مرجع)

.....
.....
.....
.....
.....
.....
.....

نرم افزار‌ها، انواع مختلفی دارند و طبقه بندی‌های متعددی برای انواع نرم افزار ارائه شده است. اما مهم‌ترین انواع نرم افزارها که در این درس با آن‌ها سروکار داریم عبارتند

از:

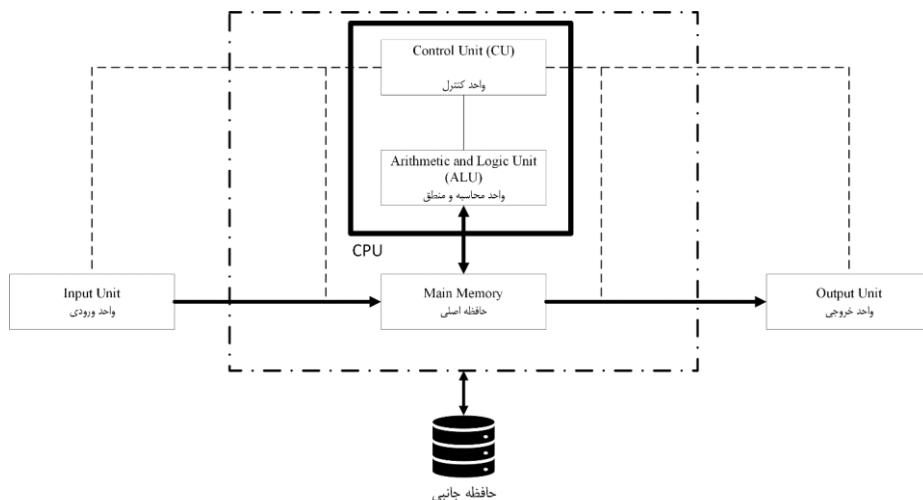
۱. سیستم‌های عامل
۲. نرم افزار‌های کاربردی و خدماتی
۳. کامپایلرها و مفسرها (در جلسات بعدی مورد بحث قرار می‌گیرند)

مثال ۳: وجود کدام یک از انواع نرم افزار‌های بالا، برای کارکرد کامپیوتر لازم و ضروری است و در صورت نبودن آن، قادر به استفاده از کامپیوتر نخواهیم بود؟ چرا؟

.....
.....
.....
.....
.....
.....

۲-۳-۱. سخت افزار

یک کامپیوتر دیجیتالی، حداقل از پنج واحد اساسی زیر تشکیل می‌شود:



شکل ۱. ساختار کلی یک کامپیوتر

نام واحد	کارکرد	مثال
ورودی (Input Unit)		
حافظه (Memory)		
محاسبه و منطق (ALU)		
خروجی (output)		
کنترل (CU)		

انواع حافظه عبارتند از

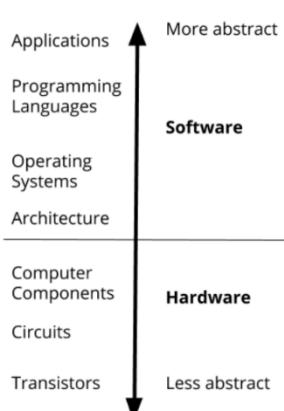
۱. حافظه اصلی

۲. حافظه جانبی (حافظه ثانویه)

۱-۴. مقدمه‌ای بر برنامه‌نویسی

معماری، به مجموعه‌ای از دستورات اشاره دارد که توسط نرم‌افزار به سخت‌افزار ارسال می‌شود و توسط سخت‌افزار، فهمیده می‌شود.

برنامه نویسی، عبارت است از انتقال نیاز و ایده به کامپیوتر و اجرای آن، توسط کامپیوتر. از طریق برنامه نویسی، ایده‌ی خود را به زبانی بیان می‌کند که برای کامپیوتر، قابل فهم باشد و کامپیوتر، بتواند آن را اجرا کند. برای این منظور، لازم است که یک زبان مشترک برای حرف زدن با کامپیوترها داشته باشیم؛ اما این کامپیوترها، هیچ زبانی را بجز زبان خودشان (یعنی زبان ماشین که صرفاً دنباله‌ای از صفر و یک است) متوجه نمی‌شوند. برای هر معماری موجود کامپیوتري، این زبان ماشین می‌تواند متفاوت باشد. این یعنی اینکه کامپیوترها، حتی زبان یک دیگر را نیز نمی‌فهمند!



۱. **برنامه نویسی، صرفاً به زبان ماشین:** برنامه‌ها، فقط با زبان صفر و یک نوشته می‌شوند و مهندس‌هایی بودند که فقط رشته‌های صفر و یک را بهم می‌بافتد.
۲. **زبان اسembلی:** برنامه نویسی را تا حدودی راحت‌تر کردند. اما بازهم دشوار بودند. چرا که برای یک کار ساده، می‌بایست چند خط برنامه نوشته می‌شد.
۳. **زبان‌های برنامه نویسی سطح بالا:** این طرز برنامه نویسی، به زبان انسان‌ها نزدیک‌تر شد.

مثال ۴: بنظر شما، سخت‌افزار و نرم‌افزار، چگونه با یک دیگر مرتبط می‌شوند؟

۱-۴. چرا لازم است برنامه‌نویسی را بیاموزیم؟

امروزه هیچ کس با کامپیوتر بیگانه نیست؛ به علل مختلفی مثل بازی، انجام امور اداری و... به کامپیوتر، نیاز داریم؛ به عبارت دیگر، همه‌ی ما در انجام امور روزمره به این وسیله نیازمندیم و به عنوان کاربر، از آن استفاده می‌کنیم؛ اما اگر بتوانیم به مرحله‌ای بررسیم که بتوانیم در جهت حل مسائل مورد نظرمان با کامپیوتر، برنامه بنویسیم و بر اساس نیاز خود، به آن دستور بدهیم، خواهیم توانست مسائل خود را به زبان کامپیوتر تبدیل کرده و آن را سریع‌تر حل کنیم.

۲-۴-۱. چگونه یک برنامه را بنویسیم؟

قبل از نوشتن یک برنامه‌ی کامپیوتری، لازم است بتوانید مسئله‌ی خود را به زبان کامپیوتر مدل‌سازی کنید و سپس، مدل خود را با یکی از زبان‌های برنامه‌نویسی، بیان کنید. تمرکز این درس، بر یادگیری توانایی مدل‌سازی مسائل به زبان کامپیوتر است. اما برای بیان مدل‌سازی، زبان C را انتخاب کرده‌ایم. اگر توانایی مدل‌سازی مسئله‌ی خود را داشته باشید، با هر زبان برنامه‌نویسی، خواهید توانست مسئله‌ی خود را بیان کنید.

۳-۴-۱. فقط علوم کامپیوتری‌ها باید برنامه‌نویسی یاد بگیرن؟

در تمامی رشته‌های تحصیلی حوزه‌های مهندسی، علوم پایه و حتی علوم پزشکی، با مسائل مختلفی روبرو هستیم. این مسائل می‌تواند انجام محاسبات سخت و یا تحلیل داده‌های حجیم، به منظور یک نتیجه‌گیری مهم باشد. همان‌طور که دیدیم، هدف برنامه‌نویسی، تبدیل یک نیازمندی به یک برنامه‌ی کامپیوتری است. بنابراین، تبدیل نیازمندی به یک برنامه‌ی قابل اجرا، در تمامی رشته‌ها می‌تواند به حل مسائل مختلف (بخصوص مسائلی که به محاسبات عددی و تحلیل داده‌ها مربوط می‌شوند، کمک خواهد کرد).

۱-۵. انواع زبان‌های برنامه‌نویسی

هر کامپیوتر، فقط زبان ماشین را متوجه می‌شوند و این زبان ماشین، می‌بایست در معماری سخت‌افزار آن‌ها، تعریف شده باشد. این زبان، صرفاً از اعداد صفر و یک تشکیل شده‌اند و فهم آن برای انسان، بسیار سخت و گنگ است. با توجه به اینکه این زبان، فاصله‌ی زیادی با زبان انسان دارد، یک زبان است.

۱-۵-۱. زبان ماشین

در اولین دوران اختراع کامپیوترها، برنامه‌ها فقط با زبان صفر و یک نوشته می‌شدند. (به عبارت دیگر، رشته‌های طویل صفر و یک بهم بافته می‌شدند).

مثال ۵: مثالی از یک برنامه ساده به زبان ماشین:

یک کامپیوتر ساده و پایه‌ای را در نظر بگیرید که یک حافظه‌ی اصلی با 2^{12} خانه و یک حافظه کمکی دارد (نام این حافظه‌ی کمکی را AC می‌گذاریم). بخشی از جدول دستورالعمل‌های این کامپیوتر، می‌تواند به شکل زیر باشد:

نام دستور	شرح دستور	کد دستوری
جمع	جمع کردن عدد ذخیره شده در یکی از خانه‌های حافظه (که آدرس آن داده شده است) با عدد ذخیره شده در حافظه AC و ذخیره سازی نتیجه در حافظه AC.	۰۰۱XXXXXXXXXXXX
بارگذاری	انتقال عدد ذخیره شده در یکی از خانه‌های حافظه و بارگذاری آن در AC.	۰۰۱۱XXXXXXXXXXXX

بنابراین، دستور زیر را در نظر بگیرید :

۰۰۰۱ ۰۰۰۰ ۰۰۱۰۰۱۰۰

طبق جدول بالا، دستور بالا به کامپیوتر دستور می‌دهد که مقدار خانه ۳۶ حافظه

اصلی را با مقدار ذخیره شده در حافظه کمکی ذخیره کن و نتیجه را در حافظه کمکی ذخیره کن.

۱-۵-۲. زبان اسembلی

درک زبان ماشین، به خصوص در برنامه‌های پیچیده‌تر، سخت و طاقت فرسا است. بنابراین، زبانی ابداع شد که به زبان انسان، کمی نزدیک‌تر شد و این زبان را، زبان اسembلی نامیدند. با ظهر زبان اسembلی، برنامه نویسی کمی راحت‌تر شد. اما این زبان، هنوز از زبان انسان‌ها، فاصله داشت و برای انجام یک کار ساده، می‌بایست چند خط برنامه نویسی صورت می‌گرفت. زبان اسembلی، به وسیله‌ی به زبان ماشین، ترجمه می‌شود.

مثالی ساده از یک نمونه زبان اسembلی : برنامه‌ی اسembلی زیر، اعداد ۱ الی ۹ را بر روی صفحه نمایش، چاپ می‌کند:

```
section .text
    global _start          ;must be declared for using gcc

_start:             ;tell linker entry point
    mov ecx,10
    mov eax, '1'

l1:
    mov [num], eax
    mov eax, 4
    mov ebx, 1
    push ecx

    mov ecx, num
    mov edx, 1
    int 0x80

    mov eax, [num]
    sub eax, '0'
    inc eax
    add eax, '0'
    pop ecx
    loop l1

    mov eax,1           ;system call number (sys_exit)
    int 0x80            ;call kernel
section .bss
num resb 1
```

جهت آشنایی بیشتر با زبان اسembلی و تغییر این مثال، به آدرس زیر مراجعه کنید:

https://www.tutorialspoint.com/compile_asm_online.php

همان‌طور که مشاهده کردید، اگرچه زبان اسembلی واضح‌تر از زبان است، برای یک عمل ساده (شمارش از ۱ تا ۱۰)، چندین خط برنامه نویسی نیاز بود. بنابراین، انسان‌ها به زبان‌هایی نیازمند شدند که شباهت بیشتری به زبان انسان داشته باشد. امروزه، عمدۀ ی

برنامه‌های کامپیوتری، با زبان‌های سطح بالا نوشته می‌شود.

همان‌طور که مشاهده کردید، اگرچه زبان اسمنبلی واضح‌تر از زبان ماشین است، اما برای یک عمل ساده (شمارش از ۱ تا ۱۰)، چندین خط برنامه نویسی نیاز بود. بنابراین، انسان‌ها به زبان‌هایی نیازمند شدند که شباهت بیشتری به زبان انسان داشته باشد. امروزه، عمده‌ی برنامه‌های کامپیوتری، با زبان‌های سطح بالا نوشته می‌شود.

۱-۵-۳. زبان‌های برنامه نویسی سطح بالا

برای تسریع امر برنامه نویسی، زبان‌های برنامه نویسی به گونه‌ای طراحی شدند که هر یک از دستورها و عبارت‌های آن، یک کار را به صورت کامل انجام دهند (به گونه‌ای که مانند زبان اسمنبلی، نیازی به نوشتن چند خط برنامه، برای یک دستور نباشیم). زبان‌های برنامه نویسی سطح بالا، به شما اجازه می‌دهند که دستورالعمل‌های برنامه‌ی خود را، مشابه با زبان انگلیسی روزمره و عبارات ریاضی معمولی بنویسید. ترجمه‌ی این دسته از زبان‌های برنامه نویسی به زبان ماشین، توسط کامپایلرها و یا مفسرها صورت می‌گیرد.

۱-۶. ترجمه‌ی زبان‌های برنامه نویسی سطح بالا به زبان ماشین

فرآیند ترجمه، به یکی از دو شکل زیر صورت می‌گیرد:

۱. ترجمه آنلاین (تفسیر)

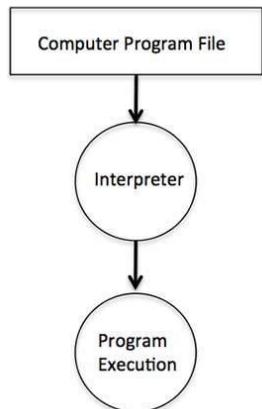
۲. ترجمه آفلاین (کامپایل)

۳. ترجمه‌ی ترکیبی

۱-۶. ترجمه‌ی آنلاین (تفسیر)

برنامه، خط به خط به مفسر داده می‌شود و مفسر، بعد از اجرای هر خط و در صورت نبود خطا، خط بعدی را دریافت و اجرا می‌کند. در این روش، تمام برنامه به یکباره ترجمه نمی‌شود و با برخورد به اولین خط، برنامه متوقف شده و خطوط بعدی اجرا نمی‌شوند.

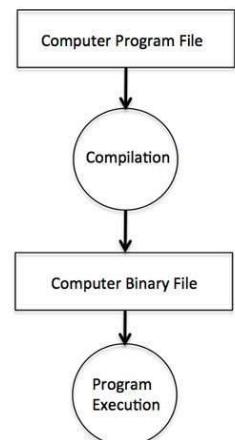
مثال ۶: مثال‌هایی از زبان‌های برنامه نویسی مفسری :



۲-۶. ترجمه‌ی آفلاین (کامپایل)

در این روش، تمام برنامه دریافت شده و تمام خطوط آن بررسی و تبدیل به زبان ماشین می‌شود و بعد از آنکه تمام برنامه ترجمه شد، برنامه اجرا می‌شود.

مثال ۷: مثال‌هایی از زبان‌های برنامه نویسی کامپایلری:



۳-۶. ترجمه‌ی ترکیبی

در این روش، برنامه‌ی نوشته شده به یک زبان میانی (نه زبان ماشین) کامپایل می‌شود و سپس، از زبان میانی، به زبان ماشین، تفسیر می‌شود.

مثال ۸: ترجمه‌ی ترکیبی را به صورت شماتیک نشان دهید:

مثال ۹: مثالی را از زبان‌های برنامه نویسی که به صورت ترکیبی نوشته می‌شوند، ارائه کنید.

مثال ۱۰: ترجمه‌ی ترکیبی، چه مزیتی دارد؟

۷-۱. سلسله مراتب ذخیره‌سازی داده‌ها

همان‌طور که در بخش‌های قبل بحث شد، داده‌ها به منظور پردازش، در ابتدا بایستی

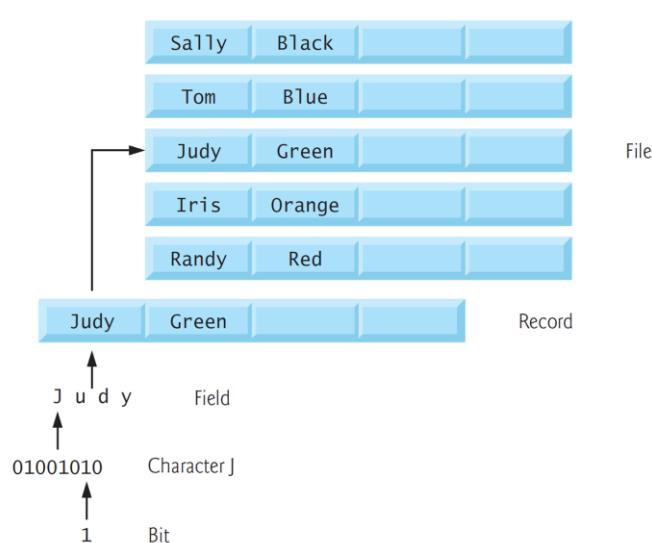
در حافظه‌ی اصلی ذخیره‌سازی شوند و همان‌طور که پیش‌تر گفته شد، داده‌ها به زبان

ماشین (که رشته‌هایی از ۰ و ۱ هستند) در حافظه ذخیره شده و فهمیده می‌شوند. بنابراین

کوچک‌ترین واحد حافظه، شامل یکی از دو مقدار ۰ یا ۱ است.

Control Characters				Graphic Symbols							
Name	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex
NUL	0	0000000	00	space	32	0100000	20	@	64	1000000	40
SOH	1	0000001	01	!	33	0100001	21	A	65	1000001	41
STX	2	0000010	02	"	34	0100010	22	B	66	1000010	42
ETX	3	0000011	03	#	35	0100011	23	C	67	1000011	43
EOT	4	0000100	04	\$	36	0100100	24	D	68	1000100	44
ENQ	5	0000101	05	%	37	0100101	25	E	69	1000101	45
ACK	6	0000110	06	&	38	0100110	26	F	70	1000110	46
BEL	7	0000111	07	'	39	0100111	27	G	71	1000111	47
BS	8	0001000	08	(40	0101000	28	H	72	1001000	48
HT	9	0001001	09)	41	0101001	29	I	73	1001001	49
LF	10	0001010	0A	*	42	0101010	2A	J	74	1001010	4A
VT	11	0001011	0B	+	43	0101011	2B	K	75	1001011	4B
FF	12	0001100	0C	.	44	0101100	2C	L	76	1001100	4C
CR	13	0001101	0D	-	45	0101101	2D	M	77	1001101	4D
SO	14	0001110	0E	,	46	0101110	2E	N	78	1001110	4E
SI	15	0001111	0F	/	47	0101111	2F	O	79	1001111	4F
DLE	16	0010000	10	0	48	0101000	30	P	80	1010000	50
DC1	17	0010001	11	1	49	0101001	31	Q	81	1010001	51
DC2	18	0010010	12	2	50	0101010	32	R	82	1010010	52
DC3	19	0010011	13	3	51	0101011	33	S	83	1010011	53
DC4	20	0010100	14	4	52	01010100	34	T	84	1010100	54
NAK	21	0010101	15	5	53	01010101	35	U	85	1010101	55
SYN	22	0010110	16	6	54	01010110	36	V	86	1010110	56
ETB	23	0010111	17	7	55	01010111	37	W	87	1010111	57
CAN	24	0011000	18	8	56	0110000	38	X	88	1011000	58
EM	25	0011001	19	9	57	0110001	39	Y	89	1011001	59
SUB	26	0011010	1A	:	58	0110010	3A	Z	90	1011010	5A
ESC	27	0011011	1B	:	59	0110011	3B	[91	1011011	5B
FS	28	0011100	1C	<	60	0111100	3C	\	92	1011100	5C
GS	29	0011101	1D	=	61	0111101	3D]	93	1011101	5D
RS	30	0011110	1E	>	62	0111110	3E	^	94	1011110	5E
US	31	0011111	1F	?	63	0111111	3F	-	95	1011111	5F

شکل ۲. جدول کارکترها (در استاندارد ASCII)



شکل ۳. سلسله مراتب داده‌ها (برگرفته از کتاب دایتل)

توضیحات	واحد سلسله مراتبی
	بیت (Bit)
	کاراکتر (character)
	فیلد (Field)
	رکورد (Record)
	فاایل (File)

۱-۸. حساب رایانه‌ای

همان‌طور که مشاهده کردیم، کوچک‌ترین واحد به منظور نگهداری اطلاعات، بیت‌ها هستند (که هر بیت یا حاوی عدد ۰ و یا حاوی عدد ۱ است)؛ همچنین مشاهده کردیم که با کنار هم قرار گیری بیت‌ها، واحدهای اطلاعاتی بزرگ‌تر درست می‌شود. بنابراین دانستن تفسیری از کنار هم قرار گیری بیت‌ها می‌تواند به ما در درک بهتر روش ذخیره، بازیابی و پردازش اطلاعات کمک کند. مجموعه‌ای از ارقام ۰ و ۱ که در کنار هم

قرار می‌گیرند، تشکیل یک عدد را در مبنای ۲ می‌دهند.

۱-۸-۱. ارزش مکانی

کمی در خاطرات خود، به دنبال روزهایی بگردیم که در کلاس اول دبستان درس می‌خواندیم؛ به خاطر می‌آوریم که معلم ما برای آموزش ارزش اعداد، از جدولی به نام **جدول ارزش مکانی استفاده** می‌کرد.

به عنوان مثال، برای تشریح عدد ۱۴۷ به این روش عمل می‌شد:

۱ دسته‌ی صدتاًی •

۴ دسته‌ی ۱ تایی •

۷ تا یکی‌ای. •

و جدول ارزش مکانی زیر را برای ما ترسیم می‌کرد :

صدگان	دهگان	یکان
۱	۴	۷

بعدها آموختیم که این عدد را می‌توان به صورت مضاربی از توان‌های ۱۰ و به صورت

زیر نوشت:

$$147 = 1 \times 10^2 + 4 \times 10^1 + 7 \times 10^0$$

بنابراین، تعریف جدول ارزش مکانی به صورت زیر نیز قابل انجام است.

صدگان	دهگان	یکان
ضریب ۱۰ ^۲	ضریب ۱۰ ^۱	ضریب ۱۰ ^۰
۱	۴	۷

مثال ۱۱: برای عدد ۱۰۲۰ جدول ارزش مکانی را ترسیم کرده و آن را به صورت مضاربی از

توان‌های ۱۰ بازنویسی کنید.

با توجه به این که پایه‌ی توصیف اعداد عدد ۱۰ است، به این اعداد اعداد مبنای ۱۰

گفته می‌شود. بنابراین اگر x یک عدد n رقمی مثل $a_{n-1}a_{n-2} \dots a_1a_0$ در مبنای ۱۰ (یا

پایه‌ی ۱۰) باشد، به صورت زیر بدست می‌آید:

$$x = \sum_{i=0}^{n-1} a_i \times 10^i$$

در نمایش اعداد مبنای ۱۰، از ارقام ۰ الی ۹ استفاده می‌شود. اما همان‌طور که دیده شد، در حساب رایانه‌ای، در هر بیت فقط از دو رقم ۰ یا ۱ استفاده می‌توان استفاده کرد؛ بنابراین پایه‌ی توصیف اعداد در حساب کامپیوتری، توان‌های عدد ۲ هستند و عددی مثل ۱۲ به صورت زیر نمایش داده می‌شود:

مکان سوم (هشتگان)	مکان دوم (چهارگان)	مکان اول (دوگان)	مکان صفرم (یکان)
ضریب 2^3	ضریب 2^2	ضریب 2^1	ضریب 2^0
۱	۱	.	.

بنابراین، عدد ۱۲ یک عدد در مبنای ۱۰ است و معادل آن در مبنای ۲ به صورت زیر نوشته می‌شود:

$$(1100)_2$$

این عدد به این صورت خوانده می‌شود (از چپ به راست) : یک، یک، صفر، صفر در مبنای دو.

۱-۸-۲. تبدیل مبناها

همان‌طور که در بخش قبل مشاهده کردید، عدد ۱۲ را می‌توان هم در مبنای ۱۰ و با ارقام ۰ الی ۹ نمایش داد و هم در مبنای ۲ با ارقام ۰ و ۱. هر دو، نمایش عدد ۱۲ هستند و دارای یک ارزش هستند. حال سوالی که در ادامه پیش می‌آید این است که: «چگونه می‌توان این دو نمایش را به یک دیگر تبدیل کرد؟». به منظور تبدیل یک عدد در مبنای ۲ به یک عدد در مبنای ۱۰، کافیست آن عدد را به صورت مجموعی از

مضارب توان‌های عدد ۲ (مطابق با جدول ارزش مکانی) بنویسیم.

مثال ۱۲: عدد ۱۱۰۱ در مبنای ۲ را به معادلش در مبنای ۱۰ تبدیل کنید.

یکان (2^0)	دوگان (2^1)	چهارگان (2^2)	هشتگان (2^3)
۱	۱	۰	۱

$$\begin{array}{ccccccc}
 & & & & \text{مرتبه عدد } ۱ & \text{مرتبه عدد } ۰ & \text{مرتبه عدد } ۱ \\
 & & & & \uparrow & \uparrow & \uparrow \\
 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 & = & 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 & = & 8 + 4 + 0 + 1 & = & 13 \\
 \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
 \text{ارزش مکانی عدد } ۱ & & \text{ارزش مکانی عدد } ۰ & & \text{ارزش مکانی عدد } ۱ & & \text{ارزش مکانی عدد } ۱
 \end{array}$$

برای سهولت کار، بالای هر رقم ارزش مکانی آن را بنویسید و ارزش مکانی رقم‌هایی که یک هستند را با هم جمع کنید.

$$(1\ 1\ 0\ 1)_2 = 1+4+1=13$$

با تغییر مبنای عدد، ماهیت آن عوض نمی‌شود؛ بلکه فقط شکل نمایش آن تغییر می‌کند، بنابراین می‌توان نوشت:

$$(1101)_2 = (13)_{10}$$

برای تبدیل یک عدد از مبنای ۱۰ به مبنای ۲، کافیست آن را به صورت **متوالی آنقدر بر ۲ تقسیم کنیم** تا جایی که دیگر نتوان آن را بر ۲ تقسیم کرد. سپس خارج قسمت و باقی‌مانده‌ی آخرین تقسیم و خارج قسمت سایر تقسیم‌ها را به ترتیب از چپ به راست در کنار هم قرار می‌دهیم.

مثال ۱۳: عدد ۵۳ را از مبنای ۱۰ به مبنای ۲ تبدیل نمایید.(پاسخ نهایی: ۱۱۰۱۰۱ در مبنای ۲)

۱-۸-۳. نمایش اعداد در مبناهای دیگر

در بخش قبل، با نمایش اعداد در مبنای ۲ آشنا شدیم. هر کدام از اعداد مبنای ۲، با یکی از ارقام ۰ یا ۱ نمایش داده می‌شوند. علاوه بر مبناهای ۲ و ۰، امکان نمایش اعداد در مبناهای دیگر نیز وجود دارد. در سایر مبناهای نیز به دنبال آن هستیم که بتوانیم اعداد را بر اساس پیمانه‌هایی از توان‌های آن پایه توصیف نماییم؛ در حالت کلی می‌توان از مبناهای ۲ الی ۱۶ برای توصیف اعداد استفاده کرد. در این خصوص به نکات زیر باید توجه کرد:

- در توصیف یک عدد در مبنای b (به طوری که $2 \leq b \leq 16$) می‌توان از

ارقام ۰ الی $b - 1$ استفاده کرد (حق استفاده از خود رقم b را نداریم).

- برای تبدیل یک عدد از مبنای b به مبنای ۱۰، کافیست همانند قبل ارقام آن را در

قالب مجموع مضارب توان‌های عدد b در نظر بگیریم؛ بنابراین اگر x یک عدد n

رقمی مثل $a_n a_{n-1} \dots a_1 a_0$ (یا پایه‌ی b) باشد، به صورت زیر

بدست می‌آید:

$$x = \sum_{i=0}^{n-1} a_i \times b^i$$

- همچنین برای تبدیل یک عدد از مبنای ۱۰ به مبنای b ، کافیست مطابق با

روش تقسیمات متوالی بر b ، نمایش آن را در مبنای b بدست آوریم.

۱-۸-۴. آشنایی با اعداد مبنای ۱۶ و روش‌های تبدیل آن

همان‌طور که در بخش قبل دیدیم، برای ذخیره‌ی هر کدام از ارقام در اعداد مبنای ۲ به تعداد بیت حافظه احتیاج داریم. همچنین آموختیم که برای نمایش هر عدد در مبنای b می‌توانیم از ارقام صفر الی استفاده نماییم.



سیستم عددی مبنای ۱۶ (Hexadecimal) یکی دیگر از سیستم های عددی کاربردی در رایانه است؛ برخی از نمونه های کاربردی آنها در کامپیوتر عبارت است از:

۱. نمایش آدرس دهی حافظه
۲. آدرس فیزیکی کارت های شبکه
۳. نمایش کد رنگ ها نمونه هایی از کاربرد این سیستم در رایانه

ارقام در اعداد مبنای ۱۶ به شرح جدول زیر هستند:

۱۵	۱۴	۱۳	۱۲	۱۱	۱۰	۹	۸	۷	۶	۵	۴	۳	۲	۱	۰	رقم
F	E	D	C	B	A	۹	۸	۷	۶	۵	۴	۳	۲	۱	۰	نمایش

سیستم شمارش مبنای ۱۶ نمایشی ساده تر برای اعداد در مبنای ۲ است. برای نمونه

عدد دودویی ۱۱۰۱۱۱۱۱۰۱۱۱۰۱۱ در مبنای ۱۶ به صورت FB6 نمایش داده می شود:

$$(11011111011)_2 = (6FB)_{16}$$

مثال ۱۴: هر کدام از ارقام مبنای ۱۶، حداکثر چند بیت را اشغال می کنند؟

تبديل یک عدد در مبنای ۱۶ به مبنای ۲، کافیست ارقام آن را به صورت چهاربیت - چهار بیت از سمت راست به چپ جداسازی کنید و معادل هر سه بیت را در مبنای ۱۶ بنویسید. در صورتی که نتوانستید آخرین بخش عدد را به ۴ بیت تقسیم کنید (کمتر از ۴ بیت بود)، به تعداد مورد نیاز صفر در انتهای عدد قرار دهید و سپس معادل آن را در مبنای ۱۶ بدست آورید.

مثال ۱۵: ارزش عدد 304_{10} در مبنای 16 را در مبنای 10 بدست آورید.

حل :

یکان (16^0)	شانزدهگان (16^1)	۲۵۶گان (16^2)
۴	۰	۳

$$\begin{array}{c}
 3 \times 16^2 + 0 \times 16^1 + 4 \times 16^0 = 3 \times 256 + 0 \times 16 + 4 \times 1 = 768 + 0 + 4 = 772 \\
 \text{مرتبه عدد } ۰ \quad \text{مرتبه عدد } ۱ \quad \text{مرتبه عدد } ۲ \\
 \text{ارزش مکانی عدد } ۰ \quad \text{ارزش مکانی عدد } ۱ \quad \text{ارزش مکانی عدد } ۲
 \end{array}$$

مثال ۱۶: تبدیل مبنای زیر را انجام دهید:

عدد 1001_2 را به مبنای 16 تبدیل کنید.

لب
?

$$1001_2 = 1+0=1 \rightarrow 1$$

$$1000_2 = 0 \rightarrow 0$$

$$1111_2 = 1+1+1+1=4 \rightarrow F$$

$$0000_2 = 0 \rightarrow 0$$

$$0001_2 = 1 \rightarrow 1$$

نکته : برای تبدیل اعداد مبنای 16 به مبنای 2 ، کافیست معادل هر رقم را در مبنای

2 (به صورت ۴ بیتی) بدست آورده و جایگزین آن رقم بکنید.

مثال ۱۷: تبدیل مبنای زیر را انجام دهید:

معادل عدد $AC1_{16}$ را در مبنای 2 بدست آورید.

A C 1

???????????????

$$A=10=(10)_2$$

$$C=12=(1100)_2$$

$$1=1=(0001)_2$$

A C 1

101011000001

۱-۹. مراحل نوشتگری و اجرای یک برنامه

۱-۹-۱. محیط ایجاد یکپارچه (IDE)

برای ایجاد یک برنامه، نیازمند محیطی هستیم که در آن، برنامه را بنویسیم و آن را مدیریت کنیم. به نرم افزاری که چنین محیطی را برای ما فراهم می‌کند، محیط توسعه جامع (IDE) گفته می‌شود. لازم به توضیح است که IDE با کامپایلر، متفاوت است. چرا که کامپایلر، یک نرم‌افزار کامپیوتری، به منظور ترجمه‌ی برنامه به زبان ماشین (زبان باینری یا زبان ۰ و ۱) است و IDE، نرم افزاری است که امکاناتی را برای توسعه‌ی نرم‌افزار، در اختیار ما می‌گذارد و حاوی امکانات زیر است:

۱. ویرایشگر کد

۲. ابزارهایی برای مدیریت فرآیند کامپایل

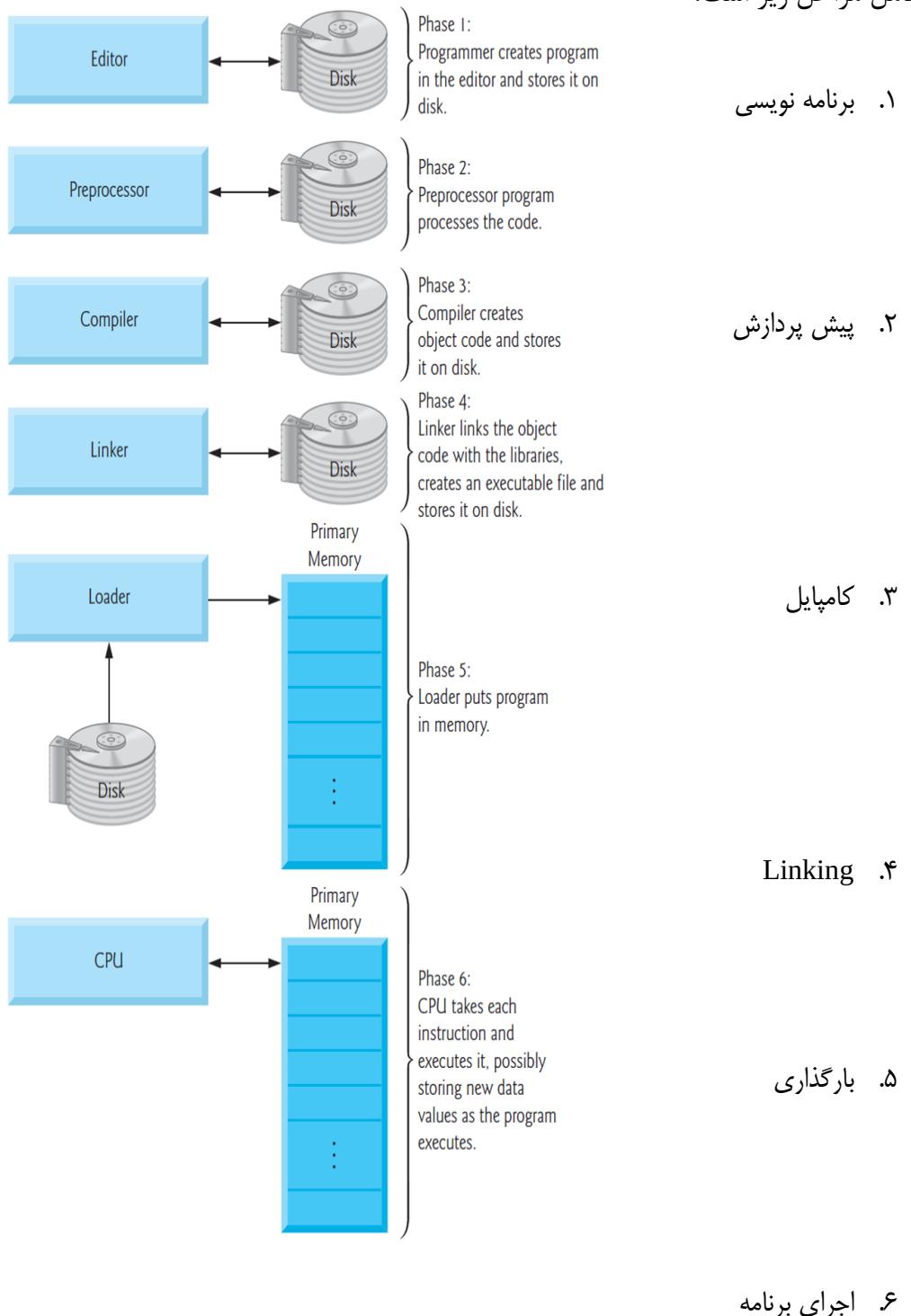
۳. ابزارهایی برای بررسی برنامه‌ها و کشف خطا.

با این حال، برخی از شرکت‌های نرم افزاری، IDE خود را به همراه یک کامپایلر به مصرف کنندگان ارائه می‌دهند. به گونه‌ای که با نصب IDE، کامپایلر زبان مربوطه نیز با آن نصب شود.

مثال ۱۸: با جستجو در اینترنت، نام چند IDE معروف را برای زبان C، بنویسید.

۱-۹-۲. مراحل ترجمه و اجرای یک برنامه‌ی نوشته شده به زبان C

یک برنامه‌ی نوشته شده به زبان C، مراحلی را می‌گذراند که در شکل ۱۷۷ کتاب دایتل، به آن اشاره شده و تشریح شده. در این شکل، مراحل اجرای یک برنامه به زبان C



۱۰-۱. ساخت یک پروژه ساده به زبان C

برای ساخت یک پروژه به زبان C ، از یکی از IDE های موجود استفاده کنید. مورد استفاده در کلاس، در صورت تمایل، میتوانید از IDE های مورد استفاده در Code Blocks، دیگر نیز استفاده کنید.

۱۰-۱-۱. مراحل ایجاد یک پروژه به زبان C

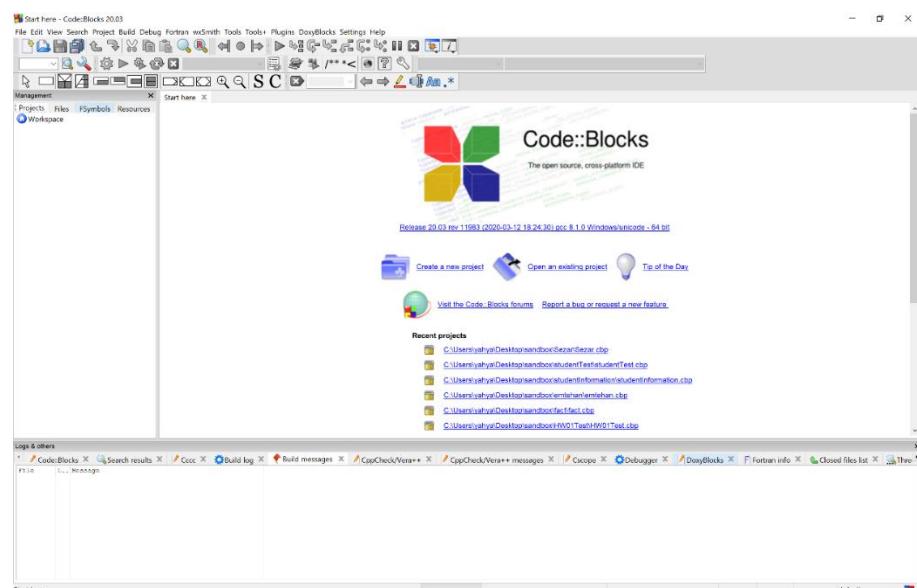
گام ۱: نصب نرمافزار Code Blocks : برای دانلود این نرمافزار میتوانید به سایت

نرمافزار مراجعه کرده و بر اساس سیستم عامل مورد نظرتان،

گام ۲: نرمافزار را اجرا کرده تا با محیط شکل ۱ مواجه شوید.

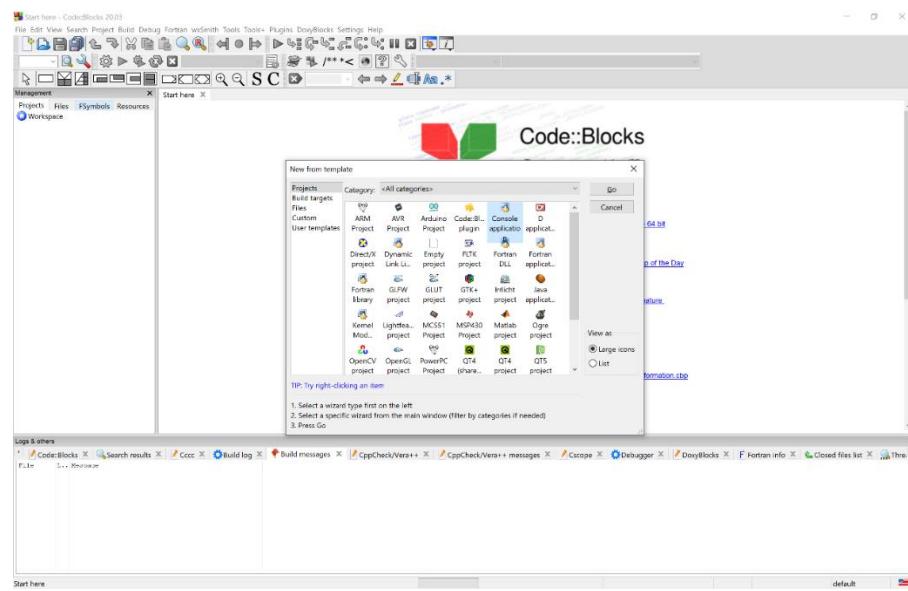
گام ۳: طبق شکل ۱ روی Create a new Project کلیک کنید تا فرآیند ساخت

یک پروژه C آغاز شود.



شکل ۱: صفحه ابتدایی کد بلاکس

گام ۴: سپس، پنجره زیر (شکل ۲) را مشاهده میکنید:



شکل ۲: انتخاب نوع پروژه. در این درس لازم است Console Application را انتخاب

کنید.

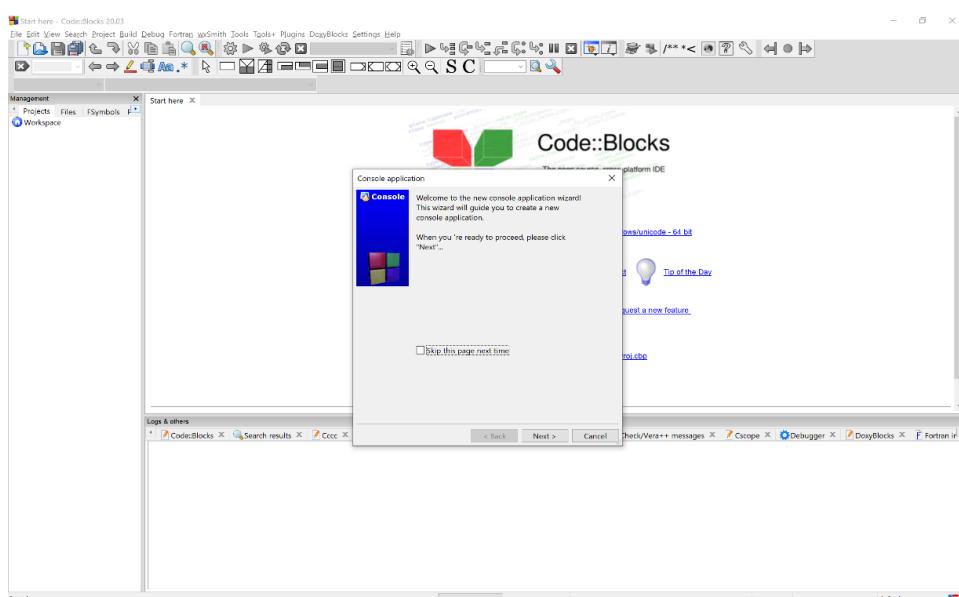
سپس گزینه Console Application را انتخاب کنید. سپس بر روی دکمه Go کلیک

کنید.

گام ۵ : این پنجره، صرفاً یک پیام خوش آمدگویی (شکل ۳) به شما می‌دهد.

می‌توانید تیک ... را بزنید تا از دفعات بعدی، این پیام به شما نشان داده نشود.

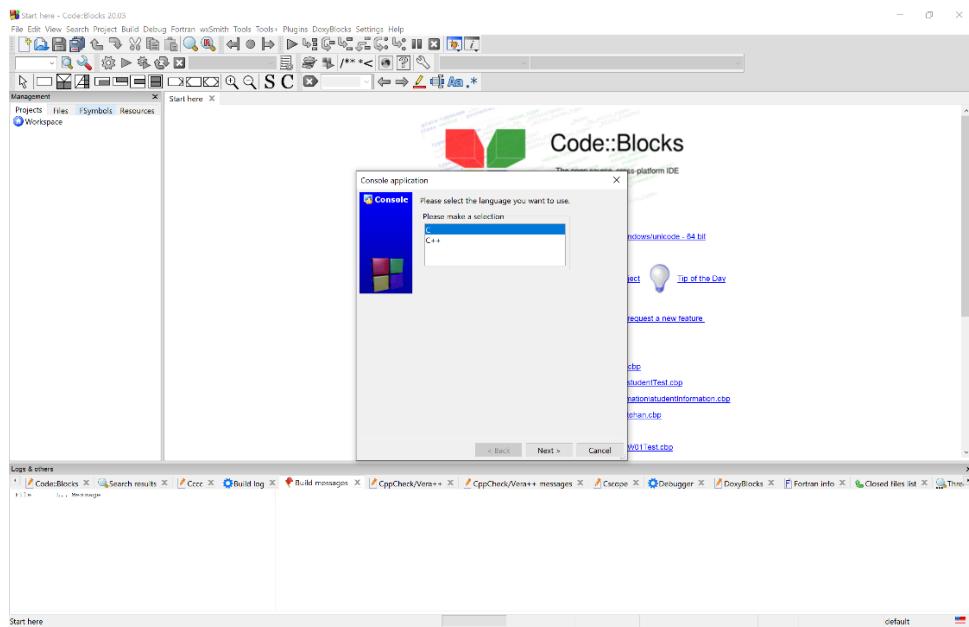
سپس بر روی Next کلیک کنید.



شکل ۳: پیام خوش آمدگویی در کدبلاکس

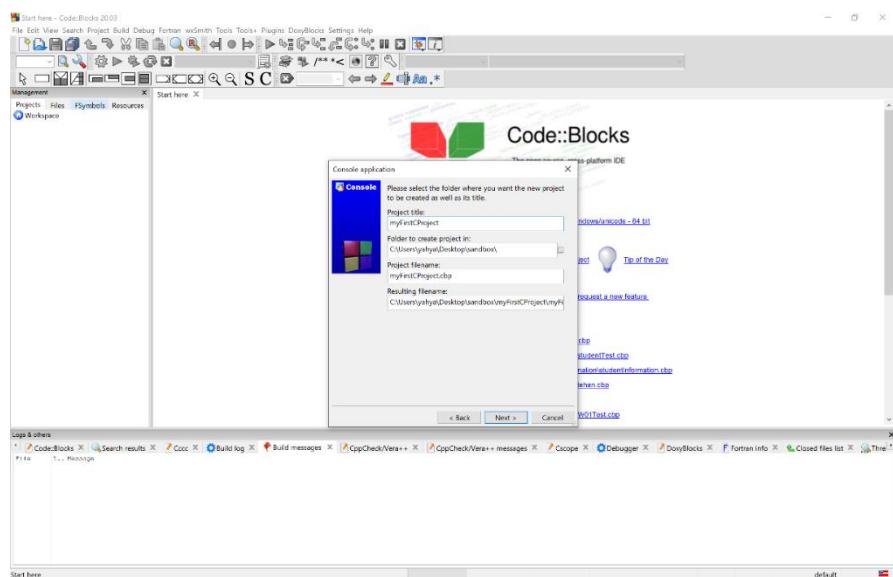
گام ۶ : در این مرحله (شکل ۴)، شما می بایست زبان مورد نظر خود را انتخاب

کنید. زبان C را انتخاب کرده و Next را کلیک کنید.



شکل ۴ : انتخاب زبان برنامه‌نویسی

گام ۷ : بعد از باز شدن پنجره‌ی زیر (شکل ۵)، مشخصات پروژه را وارد کنید.



شکل ۵ : وارد کردن مشخصات پروژه

- در فیلد Project Title ، یک عنوان به زبان انگلیسی برای پروژه‌ی خود انتخاب کنید و ترجیحاً، سعی کنید که بین کلمات آن، فاصله نباشد.
- در فیلد Folder to create Project مجلی را برای ذخیره سازی پروژه در نظر بگیرید.

- فایلد project File Name به صورت خودکار، با پر کردن فایلد اول پر می‌شود و لزومی

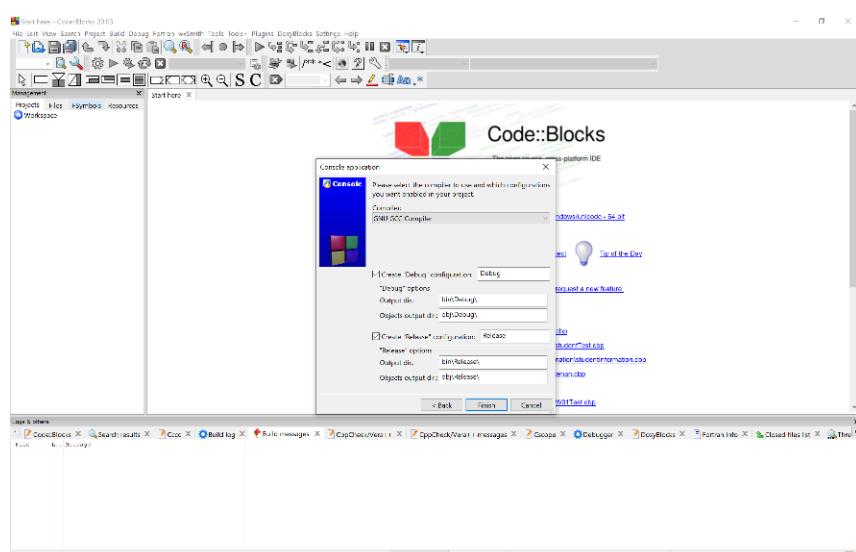
به پر کردن مجدد آن نیست.

- در فایلد آخر، محل نهایی پروژه شما، نشان داده می‌شود و به صورت خودکار، بعد از

تعیین مسیر پروژه، به طور خودکار پر می‌شود. سپس بر روی دکمه Next کلیک کنید.

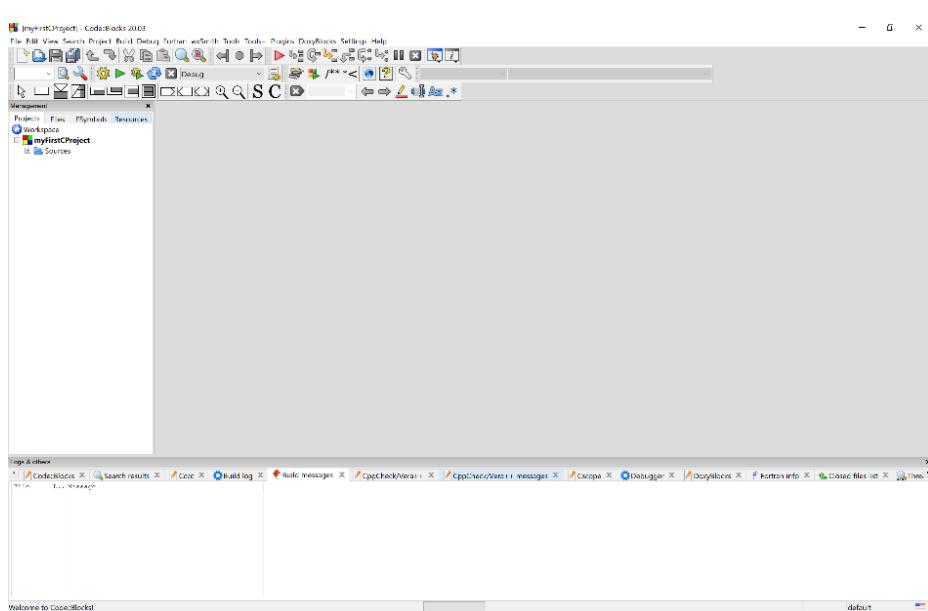
توجه کنید که در مسیر ذخیره‌سازی فایل‌های پروژه، نبایستی پوشه‌ای با نام **فارسی** داشته باشید.

گام ۸ : در نهایت، در این صفحه می‌بایست کامپایلر خود را انتخاب کنید. اگر کامپایلر به درستی نصب شده باشد، به طور پیش‌فرض GNU GCC Compiler انتخاب شده است. اگر No Compiler انتخاب شده بود، بدان معنا است که کامپایلر شما، به درستی بر روی سیستم نصب نشده است.



شکل ۶ : با کلیک بر روی **Finish**، پروژه‌ی شما ساخته می‌شود.

گام نهایی : در نوار سمت چپ برنامه، بر روی Sources دوبار کلیک کنید (و یا بر روی علامت + در کنار آیکون آن). سپس بر روی فایل main.c دو بار کلیک کرده تا در نهایت، وارد برنامه شوید.



شکل ۷ : صفحه‌ی ابتدایی پروژه‌ی ساخته شده

۱-۲-۲. اجزای یک برنامه‌ی نوشته شده به زبان C

برنامه‌ی زیر، ابتدایی ترین برنامه در زبان C است. (اگر فونت کوچک بود، ctrl را نگه دارید و غلطک اسکرول ماوس را بچرخانید) برنامه‌ی بالا، نقطه‌ی شروع برنامه نویسی به زبان C است. در ادامه، اجزای این برنامه را بررسی و تشریح می‌کنیم.

```

1 #include <stdio.h>
2 #include <stdlib.h> ❶
3
4 int main() { ❷
5 {
6     printf("Hello world!\n");
7 }
8 } ❸
  
```

شرح اجزای این برنامه، به صورت زیر است:

توضیح	کارکرد	شماره خط	شماره قرمز رنگ
		۱ و ۲	۱
		۴	۲

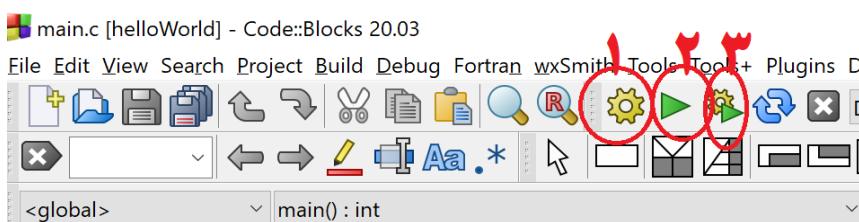
		۴	۳
		۶	۴
		۷	۵

نکته: کامنت‌گذاری یکی از راه‌هایی است که می‌توان به وسیله‌ی آن، به وضوح بهتر برنامه کمک کرد. خطهای کامنت شده توسط کامپایلر نادیده گرفته می‌شوند. کامنت به صورت یک خطی و چندخطی قابل درج است:

- ۱- یک خطی : پس از دو علامت اسلش درج می‌شود : //
- ۲- چندخطی : در بین این دو علامت درج می‌شود : /*Comment*/

۱۰-۳. کامپایل و اجرای برنامه در ابزار Code Blocks

از تولبار بالای صفحه، آیکون‌های زیر بدین صورت عمل می‌کنند:



شکل ۸ : ابزارهای کامپایل و اجرای برنامه‌ها در Code Blocks

۱. چرخ دنده: با زدن این آیکون، برنامه‌ی شما کامپایل می‌شود و به زبان ماشین ترجمه می‌شود.



۲. مثلث: با زدن این آیکون، برنامه‌ی کامپایل شده، اجرا می‌شود. (اگر تغییری داده باشد و مورد ۱ را انجام نداده باشد، تغییرات داده شده در برنامه لحاظ نمی‌شود)

۳. مثلث و چرخ دنده: با کلیک بر روی این آیکون، برنامه کامپایل شده و سپس اجرا می‌شود.

درس ۲ : آغاز برنامه نویسی

در پایان این فصل از فرآگیران انتظار می‌رود با مفاهیم زیر آشنا شده باشند:

- با ساختار و اجزای یک برنامه‌ی ساده به زبان C به صورت مقدماتی آشنا شوند.
- با دستورات ورودی و خروجی (printf و scanf) و فرمت بندی آن آشنا شوند.
- با مفاهیم مربوط به متغیرها آشنا شود.
- اصول کدنویسی تمیز (نظیر نام‌گذاری درست متغیرها را بیاموزد).

۲-۱. عملیات ورودی و خروجی در زبان C

در زبان C، به منظور انجام عملیات ورودی و خروجی، از توابعی استفاده می‌شود که از قبیل نوشته شده‌اند و مشخصات آن‌ها، در فایلی به نام stdio.h قرار دارد؛ بنابراین، اگر بخواهیم از این توابع در برنامه‌ی خود استفاده کنیم، لازم است با استفاده از فرمان #include این فایل را به برنامه ضمیمه کنیم تا در مرحله‌ی پیش پردازش، دستورات به برنامه اضافه شوند. در این جلسه به طور خاص، با تابع printf به منظور ارائه خروجی و تابع scanf به منظور دریافت ورودی، آشنا می‌شویم.

۲-۱-۱. چاپ خروجی

به منظور مشاهده مقادیر و نتایج حاصل از اجرای برنامه بر روی صفحه نمایش (خروجی استاندارد). از تابع printf استفاده می‌شود. فرم کلی این تابع به صورت زیر است:

printf ([Control String] , arg1, arg2, ...);

آنچه باید چاپ شود در String Control واقع می‌شود. مقادیری که می‌بایست در خروجی و طبق ساختار String Control باید چاپ شوند، در arg0 ، arg1 و ... می‌آیند.

این مقادیر، به جای کنترلی می‌نشینند و سایر کاراکترهای String Control ، عیناً در String Control می‌شوند. کاراکترهای کنترلی، بعد از یک علامت٪ در خروجی چاپ می‌شوند. برخی از کاراکترهای کنترلی پر کاربرد، به صورت زیر هستند.

جدول ۱. کاراکترهای کنترلی

کارکرد	کاراکتر کنترلی (بعد از٪ می‌آید)
چاپ مقادیر عدد صحیح	i , d
چاپ مقادیر اعداد اعشاری	f
چاپ کاراکتر	c
چاپ رشته (دنباله ای از کاراکترها)	s

هم چنین، کاراکترهایی وجود دارند که به علل مختلفی، نمی‌توان آن‌ها را مستقیماً در بین دو کوتیشن چاپ کرد. این کاراکترها، با \ (بخوانید بک اسلش) شروع می‌شود و به آن‌ها، Escape Sequence گفته می‌شود. در ادامه، مهم‌ترین این کاراکترها را معرفی می‌کنیم.

جدول ۲. کاراکترهای خاص (Escape Sequence)

عملکرد	Escape Sequence
خط جدید ایجاد می‌شود و cursor را به خط بعدی می‌برد.	\n
یک فاصله افقی به اندازه یک tab ایجاد می‌کند.	\t
باعث پخش صدای هشدار، از بلندگوی سیستم می‌شود. می‌توان برای اعلام خطأ به کاربر از این کاراکتر استفاده کنید.	\a
برای چاپ کاراکتر \ استفاده می‌شود.	\\"
برای چاپ علامت " استفاده می‌شود.	\"

۲-۱-۲. دریافت ورودی

در برنامه نویسی، عموماً لازم می‌شود که مقادیر عددی و یا کاراکتری، در حافظه نگهداری شوند تا در محاسبات به کار گرفته شوند و یا در صورت لزوم، تغییر کنند. محل ذخیره سازی هر متغیر، توسط سیستم عامل تعیین می‌شود. برای استفاده از این قابلیت،

برای هر متغیر مورد نظر باید کارهای زیر انجام شود:

۱. تعریف متغیر (Deceleration)

۲. مقداردهی به متغیر (Assignment)

تعریف متغیر، با ذکر نوع آن و نام آن میسر است. برای مقداردهی اولیه، از علامت `=` استفاده می‌شود. در فصل بعدی به صورت دقیق‌تر با متغیرها و انواع آن‌ها آشنا می‌شویم.

مثال ۱۹: بنظر شما مقادیر متغیرها در کدام یک از حافظه‌ها ذخیره می‌شوند؟

مثال ۲۰: متغیری به نام `number` تعریف کنید و به آن، مقدار اولیه ۱۰ را بدهید.

در زبان C ، جهت خواندن انواع داده‌ها با فرمت‌های مشخص، از طریق صفحه‌ی کلید (ورودی استاندارد) و قرار دادن آن‌ها در حافظه، از تابع `scanf` استفاده می‌شود. فرم این تابع به شکل زیر است:

`Scanf (control String, arg0, ...);`

اولین آرگومان، یک رشته است که چگونگی خواندن مقادیر متغیرهایی را که آدرس آن‌ها در `arg0` و... قرار دارد، مشخص می‌کند. در محل آرگومان‌های بعدی، آدرس متغیرهایی که خواندن مقدار آن‌ها مورد نظر است، قرار می‌گیرد. جهت استناد به آدرس، لازم است که قبل از نام متغیر، علامت `&` را قرار دهید. در `Control String` ، می‌بایست از کاراکترهای کنترلی استفاده کنید. مهم‌ترین این کاراکترها به شرح زیر هستند:

جدول ۳. کاراکترهای کنترلی به منظور استفاده در دریافت ورودی

کارکرد	کاراکتر کنترلی
خواندن مقادیر عدد صحیح	<code>%d , %i</code>
خواندن مقادیر اعداد اعشاری	<code>%f</code>
خواندن کاراکتر	<code>%c</code>
خواندن رشته (دنباله‌ای از کاراکترها)	<code>%s</code>

مثال ۲۱: برنامه‌ای را به زبان C بنویسید که دو عدد را از ورودی دریافت کرده و در خروجی

۲-۲. متغیرها

در دروس قبل آموختیم که یکی از کاربردهای کامپیوتر انجام راحت‌تر محاسبات است؛ بدون ذخیره‌سازی داده‌ها، امكان اجرای محاسبات بر روی آن‌ها میسر نیست. بنابراین لازم است پیش از انجام محاسبات بر روی داده‌ها آن‌ها را در مکانی از حافظه ذخیره کنیم. همچنین، در اثر محاسبات، ممکن است مقادیر این داده‌ها تغییر کند. در نتیجه لازم است بتوانیم مقادیر ذخیره شده در حافظه را بازیابی کنیم.

متغیر (Variable) : یک متغیر (Variable) مکانی از حافظه است که داده‌ها در آن

ذخیره شده و می‌تواند مقدار آن تغییر کند.

۲-۱. تعریف کردن متغیر

پیش از استفاده از متغیر بایستی آن را تعریف کنیم. در تعریف متغیر باید دو چیز را در مورد آن مشخص کنیم :

تعریف متغیر
Declare Variable

..... ۱.

..... ۲.

پس از مشخص کردن نوع آن و انتخاب یک نام برای آن، پس از اجرای برنامه در حافظه‌ی محلی برای ذخیره‌سازی آن تعیین می‌شود؛ آن مکان از حافظه دارای یک آدرس است که می‌تواند از عدد ۰ آغاز شده و بسته به میزان حجم حافظه، افزایش یابد. نحوه تعریف متغیر به صورت زیر است :

[Variable Type] [Variable Name];

به نام متغیر، شناسه و یا identifier گفته می‌شود.

نوع داده‌ی متغیر مشخص می‌کند که مقدار آن متغیر از چه نوعی است. مهم‌ترین انواع داده در جدول زیر مشخص شده‌اند (انواع داده‌ی دیگری نیز داریم که بررسی آن‌ها را به شما می‌سپاریم):

جدول ۴. انواع داده های پر کاربرد

کلمه کلیدی	نوع داده	مثال و یا توضیح
int	عدد صحیح	۷۷، ۰، ۱۰ و ...
float	عدد اعشاری	۱۲۰,۳۲۳۴۵۶۳ (با دقت ۷ رقم اعشار)
double	عدد اعشاری با دقت مضاعف	۱۴,۲۳۸۵۴۳۷۸۹۰۵۴۲۳۸۹ (با دقت ۱۶ رقم اعشاری)
char	کاراکتر	'a' یا '?' یا '€' و ...
long یا long int	عدد صحیح بزرگ	۲۱۴۷۴۸۳۶۴۸ - تا ۲۱۴۷۴۸۳۶۴۷
short یا Short int	عدد صحیح کوچک	۳۲۷۶۷ - تا ۳۲۷۶۸

مثال ۲۲: برنامه ای به زبان C بنویسید و در آن، دو متغیر از نوع عدد صحیح با نام های

number1 و number2 تعریف کنید.

۲-۲-۲. مقداردهی به متغیرها

هنگامی که متغیری را تعریف می کنید اتفاقات زیر رخ می دهد:

۱. نوع آن توسط شما مشخص می شود.
۲. به آن نامی را اختصاص می دهید.
۳. پس از اجرای برنامه، مکانی از حافظه توسط سیستم عامل برای آن تعیین می شود.

اما در آن مکان از حافظه، مقداری به متغیر تخصیص داده نشده است. بنابراین بایستی مقادیری را برای آن تعیین کنید.

تخصیص مقدار به متغیر (مقداردهی متغیر) به معنای ذخیره سازی یک مقدار در حافظه است. مقداردهی متغیر با استفاده از علامت تساوی = صورت می گیرد.

مثال ۲۳: به متغیرهای مثال قبل، مقادیری را تخصیص داده و سپس، آنها را چاپ کنید.
سپس متغیری را به نام sum تعریف کرده و حاصل جمع متغیرها را در آن ذخیره کرده و سپس آنها را چاپ نمایید.

نکته: تعریف متغیر و تخصیص مقداری به آن می‌تواند در یک خط انجام شود.

نکته: همان‌طور که گفته شد، پس از اجرای برنامه، مکان‌هایی در حافظه برای ذخیره‌سازی آن‌ها مشخص خواهد شد که آن مکان‌ها دارای یک آدرس مشخص هستند. شاید از خود بپرسید که چگونه می‌توان آدرس هر کدام از متغیرها را چاپ کرد؟ و یا به آن دسترسی داشت؟

به منظور دسترسی به آدرس متغیر کافیست پیش از آن، عملگر `&` را قرار دهید. آدرس هر متغیر یک عدد صحیح است.

مثال ۲۴: آدرس هر کدام از متغیرهای مثال قبل را چاپ کنید.

مثال ۲۵: خواندن مقدار متغیر عملی است (مخرب/غیر مخرب) و مقداردهی آن (مخرب/غیر مخرب) است.

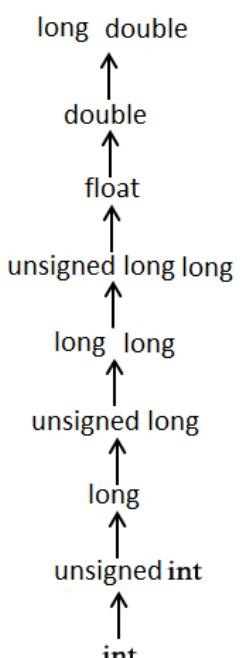
مثال ۲۶: برنامه‌ای را به زبان C بنویسید که دو عدد را از کاربر دریافت کرده و آن‌ها را باهم جمع نماید.

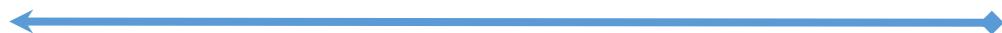
۳-۲-۲. تبدیل نوع (Type Cast)

داده‌ها با انواع مختلف می‌توانند به یک دیگر تبدیل شوند. البته توجه داشته باشید که تبدیل نوع در جهت فلش‌هاش شکل مقابله‌دار صورت گیرد. در صورتی که تبدیل نوع به صورت خودکار انجام نشود، کافیست نوع داده‌ی مقصد را در داخل پرانتز بنویسید و قبل از متغیر قرار دهید:

```
int sum = (int) sumOfNumbers;
```

توجه کنید که اگر تبدیل نوع را در خلاف جهت فلش‌ها انجام دهید، ممکن است بخشی از داده‌های خود را از دست بدهید (و یا حتی در موقعی دچار خطا زمان اجرا شوید).





مثال ۲۷: برنامه‌ای را به زبان C بنویسید که دو عدد را دریافت کرده و میانگین آن‌ها را محاسبه کند.

۴-۲-۲. نکاتی در خصوص متغیرها

۱. نام متغیرهای خود را با معنا انتخاب کنید.
۲. نام متغیرها نمی‌تواند جزو کلمات کلیدی زبان C باشد.
۳. برای نام گذاری متغیر از روش نام‌گذاری شتری (Camel Case) استفاده کنید.
۴. نام متغیر خود را خیلی طولانی انتخاب نکنید.
۵. پس از تعریف متغیر، در اولین جایی که می‌توانید به آن مقدار اولیه اختصاص دهید.
۶. توجه کنید که پس از تغییر مقدار متغیرها، مقدار قبلی آن از بین می‌رود.
۷. توجه کنید که پس از پایان یافتن اجرای برنامه تمامی مقادیر متغیرها از بین می‌روند.

درس ۳ : الگوریتم‌ها و محاسبات منطقی

در پایان این فصل از فرآگیران انتظار می‌رود با مفاهیم زیر آشنا شده باشند:

- با الگوریتم به عنوان راه حل مسائل آشنا شود.
- بتواند الگوریتم‌های ساده را بنویسد.
- با فلوچارت به عنوان یک ابزار به منظور نمایش بصری الگوریتم آشنا شود.
- تفاوت الگوریتم و فلوچارت را درک کند.
- با ساختار شرطی آشنا شود.
- بتواند ساختار شرطی را در زبان C با استفاده از دستور if پیاده‌سازی کند.



۳-۱. مقدمه حل مسئله با استفاده از الگوریتم‌ها

همان‌طور که در جلسات قبل دیدیم، برنامه نویسی ابزاری است که نیازمندی‌های یک انسان (کاربر) را به یک برنامه‌ی کامپیوتری، تبدیل می‌کند. برای حل یک مسئله با کامپیوتر، لازم است که در ابتدا، با انجام کارهای زیر، مسئله را تحلیل کنیم:

۱. مشخص نمودن ورودی‌ها (داده‌های مسئله)

۲. مشخص نمودن خروجی‌ها

۳. پردازش ورودی‌ها و تولید خروجی‌ها

کلمه الگوریتم برگرفته از
نام دانشمند ایرانی، محمد
بن موسی خوارزمی است و
تبدیل یافته‌ی کلمه
الخوارزمی است.

به طور غیر رسمی، به هر رویه محاسباتی خوش تعریف که یک یا چند مقدار را به عنوان ورودی می‌گیرد و یک یا چند مقدار را به عنوان خروجی باز می‌گرداند، الگوریتم (Algorithm) گفته می‌شود.
بنابراین، الگوریتم دنباله‌ای از محاسبات است که ما را به می‌رساند.

۲-۳. ویژگی‌های الگوریتم‌ها

۱. دارای نقطه شروع و پایان هستند.
۲. تعداد دستورهای آن‌ها متناهی و مشخص است.
۳. اجرای دستورهای آن، لزوماً به ترتیب نیست و قابل اجرا در هر شرایطی، دارای یک معنا و مفهوم مشخص باشد.
۴. دستورهای الگوریتم، باید بی‌ابهام باشد و در هر شرایطی، دارای یک معنا و مفهوم مشخص باشد.

۳-۳. انواع دستورات در الگوریتم‌ها

دستورهای الگوریتم‌ها را می‌توان به صورت زیر، طبقه‌بندی نمود:

۱. دستورات ورودی
۲. دستورات خروجی
۳. دستورات انتساب
۴. دستورات محاسباتی و منطقی
۵. دستورات تصمیم‌گیری

۴-۳. روش‌های بیان الگوریتم‌ها

همان‌طور که آموختیم، الگوریتم ابزاری مناسب برای درک مسئله است و بیان می‌کند که در هر مرحله چه کاری را با چه انجام دهیم.

۴-۳-۱. روش اول : شبه کد

شبه کد یک ابزاری است برای درک بهتر صورت مسئله و باید توجه کنید که یک ابزار و یا زبان برنامه نویسی نیست!! بنابراین شبه کدها قابل کامپایل شدن یا اجرا شدن

مثال ۲۸: شبه کد ریختن یک فنجان قهوه می‌تواند به شکل زیر باشد:

۱. شروع

۲. یک لیوان خالی بردار

۳. اگر لیوان خالی است، یک پیمانه نسکافه در آن ببریز.

۴. و گرنه آن را به مشتری تحویل بده

۵. پایان

1. Start

2. Put an Empty Mug

3. IF Mug is Empty, Fill it with a cup of coffee.

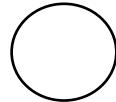
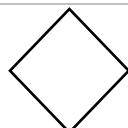
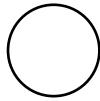
4. ELSE deliverer it to customer.

5. End

۳-۴-۲. روش دوم : فلوچارت (روندنما)

در بخش قبل، با الگوریتم‌ها آشنا شدیم و آموختیم که بیان راه حل یک مسئله، با ارائه‌ی یک الگوریتم برای آن مسئله محقق می‌شود. حال به دنبال روشی هستیم تا این راه حل را نشان دهیم تا بتوانیم اجرای آن را به خوبی درک کنیم.
یکی از راههای بیان این الگوریتم، نوشتتن آن به صورت مرحله به مرحله است. اما اگر روش بیان آن با استفاده از یک ابزار گرافیکی و تصویری باشد، بهتر می‌توانیم الگوریتم خود را درک کرده و تحلیل نماییم. فلوچارت، یکی از روش‌های نشان دادن الگوریتم به صورت گرافیکی است. در فلوچارت، از اشکال مختلفی استفاده می‌کنیم که در ادامه، با این اجزا آشنا می‌شویم.

جدول ۵. نمادهای مرسوم در ترسیم فلوچارت

مفهوم	تصویر نماد
	
	
	
	
	
	

توجه کنید که در فلوچارت‌ها، ممکن است از نمادهای دیگری نیز استفاده شود که

در اینجا به آن‌ها اشاره نکرده‌ایم.

مثال ۲۹: بنظر شما، الگوریتم و فلوچارت چه ارتباطی با یک دیگر دارند؟ آیا طراحی یک الگوریتم برای یک مسئله به معنای ترسیم فلوچارت برای آن است؟ در این خصوص توضیح دهید.

مثال ۳۰: الگوریتمی بنویسید که سه عدد صحیح را در ورودی دریافت کند و در خروجی اعلام کند که آیا سه عدد می‌توانند تشکیل یک مثلث قائم‌الزاویه را بدهند؟

حل : الگوریتم را به صورت مرحله به مرحله بیان کنید.

۱. شروع

۲

۳

۴

۵

۶

۷

۸

۹

۱۰. پایان

مثال ۳۱: الگوریتم طراحی شده در مثال ۳۰ را با استفاده از یک فلوچارت، ترسیم کرده و بیان کنید.

۳-۵. مروری بر محاسبات منطقی

علاوه بر محاسبات حسابی (مثل جمع، تفریق و...) نوع دیگری از محاسبات را داریم که به محاسبات منطقی معروف هستند. خروجی محاسبات حسابی، اعداد هستند اما خروجی محاسبات منطقی، جواب درست (True) یا نادرست (False) می‌باشد. البته عموماً معادل عددی شرط درست، عدد ۱ بوده و معادل عددی شرط غلط، عدد صفر است. نمونه‌هایی از عملگرهای منطقی را در ادامه مشاهده می‌کنید:

جدول ۶. عملگرهای منطقی در زبان C

کارکرد	عملگر	نوع عملگر منطقی
	<	عملگرهای مقایسه‌ای
	>	
	\leq	
	\geq	
	$=$	
	!	عملگرهای منطقی
	\parallel	
	$\&\&$	

درس ۱۴ : ساختارهای کنترلی

در پایان این فصل فرآگیران بایستی با مطالب زیر آشنا شده باشند:

- با انواع ساختارهای کنترلی اصلی (ترتیبی، تصمیم‌گیری و تکراری) آشنا شوند.
- با انشعاب در برنامه و روش‌های پیاده‌سازی آن آشنا شوند.
- با ساختار دستوری تکراری و انواع آن (با مشخص بودن/نبودن تعداد دفعات تکرار) آشنا باشند.

در حالت عادی، بنظر می‌رسد که تمامی دستورات باید به صورت دنباله‌ای اجرا شود؛

اما یقیناً این‌طور نیست! گاهی لازم می‌شود که به فرآخور شرایط برنامه، بخش خاصی از

آن را انجام دهیم. به گفتهٔ Böhm Jacopini همهٔ الگوریتم‌ها، سه

دستور اساسی داریم:

۱. دستورهای متوالی

۲. انتخاب، انشعاب و تصمیم‌گیری

۳. حلقه و تکرار

همان طور که در مثال‌های قبل دیدیم، گاهی لازم است که در الگوریتم‌های خود،

شرایطی را بررسی کرده و بر اساس آن شرایط، اقداماتی را انجام دهیم، در چنین شرایطی،

الگوریتم ما دارای ساختاری صرفاً متوالی نخواهد بود.

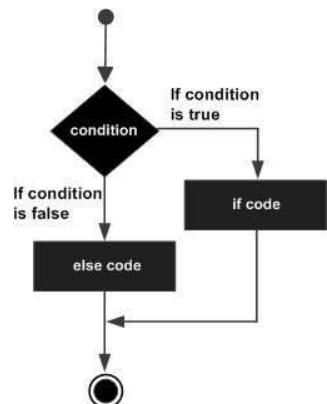
۱-۴. تصمیم‌گیری و دستورات شرطی

۱-۱-۴. ساختار تصمیم‌گیری if

در زبان C برای ایجاد انشعاب، از ساختار if استفاده می‌کنیم.

```
if(boolean_expression) {
    /* statement(s) will execute if the boolean expression is true */
}
```

اگر قصد داشته باشیم که در صورت برقرار نبودن شرط، مجموعه دستورات دیگری اجرا شوند، ساختار else را به بعد از ساختار if اضافه می‌کنیم.



```
if(boolean_expression) {
    /* statement(s) will execute if the boolean expression is true */
} else {
    /* statement(s) will execute if the boolean expression is false */
}
```

مثال ۳۲: برنامه‌ای را به زبان C بنویسید که عددی را دریافت کرده و تعیین کند که آیا این

عدد زوج است یا خیر؟

حل : در ابتدا، الگوریتمی برای آن بنویسید :

۱. شروع

..... ۲.

..... ۳.

..... ۴.

..... ۵.

۶. پایان

سپس یک فلوچارت برای آن ترسیم کنید.

در نهایت، بر اساس الگوریتم نوشته شده و فلوچارتی که برای آن ترسیم کرده‌اید،

یک برنامه به زبان C برای آن بنویسید.

نکته: اگر پس از عبارات if یا else ، علامت { قرار نگیرد، دستوری که در خط بعدی این عبارات هستند در ذیل آنها در نظر گرفته می‌شوند و دستورات بعدی، خارج از این ساختارها خواهند بود.

مثال ۳۳: برای شبه کد زیر، کدی را به زبان C بنویسید؛ این شبه کد نمره‌ی دانشجو را دریافت کرده و بر اساس نمره‌ی اکتسابی، یکی از سطوح A تا F را به دانشجو نسبت می‌دهد.

```
If student's grade is greater than or equal to 90
    Print "A"
else
    If student's grade is greater than or equal to 80
        Print "B"
    else
        If student's grade is greater than or equal to 70
            Print "C"
        else
            If student's grade is greater than or equal to 60
                Print "D"
            else
                Print "F"
```

در صورتی که بخواهیم چند شرط مختلف و مجزا را بررسی کنیم (یعنی در صورت برقرار بودن یکی از آنها، دیگر شرط‌ها چک نشوند) از ساختار else-if به شکل زیر استفاده می‌کنیم:

```
if(boolean_expression_1) {
    /* Executes when the boolean expression 1 is true */
} else if( boolean_expression_2) {
    /* Executes when the boolean expression 2 is true */
} else if( boolean_expression_3) {
    /* Executes when the boolean expression 3 is true */
} else {
    /* executes when the none of the above condition is true */
}
```

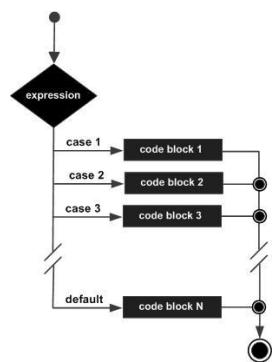
مثال ۳۴: برای الگوریتمی که در مسئله‌ی مطرح شده در مثال ۳۰: (بررسی قائم‌الزاویه بودن یک مثلث) طراحی کرده‌اید، یک برنامه به زبان C بنویسید.

۲-۱-۴. ساختار کنترلی switch – case

گاهی ممکن است بر اساس مقدار یک متغیر، بخواهیم تصمیم معینی را بگیریم. یکی دیگر از راه‌های تصمیم‌گیری استفاده از ساختاری به نام switch و case است. استفاده از

این ساختار مزایایی دارد از جمله:

۱. در مواقعي سرعت برنامه را بهبود می‌بخشد.
۲. از ایجاد if – else های متداخل و متعدد جلوگیری می‌شود.
۳. باعث خوانایی برنامه می‌شود.



روش کلی استفاده از این دستور به شکل زیر است:

```

switch(expression) {
    case constant-expression :
        statement(s);
        break; /* optional */

    case constant-expression :
        statement(s);
        break; /* optional */

    /* you can have any number of case statements */
    default : /* Optional */
        statement(s);
}
  
```

مثال ۳۵: برنامه‌ای را به زبان C بنویسید که نمره‌ی کیفی یک دانشجو را دریافت کرده و توضیحاتی را در خصوص آن به صورت زیر چاپ کند.

Description	Mark
Excellent	A
Very Good	B
Good	C
Acceptable	D
Bad	E
Fail	F

۴-۲. ساختارهای تکراری

در بسیاری از موارد، نیاز داریم که در الگوریتم خود، کاری را به صورت مکرر انجام دهیم و آن را تکرار کنیم. در برخی از موارد، تعداد دفعاتی که باید یک کار تکراری را انجام دهیم، مشخص است و در مواردی، تکرار را باید آنقدر انجام دهیم تا به یک نتیجه مطلوبی دست پیدا کنیم. در چنین شرایطی، گفته می‌شود که تکرار ما، نامعلوم است.

بنابراین، می‌توانیم ساختارهای تکرار را به صورت زیر طبقه‌بندی کنیم:

۱. ساختارهایی با تکرار مشخص: از ابتدا مشخص است که چندبار می‌خواهیم عمل تکراری را انجام دهیم.

۲. ساختارهایی با تکرار نامشخص: شرط تکرار، وابسته به شرایط مسئله است و آنقدر باید تکرار انجام شود تا به شرایط مطلوب ما در مسئله، ارضاء شود.

مثال ۳۶: مثال‌هایی را از تکرار با تعداد دفعات مشخص و نامشخص ارائه کنید.

۴-۲-۱. تکرار با تعداد دفعات نامشخص (حلقه‌ی while)

در زبان C برای پیاده‌سازی حلقه‌ها با تعداد دفعات تکرار نامشخص از ساختار حلقه‌ی while استفاده می‌شود. ساختار این حلقه‌ها به شکل زیر است:

```
while(condition) {
    statement(s);
}
```

مثال ۳۷: برنامه‌ای را به زبان C بنویسید که ۱۰ عدد را دریافت کرده، با یک دیگر جمع کرده و میانگین آن را چاپ نماید.

مثال ۳۸: (الف) الگوریتمی بنویسید که دو عدد را از ورودی دریافت کرده و عدد اول را برعدد

دوم با استفاده از تفریق تقسیم کرده و خارج قسمت را اعلام کند.

(ب) این الگوریتم را با استفاده از فلوچارت ترسیم کنید.

(ج) الگوریتم خود را با زبان C پیاده‌سازی کنید.

مثال ۳۹: برنامه‌ای را به زبان C بنویسید که عددی را از ورودی دریافت کرده و تعداد ارقام

آن را بنویسد.

حل : در ابتدا یک الگوریتم ساده برای آن بنویسید :

۱. شروع

..... ۲.

..... ۳.

..... ۴.

..... ۵.

..... ۶.

۷. پایان

۴-۲-۲. تکرار با تعداد دفعات مشخص

ملاحظه کردید که برای تکرار با تعداد دفعات نامشخص، حلقه‌ی while مناسب بود.

در این جلسه، به ساختار تکراری خواهیم پرداخت که تعداد تکرارها در آن، از قبل مشخص

است.



مثال ۴۰: با استفاده از حلقه‌ی while، برنامه‌ای را به زبان C بنویسید که جمله اول یک تصاعد حسابی (a)، قدر نسبت آن (d) و تعداد جملات (n) را دریافت کند و بعد از چاپ n جمله‌ی اول آن، حاصل جمع جملات آن تصاعد حسابی را محاسبه نماید.

حال به ساختار زیر دقت کنید :

```
int counter=1;

while(counter<=n){
    float term= a+(counter-1)*d;
    printf("a(%d) = %.3f\n",counter,term);
    sum+= term;
    counter++;
}
```

نقش هر کدام از اجزای کلیدی این حلقه را تشریح کنید و در صورت عدم وجود هر بخش، چه مشکلاتی را خواهیم داشت؟

int counter = 1;

counter=>n

Counter++;

۱-۲-۲-۴. حلقه‌ی for در زبان C

برای سهولت در استفاده از ساختارهای تکرار مبتنی بر اندیس، ساختار حلقه‌ی for

مورد استفاده قرار می‌گیرد. در این ساختار، هر سه عنصر کلیدی فوق، در یک خط بیان می‌شود. ساختار این حلقه به شکل زیر است :

```
int counter;
for(counter=1; counter<n; counter++) {
    // some program
}
```

مثال ۴۱: برنامه‌ی مثال قبل را با استفاده از حلقه‌ی for، پیاده‌سازی کنید؛ سپس، برنامه را به گونه‌ای تغییر دهید که نمایش و محاسبه‌ی حاصل جمع، صرفاً برای جملات زوج صورت بگیرد.

مثال ۴۲: الگوریتمی را بنویسید که یک عدد را دریافت کند و با استفاده از شکل *، مثلثی قائم الزاویه و متساوی الساقین را در صفحه ترسیم نماید. سپس، الگوریتم خود را به زبان C پیاده‌سازی کنید.

مثال ورودی : ۵

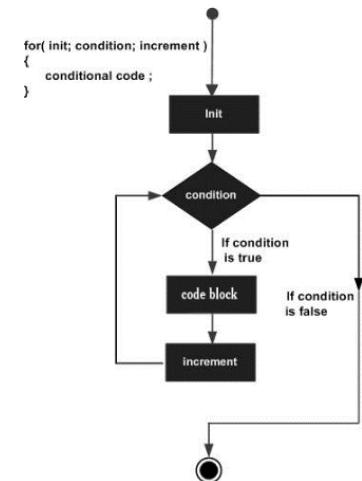
خروجی :

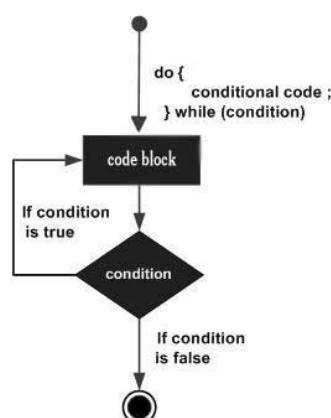
**

*

۳-۲-۴. ساختاری تکراری do-while

در بخش‌های قبلی مشاهده کردیم که در ساختارهای تکراری for و while، در ابتدا





شرط حلقه بررسی می‌شد؛ سپس در صورتی که شرط بوقرار بود، تکرار دستورات نوشته شده در حلقه ادامه می‌یافتد.

اما ممکن است در مواقعي قصد داشته باشيم که شرط ادامه یافتن حلقه را در پياناجرای حلقه بررسی نمایيم. در اين صورت استفاده از حلقه while ممکن است برنامه را پیچیده‌تر کند. بنابراین در زیان C ساختاري به نام do – while به وجود آمده است که با حلقه while تفاوت‌هایی دارد.

```

do {
    statement(s);
} while( condition );
  
```

مثال ۴۳: برنامه‌ای را به زبان C بنویسید که مجموع اعداد صحیح ورودی را محاسبه کند و در صورتی که کاربر عدد ۱ را وارد کرد، دریافت ورودی را متوقف کند.

۳-۴. دستورات break و continue

گاهی لازم می‌شود که روال اجرای یک حلقه را متوقف نماییم. یکی از راهکارهای قابل استفاده برای این منظور، استفاده از دستور break است. دستور break می‌تواند در داخل ساختارهای تکراری و ساختار switch case قرار گیرد.

همچنین در صورتی که تمایل داشته باشید که تحت شرایطی، اجرای باقی دستورات حلقه منتفی شده و دور بعدی حلقه به اجرا در آید، از دستور continue استفاده می‌شود.

مثال ۴۴: با استفاده از حلقه for و دستور continue برنامه‌ای به زبان C بنویسید که اعداد زوج بین ۱ الی ۲۰ را چاپ کند.

توجه : معمولاً استفاده از این دستورات در برنامه نویسی ساخت یافته توصیه نمی‌شود.

مثال ۴۵: بنظر شما چه راهکار جایگزینی برای استفاده از دستور break در حلقه while

می‌توان ارائه کرد؟

درس ۵ : آرایه‌ها

در پایان این بخش، فرآگیران باید با مطالب زیر آشنا شده باشند:

- با آرایه‌ها و ویژگی‌های آن در زبان C آشنا شود.
- بتواند اعضای آرایه را پیمایش کند.
- با آرایه‌ای از کاراکترها (رشته‌ها) آشنا شود.
- با آرایه‌های دو بعدی (ماتریس‌ها) آشنا شود.

۱-۵. معرفی آرایه‌ها

آرایه‌ها، مجموعه‌ای از داده‌ها و اطلاعات هستند که با یک دیگر مرتبط هستند و در غالب یک اسم ذخیره می‌شوند و دارای و یکسانی هستند.

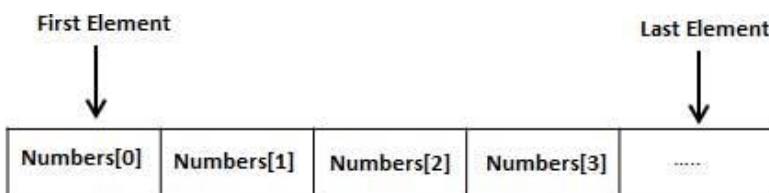
همه‌ی اعضای واقع شده در یک آرایه، دارای یک نوع و یک اسم هستند و با استفاده از می‌توان عناصر آن را از هم بازنگشتنی کرد.

آرایه‌ها، نوعی از داده ساختارها هستند که می‌توانند دنباله‌ای با سایز ثابت از عناصر همنوع را در خود ذخیره کند. آرایه‌ها، برای ذخیره سازی دنباله‌ای از داده‌ها به کار می‌روند، اما می‌توان به آن‌ها، به صورت مجموعه‌ای از متغیر‌های همنوع نگریست.

مثال ۴۶: فرض کنید قصد دارید که ۱۰ عدد صحیح را ذخیره کنید. بجای آنکه ۱۰ متغیر عدد صحیح با نام‌های number1 ، number2 ، number10 و ... ذخیره کنید، کافیست یک آرایه‌ی ۱۰ عضوی از نوع عدد صحیح به نام numbers و با ظرفیت ۱۰ عضو ایجاد کنید و برای دسترسی به یکایک آن‌ها، به صورت زیر از اندیس‌ها استفاده کنید.

numbers[0] , numbers[1] , ...

همه ارایه‌ها، متشکل هستند از مجموعه‌ای از خانه‌های حافظه، به طوری که خانه‌ای با آدرس بزرگ‌تر، متناظر است با خانه اول ارایه و خانه‌ای با آدرس کوچک‌تر، متناظر است با عنصر انتهایی ارایه.



۲-۵. تعریف ارایه‌ها در زبان C

برای ذخیره سازی یک متغیر مانند number و از جنس int به شکل زیر تعریف را انجام می‌دادیم:

int number;

حال، مجموعه‌ای از عناصر را به صورت زیر، مشخص می‌کنیم:

type arrayName [arraySize];

هر کدام از اجزای تعریف، به شرح زیر هستند:

type .۱

arrayName .۲

arraySize .۳

بنابراین، ارایه‌ای از اعداد صحیح با سایز ۱۰ و از نوع int به صورت زیر تعریف می‌شود:

۵-۳. مقداردهی اولیه به آرایه‌ها

یکی از راه‌های مقداردهی اولیه آرایه‌ها به صورت زیر است:

`double balance[5] = {200, 300, 325, 12, 3};`

همچنین، چنانچه تمام پنج عنصر آرایه‌ی بالا تعیین نشوند، عناصر تعیین نشده با عدد صفر جایگزین می‌شوند.

در صورتی که طول آرایه را تعیین نکنیم، به تعداد عناصر داده شده در بین دو آکلاد، به آرایه حافظه تخصیص داده می‌شود. به عنوان مثال :

`double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};`

در این صورت، آرایه‌ی بالا به صورت زیر در حافظه ذخیره می‌شود:

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

مثال ۴۷: با استفاده از زبان برنامه نویسی C، برنامه‌ای بنویسید که مجموعه‌ای از اعداد را دریافت کرده و مقدار کمینه‌ی آن را (به همراه شماره‌ی اندیس آن را) در خروجی چاپ کند.

مثال ۴۸: برنامه‌ای را به زبان C بنویسید که مجموعه‌ای از عناصر را دریافت کرده و ضمن تعیین کوچکترین عنصر، آن را با اولین عنصر داده شده عوض کند.

مثال ۴۹: (مرتب‌سازی انتخابی)

برگرفته از مجله‌ی

فرادرس

مرتب‌سازی انتخابی یک الگوریتم مرتب‌سازی ساده است. این الگوریتم مرتب‌سازی

یک روش مرتب‌سازی درجا و مبتنی بر مقایسه است که در آن لیست به دو بخش

تقسیم می‌شود، بخش مرتب در سمت چپ و بخش نامرتب در سمت راست قرار

می‌گیرد. در ابتدا بخش مرتب خالی است و بخش نامرتب کل لیست را شامل می‌شود.

کوچکترین عنصر از آرایه نامرتب انتخاب می‌شود و با عنصری که در انتهای سمت

چپ قرار دارد تعویض می‌شود. این فرایند تا زمانی که به کران سمت راست آرایه

بررسیم ادامه می‌یابد. آرایه‌ای که در تصویر زیر نشان داده شده را در نظر بگیرید:



برای این که دو عنصر اولیه لیست مرتب باشند، باید همه لیست به صورت ترتیبی بررسی شوند. موقعیت نخست حاوی مقدار ۱۴ است که در حال حاضر در وضعیت مرتب قرار دارد، کل لیست را جستجو می‌کنیم و درمی‌یابیم که ۱۰ مقدار کمتری دارد:



بنابراین جای ۱۴ و ۱۰ را عوض می‌کنیم. پس از چرخه نخست مشخص می‌شود که ۱۰ کوچک‌ترین عنصر لیست است و در موقعیت ابتدایی لیست مرتب قرار می‌گیرد.



برای موقعیت دوم که اینک مقدار ۳۳ را در خود جای داده، باقی مانده لیست را به همان روش خطی بررسی می‌کنیم.



درمی‌یابیم که ۱۴ دومین مقدار کوچک در آرایه ما است و باید در جایگاه دوم لیست قرار داشته باشد، این مقادیر را تعویض می‌کنیم

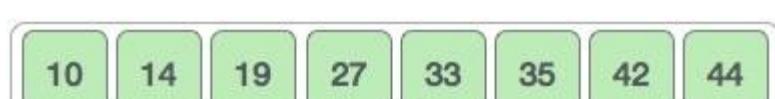
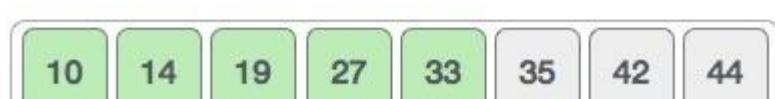
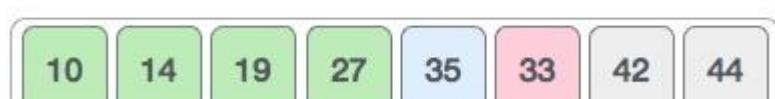
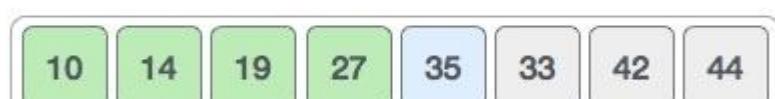
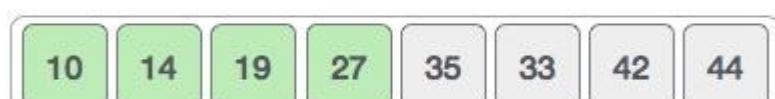
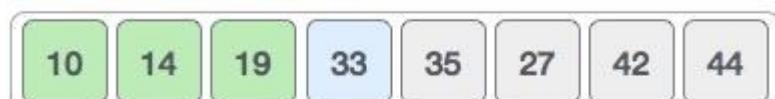


پس از چرخه دوم اینک دو مقدار داریم که در ابتدای لیست به صورت مرتب قرار گرفته‌اند:



همین فرایند در مورد بخش باقی مانده لیست نیز به کار گرفته می‌شود. در ادامه کل

فرایند مرتب‌سازی به صورت مرحله به مرحله و تصویری ارائه شده است:



۴-۵. رشته‌ها، آرایه‌ای از کاراکترها

رشته‌ها، در اصل آرایه‌هایی یک بعدی هستند که با کاراکتر تهی (کاراکتر ۰\0)، پایان یافته‌اند.

به عنوان مثال، رشته‌ی Hello به صورت زیر، در آرایه ذخیره می‌شود:

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

رشته‌ها، آرایه‌ای از کاراکترها هستند که برای سهولت، می‌توانند به صورت زیر تعریف شوند:

```
char greeting[6] = "Hello";
```

رشته‌ی بالا، به صورت زیر در حافظه ذخیره می‌شود:

Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

توجه مهم : مسئولیت حفظ کران‌های اندیس‌ها، بر عهده‌ی برنامه نویس است و در صورت

عدم رعایت آن، برنامه دچار خطاهای زمان اجرا خواهد شد.

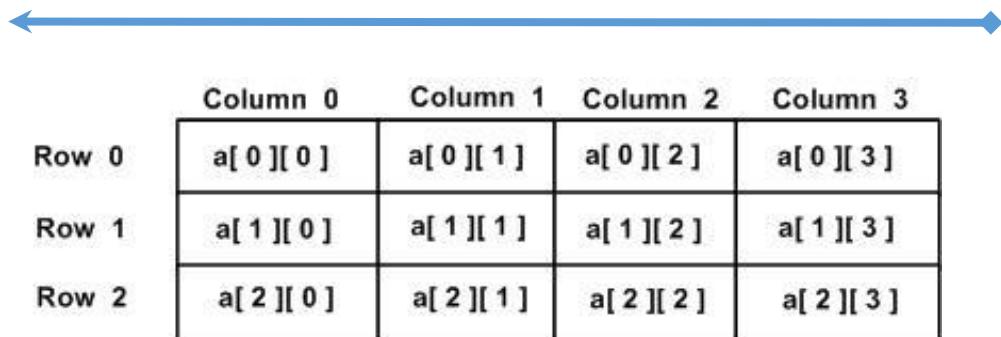
۵-۵. آرایه‌های دو بعدی و چند بعدی

آرایه‌ی دو بعدی، آرایه‌ای از آرایه‌های دو بعدی است و بصورت زیر تعریف می‌شود:

```
type name[size1][size2];
```

اجزای این تعریف، مشابه با آرایه‌ی یک بعدی است. آرایه‌ی دو بعدی را می‌توان به

صورت زیر، در حافظه‌ی اصلی فرض کرد:



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

۵-۱. نحوه دسترسی به عناصر آرایه‌ی دو بعدی

همچنین، نحوه دسترسی به اعضای این آرایه به صورت زیر است.

name [size1] [size2];

یادآوری : ماتریسی که تمامی درایه‌های بالای قطر اصلی آن صفر باشد، پایین مثلثی است.

به طوری که size1 برابر با اندیس سطر و size2 برابر با اندیس ستون خواهد بود.

مثال ۵: برنامه‌ای را به زبان C بنویسید که یک ماتریس مربعی را دریافت کرده و بررسی

کند که آیا ماتریس پایین مثلثی است یا خیر؟

درس ۶ : توابع

در پایان این بخش، فرآگیران باید با مفاهیم زیر آشنا شده باشند:

- انواع دستورات در یک برنامه‌ی کامپیوتری
- مفهوم فراخوانی
- روش تعریف توابع و به کارگیری آن در زبان C
- آشنایی با مفهوم میدان دید (Scope)
- آشنایی با توابع کتابخانه‌ای (کتابخانه‌ی math)

۶-۱. انواع دستورات در کامپیوتر

برنامه‌ها، از دستورها تشکیل می‌شوند. انواع (از نگاهی دیگر) دستورها به صورت زیر

هستند:

۱. تعریف و تخصیص (Assignment / deceleration)

۲. رفتن به (go to)

۳. فراخوانی عملیات (Call Operation)

۴. بازگشت (return)

همچنین، دستورات می‌توانند به صورت ساده استفاده شوند و با به صورت پیچیده، در بلوک‌ها و ساختارهای تکرار مورد استفاده قرار بگیرند. اگر توابع نباشند، بسیاری از سیستم‌های نرم افزاری قابل پیاده‌سازی نیستند. توابع ما را قادر می‌سازند تا بتوانیم برنامه‌های خود را به صورت پیمانه‌ای بنویسیم. اگر از توابع استفاده نکنیم، ناچار به تکرار کد هستیم که برنامه را ناخوانا و پیچیده می‌کند.

به طور کلی می‌توان گفت که هدف اصلی استفاده از توابع، شکستن برنامه‌ها به قطعه‌های کوچک‌تر و با قابلیت استفاده‌ی مجدد است.

۶-۲. مزایای استفاده از توابع

استفاده از توابع دارای مزایای متعددی است، از جمله:

۱. شکستن مسئله به زیر مسائل کوچک‌تر
۲. ردیابی سریع‌تر و راحت‌تر برنامه
۳. کنترل برنامه به صورت قطعه قطعه
۴. درک بهتر برنامه
۵. جلوگیری از به وجود آمدن کد تکراری
۶. امکان استفاده از توابع به صورت فراخوانی (قابلیت استفاده‌ی مجدد)
۷. استفاده‌ی بهینه از حافظه جهت تخصیص به متغیرها

۶-۳. تعریف و به کارگیری توابع در زبان C

برای تعریف و استفاده از یک تابع، باید وظیه‌ی تابع مشخص شود:

۱. تابع مورد نظر، چه فرآیندی را بایستی اجرا نماید؟
۲. برای این که تابع محاسبات مطلوب ما را انجام دهد، به چه داده‌هایی نیاز دارد؟
۳. تابع چه خروجی‌ای دارد؟ و این خروجی به چه صورتی مورد استفاده قرار

بنابراین برای تعریف و استفاده از تابع، در ابتدا لازم است **الگوی آن** را به کامپایلر معرفی کنیم؛ به این کار، Function Prototype گفته می‌شود؛ سپس لازم است تابع را تعریف نماییم؛ برای این کار، دستورات تابع را به صورت تکه برنامه‌ای مستقل می‌نویسیم. در نهایت از تابعی که ساختیم در جای مورد نظر استفاده می‌کنیم و برای این منظور، تابع را فراخوانی می‌کنیم.

۶-۳-۱. گام اول: معرفی الگوی تابع (Function Prototype)

استفاده از تابع، با استفاده از فراخوانی آن صورت می‌گیرد؛ اما لازم است پیش از فراخوانی تابع، الگوی آن را به کامپایلر معرفی کنیم؛ در این صورت، کامپایلر شکل تابع را با این الگو تطبیق می‌دهد و در صورت عدم مطابقت، خطای کامپایل به وجود می‌آید. الگوی تابع دارای سه جزء اصلی است:

```
return_type function_name( parameter list );
```

return_type .۱

function_name .۲

parameters (arguments) .۳

آرگومان‌های تابع، در اصل همان ورودی‌های تابع هستند که ارتباط تابع را با دنیای بیرون برقرار می‌کنند.

نکته: هدف از تعریف Prototype در توابع، تطبیق آن‌ها با موارد زیر است:

۱. تعداد آرگومان‌ها

۲. نوع داده‌ی آرگومان‌ها

۳. ترتیب صحیح آرگومان‌ها

۴. نوع برگشتی تابع

۶-۳-۲. گام دوم: تعریف دستورات تابع

تابع، یک تکه برنامه‌ی مستقل است که قسمتی از یک برنامه را تشکیل می‌دهد. فرم

کلی تعریف یک تابع به صورت زیر است:

```
return_type function_name( parameter list ) {
    body of the function
}
```

در سطر اول، لازم است نوع برگشتی تابع، نام تابع و آرگومان‌های آن را معرفی کنیم.

به این بخش از تابع، امضای آن تابع (Signature) گفته می‌شود؛ امضای تابع باید مطابق با پروتوتایپ آن باشد.

در ادامه، دستورات تابع در بین دو آکلاد نوشته می‌شود؛ به هر آنچه که در بین دو آکلاد نوشته می‌شود، بدن‌های تابع گفته می‌شود. در بدن‌های تابع همانند برنامه‌نویسی معمولی می‌توان متغیرهایی را تعریف کرد و دستوراتی را نوشت. به متغیرهایی که در بدن‌های تابع تعریف می‌شود، متغیرهای محلی (Local Variables) گفته می‌شود.

متغیرهای محلی، منحصرا در داخل بلوکی دیده می‌شوند که در آن تعریف شده‌اند.

با رسیدن به دستور return ، مقدار نهایی محاسبات تابع به محل فراخوانی تابع ارسال



می‌گردد؛ در اینجا اجرای تابع خاتمه یافته و ادامه‌ی اجرای برنامه، از محل فراخوانی ادامه پیدا می‌کند.

نکته : با خارج شدن از بلوک تابع، متغیرهای محلی آن از حافظه پاک می‌شود.
(البته راهکارهایی برای جلوگیری از این اتفاق وجود دارد).

نکته : توابع می‌توانند مقداری را برنگردانند؛ در این صورت، مقدار برگشتی آن‌ها void خواهد بود.

نکته: تابع می‌تواند هیچ آرگومانی را دریافت نکند.

۶-۳-۳. گام سوم : فراخوانی تابع

در نهایت، هرگاه نوبت به اجرای تابع مورد نظر رسید، بایستی آن تابع را فراخوانی نمود. در این صورت، اجرای برنامه به ابتدای تابع منتقل می‌شود و پس از اتمام اجرای تابع، به خط بعد از فراخوانی بر می‌گردد. فراخوانی به صورت زیر انجام می‌شود:

`FunctionName(args);`

مثال ۱۵: تابعی را به زبان C بنویسید که عددی صحیح را دریافت کرده و فاکتوریل آن را چاپ کند.

نکته : توابع می‌توانند در بدنه‌ی خود، یک دیگر را فراخوانی کنند.

مثال ۱۶: هر عدد طبیعی بزرگ‌تر از ۱ که جز خودش و ۱ مقسوم علیه دیگری نداشته باشد، اول است. تابعی به نام `isPrime` بنویسید که یک عدد صحیح مثبت را به عنوان آرگومان ورودی بگیرد و در صورت اول بودن، مقدار ۱ و در غیر این صورت، مقدار ۰ را به محل فراخوانی بازگرداند.

مثال ۱۷: با استفاده از تابع مثال قبل، برنامه‌ای را بنویسید که یک عدد صحیح و مثبت را از کاربر دریافت کند و بررسی کند که آیا می‌توان آن را به صورت جمع دو عدد اول

نوشت یا خیر؟ اگر چنین است، آن دو عدد اول را چاپ کند و در غیر این صورت، پیغام را چاپ کند. در صورتی که امکان چاپ دو عدد اول به چند طریق ممکن است، تمامی حالات را چاپ کند.

به عنوان مثال به ازای دریافت عدد ۱۰ باید خروجی زیر را چاپ کند.

$$10=3+7$$

$$10=5+5$$

و در صورت دریافت عدد ۳ باید Impossible را چاپ نماید. (میانترم پاییز ۹۷)

۶-۴. میدان دید (Scope)

هر کدام از متغیرهایی که در برنامه تعریف می‌شوند، در محدوده‌های خاصی از برنامه قابل رویت، استفاده و تغییر هستند. به محدوده‌ای از برنامه که متغیری در آن قابل رویت است، میدان دید آن متغیر (Scope) گفته می‌شود. میدان‌های دید برای هر متغیر، از ابتدای علامت { آغاز شده و به } ختم می‌شود. در کتاب درسی دایتل، ۴ نوع میدان دید معرفی شده است که در اینجا به دو مورد آن می‌پردازیم:

۱. میدان دید در فایل (File Scope) :

۲. میدان دید در بلوک (Block Scope) :

مثال ۵۴: کد زیر را در نظر بگیرید:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int x=10;
7     if(x<100) {
8         int x=20;
9         int y=10;
10        printf ("x1 = %d\n", x);
11        printf ("y1 = %d\n", y);
12    }
13    printf ("x2 = %d\n", x);
14    printf ("y2 = %d\n", y);
15
16    return 0;
17 }
18

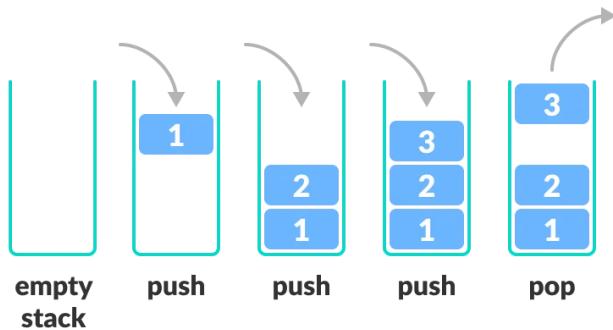
```

الف) کد دارای یک خطای کامپایلری است. ضمن بیان علت آن خطأ، آن را رفع کنید.

ب) پس از رفع خطای کامپایل، خروجی آن را بنویسید.

۶-۵. پشته‌ی فراخوانی (Call Stack)

گاهی تمایل دارید که داده‌هایی را با ترتیب ذخیره کنید و با هربار درخواست، به آخرین داده‌ی ذخیره شده دست پیدا کنید. بنابراین آن‌ها را به صورت یک پشته‌ای از داده‌ها ذخیره می‌کنید.

**مثال ۵۵:** عبارت درست را انتخاب کنید :

در یک پشته، اولین عضوی که به آن وارد (push) می‌شود (اولین / آخرین) عنصری است که از پشته خارج (Pop) می‌شود. بنابراین به پشته (.....) نیز گفته می‌شود.

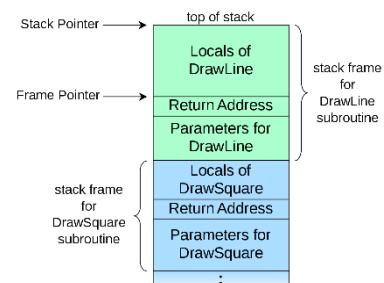
جالب است بدانید که توابع نیز با فراخوانی یک دیگر، پشته‌ای از فراخوانی را در حافظه درست می‌کند.

مثال ۵۶: کد زیر را در نظر بگیرید. پشته‌ی فراخوانی این تابع ترسیم شده است. ضمن بیان خروجی قطعه کد زیر، علت شکل‌گیری پشته‌ی فراخوانی آن را شرح دهید.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 void drawLine(int);
4 void drawSquare(int);
5
6 int main()
7 {
8     drawSquare(2);
9     return 0;
10 }
11
12 void drawSquare(int length) {
13     printf("I draw a Square with Area %d\n", length*length);
14     drawLine(length+10);
15 }
16
17 void drawLine(int length) {
18     printf("I draw a Line with Length %d\n", length);
19 }
20

```



۶-۶. فراخوانی با ارجاع و فراخوانی با مقدار

همان‌طور که به خاطر دارید، اطلاعات توابع در قالب آرگومان‌ها به آن ارسال می‌شوند. در حالت عادی، با اتمام اجرای بدنی تابع، تمامی متغیرهای محلی آن نیز از بین می‌روند؛ در نتیجه، متغیری که در بیرون از تابع تعریف شده و به تابع داده شده است تغییر نمی‌کند و یک کپی از آن در بدنی تابع ایجاد می‌شود؛ به این روش، فراخوانی با مقدار گفته می‌شود. اما گاهی ممکن است بخواهیم یک متغیر را در خارج از تابع تغییر دهیم؛ در نتیجه باید آدرس آن را به تابع بدهیم و از طریق آدرس، آن را تغییر دهیم. به این کار، فراخوانی با مرجع گفته می‌شود.

مثال ۵۷: کد زیر را در نظر بگیرید و خروجی آن را بنویسید؛ سپس نوع فراخوانی آن را مشخص کنید.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 void printDouble(int);
4
5 int main()
6 {
7     int x=10;
8     printf("%d\n", x);
9     printDouble(x);
10    printf("%d\n", x);
11    return 0;
12 }
13
14 void printDouble(int x) {
15     x*=2;
16     printf("%d\n", x);
17 }
18

```

مثال ۵۸: برنامه‌ای را به زبان C بنویسید که یک آرایه را در ورودی دریافت کرده و عناصر آن را دوباره کند.

۷-۶. بازگشت (Recursion)

تا کنون برنامه‌هایی را مشاهده کردیم که ترکیبی از چند تابع بودند که یک دیگر را فراخوانی می‌کردند. گاهی در محاسبات تکراری، هر بخش از محاسبه می‌تواند بر اساس بخش قبلی اش انجام شود؛ در نتیجه می‌توان هر کدام از نسخه‌های آن را بر اساس یک نسخه با اندازه‌ی کوچک‌تر بیان کرد. به عنوان مثال، برای محاسبه‌ی فاکتوریل یک عدد

داریم :

$$n! = n \times (n - 1)!$$

در این مثال، مشاهده کردیم که که فاکتوریل عددی مثل n ، بر اساس فاکتوریل $n-1$ بیان شد؛ این نگاه می‌تواند به درک بهتر مسئله کمک بهتری کند. اما باید توجه کرد که در صورتی می‌توان از این رویکرد استفاده کرد که مسئله، دارای یک شرط پایه باشد تا بعد از تعداد متناهی از فراخوانی تابع به یک جواب نهایی رسید؛ به عنوان نمونه، در

حالت پایه‌ای برای محاسبه‌ی فاکتوریل داریم :

$$0! = 1$$

به منظور پیاده‌سازی این مسائل، می‌توان توابعی را ایجاد کرد که در آن‌ها، تابع اقدام به فراخوانی خودش می‌نماید. هر تابع بازگشتی، با دریافت ورودی‌هایش، می‌تواند یکی از دو مسیر زیر را ادامه دهد:

۱. حالت پایه‌ای را اجرا کند (در مثال قبل، این حالت پایه‌ای می‌شود $0!$)
۲. مجدداً خودش را با ورودی‌های جدید فراخوانی کند.

بنابراین، به طور کلی، یک تابع بازگشتی لازم است حداقل دو بخش اساسی

داشته باشد :

۱. شرط پایه‌ای (base case)

۲. بازگشت (Recursion)

در صورت عدم وجود شرط پایه‌ای، فراخوانی تابع به تعداد دفعات نامتناهی صورت



می‌گیرد و درنهایت با پر شدن حافظه (که حاوی پشته‌ی فراخوانی است) سرریز پشته

رخ می‌دهد (Stack overflow).

مثال ۵۹: برنامه‌ای را به زبان C بنویسید که عددی را از ورودی دریافت کرده و فاکتوریل آن را

محاسبه کند.

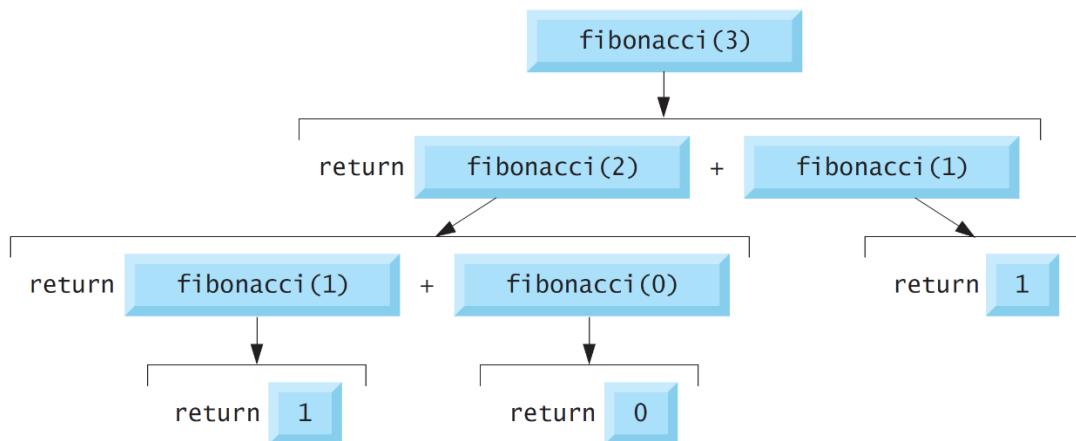
مثال ۶۰: برنامه‌ای را به زبان C بنویسید که عددی را از ورودی دریافت کرده و جمله‌ی n ام

دباله‌ی فیبوناچی را بیابد. آیا بنظر شما، پیاده‌سازی بازگشتی آن به صرفه‌تر است یا

غیربازگشتی؟

$$fibo(n) = fibo(n - 1) + fibo(n - 2)$$

$$fibo(0) = 0 \text{ and } fibo(1) = 1$$



۶-۷-۱. بازگشت یا تکرار؟

بر اساس مثال اخیر می‌توان مشاهده کرد که بازگشت، بسیار شبیه به ساختارهای تکراری عمل می‌کند و در نهایت، منجر به تکرار در فراخوانی می‌شود. شباهت‌ها و تفاوت‌هایی بین استفاده از ساختارهای تکراری و روابط بازگشتی وجود دارد که به شرح زیر است:

۱. در تکرار، به صورت صریح از ساختارهای تکراری (حلقه‌ها) استفاده می‌کنیم. اما در

بازگشت، عمل فراخوانی همان تابع را به صورت تکراری انجام می‌دهیم.

۲. توابع بازگشته بر مبنای ساختار کنترلی انتخاب استوار هستند؛ اما ساختارهای

تکراری بر مبنای ساختار کنترلی تکرار استوار می‌باشند.

۳. در هر دو روش، بر اساس یک شرط پایانی تکرار پایان می‌یابد. در ساختار تکراری بر

مبنای عدم برقراری یک شرط و در بازگشت، بر مبنای برقراری شرط پایه‌ای تکرار

پایان می‌یابد.

۴. در ساختارهای کنترلی تکراری می‌توان از یک شمارنده استفاده کرد؛ اما در بازگشت،

با کوچکتر کردن مسئله، به تکرار ادامه می‌دهد.

۵. هر دو می‌توانند به فاجعه‌ی تکرار نامتناهی دچار شوند! (شما بگویید در هر کدام از این

حالات، چگونه این اتفاق پیش می‌آید).

درس ۷ : اشاره‌گرها

در پایان این بخش، فرآگیران باید با مطالب زیر آشنا شده باشند:

- با مفهوم آدرس یک متغیر آشنا شود.
- تفاوت بین آدرس یک متغیر و مقدار آن را متوجه شود.
- با مفهوم اشاره‌گر آشنا شود.
- بتواند با داشتن آدرس یک متغیر، مقدار آن را تغییر دهد.
- ارتباط آرایه و اشاره‌گر را متوجه شود و بتواند با اشاره‌گر، به آرایه‌ای از عناصر اشاره کند.
- با عملگرهای محاسباتی بر روی اشاره‌گرها آشنا شود.

۷-۱. مقدمه‌ای بر اشاره‌گرها

یکی از مهم‌ترین قابلیت‌های زبان C، امکان استفاده از اشاره‌گرها می‌باشد. اشاره‌گرها، کارایی، قدرت و انعطاف پذیری برنامه‌ها را بیشتر می‌کنند. علاوه بر نیاز به بالا بردن کارایی در برنامه‌ها، گاهی شرایطی رخ می‌دهد که نمی‌توانیم از روش‌های قبلی برای برنامه نویسی، استفاده کنیم و ناچاریم که از اشاره‌گرها استفاده کنیم.

اشارة‌گرها، ابزارهایی قدرتمند هستند که اجازه می‌دهند که به طور مستقیم با حافظه در ارتباط باشیم و با حذف نقل و انتقال‌های اضافی بین خانه‌های حافظه، سرعت را افزایش داده و کارایی را بالاتر ببریم. هر کدام از خانه‌های حافظه، دارای یک شماره هستند که از آن به عنوان آدرس حافظه یاد می‌شود. با تعریف هر متغیر، خانه‌ای در حافظه به آن تخصیص داده می‌شود و مقدار متغیر، در آن خانه از حافظه ذخیره می‌شود.

مثال ۱۶: شکل زیر، بخشی از حافظه را نشان می‌دهد.

فرض کنید تعریف کنیم : `int x = 10`

	1000
	1001
10	1002
	1003

یکی دیگر از راههایی که می‌توان مقدار متغیری را به توان ۳ رساند ، تعریف تابعی است که ورودی دریافتی را سه مرتبه در خود ضرب کرده و سپس ، آن را `return` کند.

برنامه‌ای بنویسید که این کار را انجام دهد. سپس به سؤالات زیر پاسخ دهید:

الف) آیا متغیر اصلی نیز تغییر می‌کند؟

ب) این تغییر، چگونه صورت می‌گیرد؟ به صورت شماتیک، آن را توضیح دهید.

ج) چه کاری را انجام دهیم، تا کارایی برنامه بالاتر برود؟

مثال ۱۷: چرا در روش قبل، مقدار متغیر تغییر نمی‌کرد؟

چه باید کرد؟ باید تابعی را ایجاد کنیم که تغییرات مورد نیاز را، دقیقاً در یک خانه

ی خاصی از حافظه و بدون نقل و انتقال اضافی، اعمال کند کند. چنین تابعی باید

ویژگی‌های زیر را داشته باشد:

۱. باید آدرسی را دریافت کند که در آن آدرس، قصد تغییر را داریم.

۲. خروجی ندارد. چون تغییرات به صورت درجا اعمال می‌شود.



برای این منظور، ورودی تابع مد نظر، می‌بایست آدرسی از حافظه باشد که می‌خواهیم محتوای آن را تغییر دهیم؛ این آدرس، یک اشاره گری است به یک نوع داده از جنس int . اشاره گر به یک نوع داده را از جنس int، به صورت زیر تعریف می‌کنیم:

```
int* aPtr;
```

از این عبارت، معانی زیر را می‌توان دریافت کرد:

۱. aPtr ، یک اشاره گر است به متغیری از جنس int .

۲. aPtr ، محل ذخیره سازی (آدرس) یک متغیر از نوع int است.

۱-۱-۷ عملگرهای & و *

در این بخش، با دو عملگر آشنا می‌شویم:

۱. عملگر & که به نام عملگر Address می‌شناسیم.

۲. عملگر * که از آن به عنوان عملگر deference یاد می‌شود.

عملگر دوم، به دو طریق می‌تواند به کار برود:

اگر در تعریف متغیر به کار برود، بدان معنا خواهد بود که آن متغیر، حاوی آدرسی

است به یک نوع داده‌ی خاص. به طور کلی، چنین متغیری به صورت زیر تعریف می‌شود:

```
variable* dataType;
```

اگر در حین استفاده از متغیر به کار برود، به معنای دریافت محتوای ذخیره شده

در آدرس است.

مثال ۳۶: با استفاده از مفهوم اشاره گر، تابعی را تعریف کنید که موارد زیر را انجام دهد:

۱. مقدار متغیر را بصورت درجا، به توان ۳ برساند.

۲. آدرس متغیر و مقدار آن را، پیش از دوباره کردن، چاپ کند.

نکته: اگر aPtr اشاره گر (آدرس) باشد، *aPtr مقداری را خواهد داد که در آدرس ذخیره شده است. به روش پیاده‌سازی تابع قبلی، Call by Reference aPtr گفته می‌شود.

۲-۷. کاربردهایی از اشاره گرهای

اشارة گرهای، در موارد زیر می‌توانند کاربرد داشته باشند.

۱. توابع، حداکثر یک خروجی دارند. اما به وسیلهٔ اشاره گرهای می‌توان کاری کرد

که از توابع بیش از یک خروجی دریافت کنیم.

۲. استفاده از اشاره گر، می‌تواند باعث افزایش سرعت برنامه شود.

۳. اشاره گرهای لازمهٔ پیاده‌سازی بسیاری از الگوریتم‌های معروف و داده

ساختارهای معروف هستند.

۳-۷. چند نکته در خصوص تعریف اشاره گرهای

۱. بهتر است که جهت خوانایی برنامه، از پسوند Ptr بعد از نام متغیر استفاده کنید.

۲. اگر برای یک اشاره گر، مقداردهی خاصی را مدنظر ندارید، مقدار آن را NULL

بگذارید.

۴-۷. مثال‌های بیشتر

مثال ۶۴: برنامه‌ای را به زبان C بنویسید که با استفاده از اشاره‌گرهای، جای دو متغیر را تعویض کند.

مثال ۶۵: برنامه‌ای را به زبان C بنویسید و در آن، تابعی به نام swap را تعریف کنید. این تابع باقی‌ماندهٔ آرایه را باهم تعویض کند.

مثال ۶۶: (مرتب سازی حبابی) مرتب سازی حبابی یک الگوریتم مرتب سازی ساده است.

این الگوریتم مرتب سازی یک الگوریتم مبتنی بر مقایسه است که در آن هر جفت از

عناصر مجاور با هم مقایسه می شوند و در صورتی که در ترتیب مطلوب نباشند با هم

تعویض می شوند. این الگوریتم برای مجموعه داده های بزرگ مطلوب نیست، زیرا

پیچیدگی حالت میانگین و بدترین حالت آن برابر با $O(n^2)$ است که در آن n تعداد

آیتم هایی است که باید مرتب شوند

۱-۴-۷. طرز کار مرتب سازی حبابی

به عنوان مثال یک آرایه نامرتب را در نظر می گیریم:

14	33	27	35	10
----	----	----	----	----

مرتب سازی حبابی کار خود را با نخستین دو عنصر آغاز می کند و آنها را مقایسه

می کند تا ببینند کدام یک بزرگ تر است:

مرتب سازی حبابی کار خود را با نخستین دو عنصر آغاز می کند و آنها را مقایسه

می کند تا ببینند کدام یک بزرگ تر است:

14	33	27	35	10
----	----	----	----	----

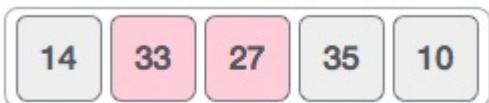
در این مثال ۳۳ از ۱۴ بزرگ تر است و از این رو این دو عنصر هم اینک در وضعیت

مرتب هستند، سپس دو مقدار ۳۳ و ۲۷ را مقایسه می کنیم:

14	33	27	35	10
----	----	----	----	----

در می یابیم که ۲۷ کوچک تر از ۳۳ است و لذا موقعیت این دو عنصر باید تعویض

شود.



اینک آرایه به صورت زیر درمی‌آید:



سپس ۳۳ و ۳۵ را مقایسه می‌کنیم. مشخص است که این دو مقدار در وضعیت مرتب

هستند.



سپس دو مقدار بعدی یعنی ۳۵ و ۱۰ مقایسه می‌شوند



می‌دانیم که ۱۰ کوچک‌تر از ۳۵ است. از این رو این دو مقدار مرتب نیستند.



این دو مقدار را تعویض می‌کنیم. درمی‌یابیم که به انتهای آرایه رسیده‌ایم. پس از یک

چرخه، آرایه به صورت زیر درمی‌آید:

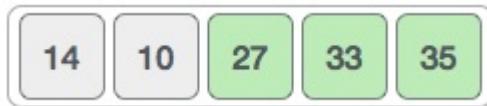


برای این که فرایند کار را خلاصه کنیم وضعیت آرایه را پس از هر چرخه مرتب‌سازی

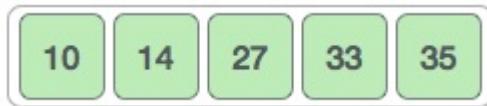
در ادامه ارائه کرده‌ایم. پس از چرخه دوم آرایه به صورت زیر درمی‌آید:



توجه کنید که پس از هر چرخه، دست کم یک مقدار تغییر یافته است



و زمانی که دیگر هیچ تعویضی رخ ندهد، الگوریتم مرتب‌سازی حبابی می‌فهمد که آرایه به طور کامل مرتب شده است.



۷-۵. ثابت کردن اشاره‌گر و مقادیر آن

همان‌طور که دیدیم، اشاره‌گرها به خانه‌هایی در حافظه اشاره می‌کنند که دارای یکی از انواع داده‌ها می‌باشد. هر کدام از مقادیر زیر، می‌توانند در طول اجرای برنامه ثابت بوده و یا تغییر کنند:

۱. اشاره‌گر به یک متغیر.

۲. مقدار متغیری که اشاره‌گر به آن اشاره می‌کند.

در خیلی از مواقع می‌توان به منظور جلوگیری از تغییرات ناخواسته در اشاره‌گرها و یا مقادیر متغیرهایی که به آن‌ها اشاره می‌شود، اشاره‌گرها و مقدار آن‌ها را ثابت تعریف کرد؛ برای این منظور، بر اساس نیاز می‌توان از کلیدواژه‌ی **const** استفاده نمود. استفاده از این کلیدواژه بر اساس جدول زیر می‌تواند صورت گیرد.

اشارة‌گر به متغیر قابل تغییر است.	اشارة‌گر به متغیر ثابت است	
<code>const int* aPtr;</code>	<code>const int* const aPtr;</code>	محتوی مغایر ثابت است
<code>int *aPtr;</code> همان‌چیزی که تا الان می‌دانستیم.	<code>int *const aPtr;</code> آرایه‌ی معمولی	محتوی مغایر قابل تغییر است.

مثال ۷۶: در کدام یک از حالات گفته شده در جدول بالا، آرایه‌ای را خواهیم داشت که

مقدار آن قابل تغییر نیست؟

مثال ۷۸: کد زیر را در نظر بگیرید و در خصوص صحت آن بحث کنید.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int x,y;
7     int *const xPtr = &x;
8     *xPtr= 10;
9     printf("x = %d\n",x);
10    xPtr =&y;
11
12    return 0;
13 }
14

```

۷-۶. اشاره‌گرها و ارتباط آن‌ها با آرایه‌ها

همان‌طور که در بخش قبل دیدیم، آرایه‌ها اشاره‌گرهایی هستند که می‌توانند مقداری و اشاره‌گری داشته باشند.

توجه کنید که عبارات زیر معادل هستند:

```

int numbers[ ];
int *const numbers[ ];

```

در نتیجه، عبارت numbers (اسم آرایه) با آدرس numbers اولین عضو آن (یعنی عبارت) معادل خواهد بود؛ همچنین، عبارت a[i] (یعنی عضو آن دنباله) با عبارت زیر هم ارز خواهد بود:

`*(&numbers[i])`

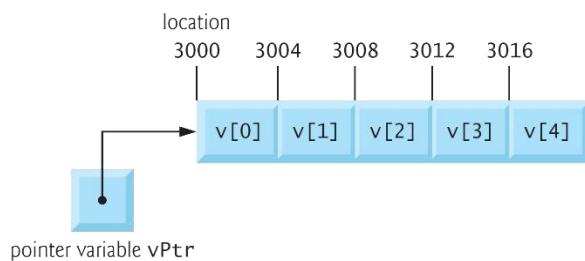
به فرم بالا، فرم آفست آرایه گفته می‌شود.

۷-۷. اعمال حسابی بر روی اشاره‌گرها

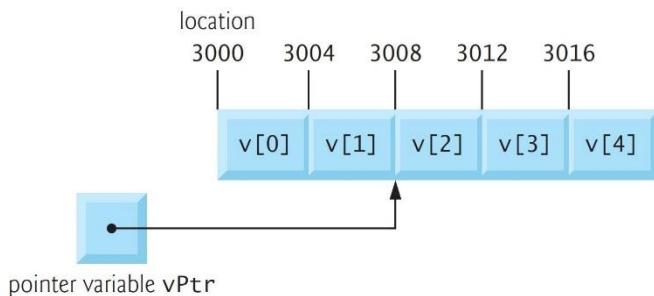
همان‌طور که گفته شد، اشاره‌گرها حاوی شماره‌های خانه‌های حافظه هستند؛ در نتیجه می‌توان بر روی آن‌ها اعمال حسابی جمع و تفریق را انجام داد. توجه کنید که این محاسبات در صوتی معنا دارند که اشاره‌گرهای مورد نظر، به بخش‌های مختلفی از یک آرایه‌ی واحد اشاره کنند.



۱. می‌توان به اشاره‌گری، عددی را اضافه کرد. در این صورت، حاصل آن، مکانی از حافظه است که بعد از ذخیره‌سازی داده در محل قبلی واقع شده است.



مثال ۶۹: به عنوان مثال، فرض کنید که v یک آرایه است که هر عضو آن، ۴ بیت است (شکل بالا). حاصل جمع v به با عدد ۲، ما را به اشاره‌گری به ابتدای عنصر ۲ می‌رساند.



۲. می‌توان دو اشاره‌گر را از یک دیگر کم کرد. در این صورت، تعداد عناصر موجود بین آن دو آدرس بدست می‌آید.

احطرار : این کار در صورتی معنا دارد که هر دو اشاره‌گر، مربوط به یک آرایه‌ی واحد باشند.

درس ۸ : کاراکترها و رشته‌ها

در پایان این بخش، فرآگیران باید با مطالب زیر آشنا شده باشند:

- با کاراکترها در کامپیوتر و زبان C آشنا شود.
- با توابع مختلف کار با کاراکترها (خصوصا کتابخانه ctype) آشنا شود.
- با کلیات رشته‌ها آشنا شود.
- توابع اساسی کار با رشته‌ها را بشناسد (خصوصا کتابخانه string).

۱-۸. مقدمه‌ای بر رشته‌ها و کاراکترها

اساسی‌ترین نوع داده، برای کار با متن‌ها و برنامه‌هایی که با متن سر و کار دارند، کاراکترها هستند. کاراکترها به شکل اعدادی صحیح در حافظه ذخیره می‌شوند و از طریق جدولی مثل جدول ASCII بازیابی می‌شود. نحوه تعریف رشته‌ها به صورت زیر است:

char str[]= “salam;”

این بدان معناست که str یک رشته و در اصل، آرایه‌ای از کاراکترها است. این آرایه به شکل زیر خواهد بود:

char str[] = {‘s’ ,‘a’ ,‘l’ ,‘a’ ,‘m’ ,‘\0’};

نکته: رشته‌ها در بین دو علامت ظاهر می‌شوند، اما کاراکترها در بین دو علامت ظاهر می‌شوند.

مثال ۷۰: رشته‌ای را به طول ۵ تعریف کنید و از کاربر، مقدار آن را دریافت کرده و نمایش دهید.

۲-۸. کتابخانه‌هایی برای کار با رشته‌ها و کاراکترها

با توجه به اینکه رشته‌ها و کاراکترها، در زبان C کاربردهای زیادی دارند،

کتابخانه‌هایی برای کار کردن راحت‌تر با آن‌ها ایجاد شده است. در این راستا، با سه کتابخانه

آشنا می‌شویم:

ctype.h •

stdio.h و stdlib.h •

string.h •

۲-۸. پردازش ورودی و خروجی با استفاده از کتاب خانه stdio

کتابخانه‌ی stdio امکاناتی را برای خواندن و نوشتan رشته‌ها فراهم می‌کند. این

کتابخانه، امکانات و توابع زیادی را دارد، اما در این بخش، به مرور مهم‌ترین توابع این

کتابخانه می‌پردازیم.

برخی از توابع مهم این کتابخانه در جدول زیر آمده است.

عملکرد	تابع
	getchar
	putchar
	gets
	puts

مثال ۷۱: برنامه‌ای را به زبان C بنویسید که با استفاده از تابع getchar یک خط متن رو

دریافت کرده و در یک رشته ذخیره کند.



۲-۲-۸. پردازش کاراکترها با کتابخانه ctype

توابع این کتابخانه، صرفاً برای کار با کاراکترها هستند. ورودی این توابع، یا کاراکترها هستند یا EOF (معمولًاً معادل با ۱- است). در ادامه با مهم‌ترین توابع آن آشنا می‌شویم:

کارکرد	نام تابع
	isdigit
	isalpha
	isxdigit
	islower
	isupper
	isspace
	iscntrl
	ispunct
	isprint
	isgraph

نکته (if یک خطی) : می‌توان شرط‌ها را به صورت زیر نوشت:

<condition> ? <action if condition is true> : <action if condition is false>;

۲-۳-۸. کتابخانه String

در کتابخانه string ، توابعی کاربردی و مفید به منظور کار با رشته‌ها در نظر گرفته شده است. عمده‌ی کاربرد این تابع در موارد زیر است:

۳. اندازه‌گیری طول رشته‌ها

۴. مقایسه‌ی رشته‌ها

۵. چسباندن رشته‌ها به یک دیگر

۶. جستجو در رشته‌ها

۷. شکستن رشته‌ها با نشانه‌گذاری

۱-۳-۲-۸. تابع `strlen`

طول یک رشته را می‌توان با استفاده از تابع `strlen` بدست آورد. این تابع، آدرس یک رشته را دریافت کرده و طول آن را (تعداد تمامی کاراکترهای آن را – بجز کاراکتر تهی) بدست می‌آورد.

مثال ۷۲: برنامه‌ای را به زبان C بنویسید که یک رشته را دریافت کرده و آن را معکوس نماید.

۲-۳-۲-۸. کپی یک رشته در رشته‌ی دیگر

از فصول قبل می‌دانیم که در صورتی که بخواهیم یک متغیر را در متغیر دیگری کپی کنیم، از علامت تساوی (تخصیص) استفاده می‌کنیم.

مثال ۷۳: آیا بنظر شما می‌توانیم یک رشته را با علامت تخصیص در رشته‌ی دیگر ذخیره کنیم؟ اگر پاسخ منفی است دلیل آن را توضیح دهید.

۳-۳-۲-۸. توابع `strncpy` و `strcpy`

برای کپی کردن یک رشته در رشته‌ی دیگر، لازم است از تابع زیر استفاده شود:

۱. تابع `strcpy` در صورتی که بخواهید تمام یک رشته را در رشته‌ی دیگر کپی کنید.

```
strcpy(char* string2, char* string1);
```



۲. تابع `strncpy` در صورتی که بخواهد تعداد مشخصی از کاراکترهای یک

رشته را در رشته‌ی دیگر ذخیره کنید.

```
Strncpy(char* string2, char* string1, int n);
```

توجه: دقت کنید که رشته‌ی مقصد به اندازه‌ی کافی جا داشته باشد.

توجه: تابع `strncpy` در صورتی که مجموع کاراکترهای **غیرنهی** رشته‌ی مبدا و

مقصد به صورت اکید از آرگومان سوم کوچکتر نباشد، در این صورت کاراکتر تهی به

انتهای رشته‌ی مقصد اضافه نمی‌شود. در این صورت حتماً کاراکتر تهی را در انتهای

رشته قرار دهید.

درس ۹ : ساختارها

در پایان این بخش، فرآگیران باید با مطالب زیر آشنا شده باشند:

- با کاراکترها در کامپیوتر و زبان C آشنا شود.
- با توابع مختلف کار با کاراکترها (خصوصا کتابخانه ctype) آشنا شود.
- با کلیات رشته‌ها آشنا شود.
- توابع اساسی کار با رشته‌ها را بشناسد (خصوصا کتابخانه string).

۱-۹. ساختار چیست؟

در بسیاری از برنامه‌ها، گروهی از داده‌ها در ارتباط با یک دیگر هستند؛ به گونه‌ای که مجموعه‌ی آن‌ها، یک واحد اطلاعاتی جامع تری را تشکیل می‌دهد. به عنوان مثال، اطلاعات یک دانشجو، یک واحد اطلاعاتی مرکب است که شامل داده‌هایی نظیر موارد زیر است:

- نام
- نام خانوادگی
- شماره دانشجویی
- معدل

معمولًاً به مجموعه‌ای از اطلاعاتی که یک واحد اطلاعاتی کامل تری را تشکیل می‌دهند، یک رکورد گفته می‌شود و هر کدام از اطلاعات بالا، یک فیلد نامیده می‌شوند. خصوصیت رکورد آن است که نوع هر یک از فیلدهایش می‌تواند با فیلدهای دیگر، متفاوت باشد.

در زبان C برای ذخیره و پردازش یک رکورد، از نوع داده‌ای به نام ساختار استفاده می‌شود. ساختار یک نوع داده‌ی مرکب است که مجموعه‌ای از چند نوع داده می‌باشد. در حقیقت عضوهای ساختار همان **فیلد**‌های رکورد هستند.

ساختارها (Structs)، مجموعه‌ای از متغیرهای مرتبط با یک دیگر و تحت یک نام هستند. شاید این ویژگی، تا حدودی شما را به یاد تعریف آرایه‌ها بیندازد؛ اما ساختارها یک فرق اساسی با آرایه‌ها دارند: «در آرایه‌ها، تمامی عناصر بایستی از یک نوع داده باشند، این درحالی است که در ساختارها، عناصر می‌توانند از یک نوع داده نباشند.

۲-۹. کاربرد ساختارها

ساختارها به منظور ایجاد یک رکورد به منظور ذخیره‌سازی می‌تواند مورد استفاده قرار گیرد. همچنین یکی از ابزارهای اساسی در پیاده‌سازی داده‌ساختارهایی نظیر لیست پیوندی می‌باشد.

فراموش نکنید که حتماً
بایستی پس از بستن آکلاد،
علامت سمی کالن را قرار
دهید.

۳-۹. تعریف یک ساختار

یک ساختار به شرح زیر تعریف می‌شود:

```
struct [structure tag] {
    member definition;
    member definition;
    ...
    member definition;
} [one or more structure variables];
```

۱. کلیدواژه‌ی struct

۲. برچسب ساختار (Struct Tag)

۳. اعضای ساختار (struct members)

نکته: هر کدام از اعضای ساختار باید دارای نامی واحد باشد، اما تعریف نامهای یکسان در ساختارهای متفاوت بلامانع است. در ادامه، نمونه‌ای از یک ساختار را مشاهده می‌کنید.

```
struct Person {
    char firstName[30];
    char lastName[30];
    int age;
    double salary;
};
```

۴-۹. مقداردهی و به کارگیری ساختارها

به منظور به کارگیری ساختارها، لازم است در ابتدا مقادیر آن‌ها را در قالب یک متغیر ذخیره کنیم. برای این منظور، مطابق با الگوی زیر عمل می‌کنیم:

```
struct STRUCT_TAG VARIABLE_NAME;
```

مقداردهی به یک ساختار به دو صورت قبل انجام است:

۱. نوشتن مقادیر در بین {} .
۲. تعریف ساختار و مقداردهی آن در بخش دیگری از برنامه. برای دسترسی به فیلدهای یک ساختار از علامت نقطه پس از نام آن ساختار استفاده می‌کنیم.

مثال ۷۴: ساختار Person را با هر دو روش مقداردهی کرده و مقادیر آن را چاپ کنید.

```
struct Person ali = {"Ali", "Alavi", 20, 19.2};

printf("%s\n", ali.firstName);
printf("%s\n", ali.lastName);
printf("%d\n", ali.age);
printf("%lf\n", ali.salary);

struct Person student;
scanf("%s", student.firstName);
scanf("%s", student.lastName);
scanf("%d", &student.age);
scanf("%lf", &student.salary);
```

نکته: همانند تمامی متغیرها (بر خلاف آرایه‌ها) می‌توانید ساختارها را مثل یک

متغیر معمولی در متغیر دیگری و با عمل انتساب ($=$) اختصاص دهدید (مشروط بر آن که هردو از یک نوع باشند).

اخطار : امکان مقایسهٔ دو ساختار با دو عملگر $=$ و $!=$ وجود ندارد (بنظر شما چرا؟)

۵-۹. ارسال ساختار به تابع

به صورت پیش فرض (بر خلاف آرایه‌ها) ارسال ساختارها به آرایه‌ها به صورت ارسال با مقدار (Pass by Value) است. اما می‌توان همانند آرایه‌ها، کاری کرد که ارسال ساختارها به صورت ارسال با ارجاع (Pass by Reference) صورت گیرد.

مثال ۷۵: تابعی را بسازید که یک نمونه ساختار از نوع Person را دریافت کرده و آن را چاپ کند.

مثال ۷۶: تابعی را بسازید که یک نمونه از ساختار Person را به همراه یک رشته دریافت کرده و رشته‌ی دریافت شده را جایگزین نام آن نمونه بکند.

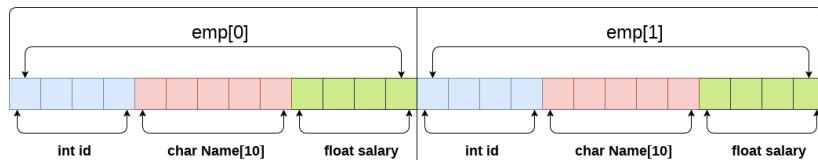
نتیجه: در صورتی که از روش ارسال با مقدار استفاده شود، مقدار ساختار پاس داده شده تغییر نمی‌کند و اگر بخواهیم تغییر کند

۶-۹. آرایه‌ای از ساختارها

فرض کنید بخواهیم آرایه‌ای از ساختارها ایجاد کنیم. در این صورت، کافی است که همانند تمامی متغیرهای عادی اقدام به ایجاد یک آرایه‌ای از متغیرهایی از نوع آن ساختار

```
struct STRUCT_TAG ARRAY_NAME [size];
```

در شکل بعد، نحوه ذخیره‌سازی اعضای یک آرایه‌ای از ساختارهای مقداردهی شده و نحوه ذخیره‌سازی آن‌ها در حافظه آورده شده است.



```
struct employee
{
    int id;
    char name[5];
    float salary;
};

struct employee emp[2];
```

مثال ۷۷: آرایه‌ای ۵ عضوی از نوع ساختار Person ایجاد کرده و مقادیر مشخصات ۵ شخص را از کاربر دریافت کنید و سپس در آرایه، ذخیره سازی کنید.

۷-۹. تعریف نوع داده جدید با دستور **typedef**

با این دستور، می‌توانید انواع داده‌ها (Type Data) را به صورت دلخواه، تغییر نام دهید. به عنوان مثال، اگر تمایل دارید که برای تعریف یک متغیر از نوع عدد صحیح، از نوع داده‌ی دلخواه Integer به جای int استفاده کنید، خواهید داشت:

```
typeof int Integer;
```

دستور قبل، به کامپایلر می‌گوید که هرجا Integer را دید، به جای آن int بگذارد.

برای راحتی در استفاده از ساختارها، می‌توانید به یکی از دو طریق زیر، از دستور **typedef** استفاده نمایید:

حالت ۱ :

```
typedef struct employee Employee;
```

حالت ۲ :

```
typedef struct{  
    // definition  
} Student;
```

درس ۱۰ : ذخیره و بازیابی در فایل‌ها

در پایان این بخش، فرآگیران باید با مطالب زیر آشنا شده باشند:

- با انواع فایل‌ها آشنا شود.
- ضرورت استفاده از فایل‌ها را درک کند.
- بتواند یک فایل متنی را ایجاد کرده و باز کند.
- بتواند در یک فایل متنی بنویسد و از آن بخواند.

۱-۱۰. فایل چیست و چه ضرورتی دارد که از آن استفاده کنیم؟

برنامه‌هایی را که تاکنون نوشته‌اید را به خاطر بیاورید. تمام این برنامه‌ها، با پایان یافتن اجرای برنامه، به حالت قبل بر می‌گردند و تمامی اطلاعاتی که در آن درست شده است، پاک می‌شود. همان طور که می‌دانیم، انجام چنین کاری با داده‌های خیلی زیاد، منجر به از بین رفتن آن می‌شود.

از سوی دیگر، می‌دانیم که در حین اجرای برنامه، اطلاعات آن در حافظهی بارگزاری می‌شود و ظرفیت این حافظه محدود است و با خاموش شدن کامپیوتر، این اطلاعات نیز از بین می‌رود؛ در نتیجه لازم خواهد بود که اطلاعات مورد نیاز خود را در حافظهی ذخیره کنیم.

فایل‌ها به ما امکان ذخیره‌ی دائمی اطلاعات و بازیابی آن‌ها را در صورت لزوم، می‌دهد.

۲-۱۰. انواع فایل‌ها

به طور کلی، دو نوع فایل داریم:

۱. فایل‌های متنی

۲. فایل‌های باینری (دودویی)

و در حالت کلی، فایل چیزی نیست به جز یک دنباله‌ای از بایت‌ها که با نشان EOF

پایان می‌یابد.

۳-۱۰. بازکردن فایل متنی و اشاره به آن

زمانی که یک فایل را باز می‌کنید، اطلاعات آن در بخشی از حافظه‌ی اصلی بارگزاری می‌شود و اشاره‌گری به ابتدای آن به وجود می‌آید.

```
FILE * filePtr;
```

باز کردن فایل متنی با دستور fopen صورت می‌گیرد. این تابع، آدرس فایل و مود آن را به عنوان ورودی دریافت کرده و در صورتی که بتواند فایل را به صورت موفقیت آمیز باز کند، اشاره‌گر فایل را مقداردهی کرده و در غیر این صورت، مقدار NULL را بر می‌گرداند.

```
filePtr = fopen(FILE_PATH , FILE_MODE);
```

جدول ۷. مودهای کار با فایل و محل اشاره‌گر متناظر با آن

کار کرد	محل قرارگیری اشاره‌گر	مود فایل
بازکردن فایل و خواندن از آن	ابتدای فایل	r
نوشتن در فایل (اگر فایل موجود باشد، از میان رفته و مجدداً ایجاد می‌شود)	ابتدای فایل	w
نوشتن و خواندن	ابتدای فایل	r+
افزودن به انتهای فایل	انتهای فایل	a

توجه : چنانچه فایلی را برای خواندن باز کنید در حالی که موجود نباشد، اشاره‌گر فایل NULL می‌شود. بنابراین بهتر است پیش از کار با فایل، از NULL نبودن آن اطمینان حاصل کنید.

پس از کار با فایل، لازم است فایل بسته شود. در صورتی که فایل بسته نشود،



آخرین تغییرات بر روی آن ذخیره نمی‌شود. اگرچه با قطع برنامه به صورت خودکار فایل بسته می‌شود اما بهتر است خودتان این کار را انجام دهید.

۴-۱۰. نوشتن در فایل متنی

برای این منظور، می‌توان از توابع مختلفی استفاده کرد که بسیار مشابه با توابعی هستند که به وسیله‌ی آن‌ها، رشته‌هایی را می‌نوشتمیم و یا می‌خواندیم.

تابع	شکل استفاده	کارکرد
fprintf	fprintf(filePtr, controlStr, ...args);	
fputc	fputc(char, filePtr);	
fputs	fputs(string, filePtr);	

۵-۱۰. خواندن از فایل متنی

توابعی که به منظور خواندن فایل متنی به کار می‌روند نیز بسیار مشابه با توابعی هستند که به منظور خواندن رشته‌ها مورد استفاده قرار می‌گرفتند.

تابع	شکل استفاده	کارکرد
fscanf	fscanf(filePtr, controlStr, ...args);	
fgetc	fgetc(char, filePtr);	
fgets	fgets(string, numOfChars, filePtr);	

۶-۱۰. دیگر توابع کاربردی کار با فایل‌ها

توابع دیگری نیز به منظور کار با فایل می‌توانند مورد استفاده قرار گیرند :

۱. تابع **feof** : اشاره‌گر فایل را دریافت کرده و تعیین می‌کند که آیا به انتهای فایل

رسیدید یا خیر؟

۲. تابع `fseek` : اشاره‌گر فایل را دریافت کرده و شما را به ابتدای فایل برمی‌گرداند.

مراجع

۱. چگونه به زبان C برنامه بنویسم؟ اثر دایتل و دایتل

۲. ویراست هشتم (ترجمه علیرضا زارعپور) از انتشارات نص – (۲۰۱۶)

3. C : How to Program? , Ditel & Dited 9th Edition

۴. برنامه نویسی به زبان C ، هایده علی آبادی، انتشارات گسترش علوم پایه

۵. فیلم آموزشی اصول و مبانی برنامه نویسی، با تدریس دکتر کلامی هریس از موسسه فرادرس :

<https://faradars.org/courses/fvrprg101-programming-basics-concepts>

۶. آموزش تبدیل فلوچارت به کد با فلوگوریم، با تدریس مهندس وحید باقی از موسسه فرادرس :

<https://faradars.org/courses/fvsft96041-convert-flowchart-to-code-using-flowgorithm>

۷. آموزش برنامه نویسی به زبان C، با تدریس دکتر سید مصطفی کلامی هریس از موسسه فرادرس :

<https://faradars.org/courses/fvrc101-c-programming>

۸. آموزش مبانی برنامه نویسی با رویکر حل مسئله، با تدریس منوچهر بابایی از انتشارات فرادرس :

<https://faradars.org/courses/fvprg9412-programming-algorithm-and-flowchart>

9. C Tutuorial, Tutorial Points :

<https://www.tutorialspoint.com/cprogramming/index.htm>

10. Tutorial Points : Computer Programming Tutorial :

https://www.tutorialspoint.com/computer_programming/index.htm

۱۱. کتاب درسی دانش فنی پایه (ویژه‌ی هنر آموزان رشته‌ی کامپیوتر – پایه‌ی دهم فنی و حرفه‌ای) :

<http://chap.sch.ir/books/7457>

۱۲. حل تمرین‌های C به همراه فلوچارت، غلامرضا رحیمی (افشین)، چاپ هفتم ویراست سوم، انتشارات

الماضی دانش : ۱۳۹۸