مبانی کامپیوتر و برنامه سازی

محمدهادی علائیان

۱۵ آذر ۱۴۰۰

فهرست مطالب

فهرست تصاویر

فهرست جداول

فصل ۱

مقدمه

کامپیوتر: ابزاری برای محاسبات و تصمیم گیری منطقی، میلیون ها بار سریع تر از انسان.
برای حل مسئله :

- گام اول، تجزیه و تحلیل و شناخت مسئله.
- گام دوم، طراحی الگوریتم ⟵ یک دنباله منطقی از دستورات.
- گام آخر، تفهیم الگوریتم به کامپیوتر ⟵ با زبان های برنامه نویسی

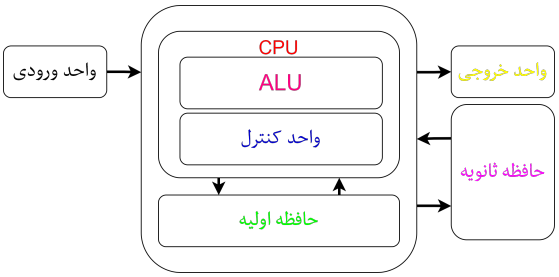
کامپیوتر از دو بخش سخت افزار و نرم افزار تشکیل شده است.

- سخت افزار: کلیه دستگاههای الکتریکی، الکترونیکی و مکانیکی تشکیل دهنده یک کامپیوتر را سخت افزار آن می گوییم. مجموعه ای از ابزار ها مثل موس و ...
- نرم افزار: مجموعه برنامه های کامپیوتری که توسط انسان برای یک کاربرد خاص نوشته شده اند و بدون آنها سخت افزار قادر به کاری نیست. نرم افزار کامپیوتر به دو دسته اصلی تقسیم می گردد :
  - نرم افزارهای کاربردی : نرم افزارهایی هستند که برای یک کاربرد خاص و رفع یک نیاز مشخص کاربران نوشته شده اند. مانند سیستمهای حسابداری، دبیرخانه، سیستم انتخاب واحد دانشگاهی، انواع بازیها
  - نرم افزارهای سیستمی : نرم افزارهایی هستند که برای ایجاد و یا اجرای برنامه های کاربردی نوشته می شوند. مهمترین برنامه سیستمی، سیستم عامل است. سیستم عامل نرم افزاری است که ارتباط بین سخت افزار و کاربران (یا برنامه های کاربردی کاربران) را فراهم می سازد. در حقیقت سیستم عامل مدیریت منابع سخت افزاری یک کامپیوتر را بعهده دارد.

اجزای اصلی کامپیوتر

- واحد های ورودی (input unit): ابزاری برای ورود اطلاعات.
- واحد های خروجی (output unit): ابزاری برای خروج اطلاعات.
- واحد حافظه اولیه (primary memory): محلی برای نگهداری و ذخیره اطلاعات و داده ها. (خصوصیات حافظه اصلی ⟵ ۱-حجم نه چندان زیاد، ۲-سرعت بالا، ۳- عدم ماندگاری اطلاعات چون هرگاه کامپیوتر را روشن می کنیم حافظه اصلی (RAM) خالی است.) یکی از حافظه های اولیه RAM نام دارد که مخفف شده از Random Access Memory است. دلیل نام گزاری Random برای این است که می توان به صورت تصافی هر یک از خانه های حافظه را درخواست داد و حافظه پاسخگو خواهد بود.
- واحد محاسبات منطقی (ALU): انجام عملیات ریاضی (مانند ضرب، جمع، تفریق و تقسیم) و عملیات منطقی (مانند مقایسه بین اعداد) را بر عهده دارد.

- واحد پردازش مرکزی (CPU): هماهنگی و نظارت بر کار سایر اجزای کامپیوتر را بر عهده دارد. به عنوان مثال واحد های ورودی را کنترل کند تا داده ورودی را دریافت کند و یا داده های اماده شده برای واحد های خروجی را در اختیار واحد خروجی قرار دهد و همچنین به ALU دستور دهد که چه زمانی بر روی چه داده هایی چه نوع پردازشی انجام دهد.



شکل ۱.۱: سخت افزار و معماری بلوکی از اجزای داخلی یک کامپیوتر

- واحد حافظه ثانویه (secondary storage): محلی برای نگه داری داده هایی که در حال حاضر مورد استفاده قرار نمیگیرند. دسترسی کند و قیسمت ارزان از جمله خصوصیات این ابزار است. ما آنها را عموما با نام هارد دیسک یا Hard-disk می شناسیم.

۱.۱ سیستم عامل چیست؟

برنامه ای است که بین سخت افزار و نرم افزار ارتباط برقرار می کند. برنامه ای که تقسیم منابع (حافظه و CPU) را انجام دهد.

windows , unix , linux , ms-Dos , solaris , macos .

۲.۱ نحوه پردازش اطلاعات در سیستم عامل:

۱. Batch processing ⟵ عدم تعامل با کاربر و از مد افتاده. فقط یک کار انجام می دهد.

۲. parallel processing ⟵ پردازش موازی. چند تا CPU به طور هم زمان محاسبات را انجام می دهند.

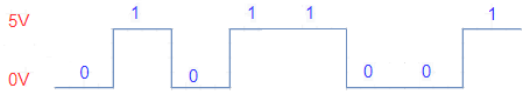
۳. Time sharing ⟵ به اشتراک گذاشتن زمان. کاری که کامپیوترهای خانگی انجام می دهند.

۳.۱ نحوه محاسبات سیستم عامل

- متمرکز ⟵ کامپیوتر های خانگی.
- توزیع شده ⟵ client, p2p, server

۴.۱ اینترنت و وب

شبکه ای از کامپیوتر های بهم متصل و گسترده ترین وسیله ارتباط در دنیا. از خصوصیات آن دسترسی خیلی سریع به اطلاعات، تغییردر روش های تجارت.



شکل ۲.۱: نمونه ای از پالس های الکتریکی.

۵.۱ نمایش داده ها

داده ها در کامپیوتر به صورت پالس های الکتریکی هستند. نمونه ای از این پالس ها را در شکل ۲.۱ مشاهده می شود.
داده در کامپیوتر ترکیبی از صفر و یک است که به آن نمایش دودویی (Binory) نیز گفته می شود. کوچک ترین واحد نمایش داده Bit بیت (Bit) نام دارد که یک مقدار ۰ یا ۱ دارد.
هر هشت بیت را یک بایت (Byte) نام دارد.

$$1\ \text{kb} \longrightarrow 1024 = 1 \times \ 2^{10}\ \text{Byte}$$
$$1\ \text{mb} \longrightarrow 1 \times \ 2^{20}\ \text{Byte}$$

۱.۵.۱ نمایش داده ها و تبدیل مبنا

برای نمایش اطلاعات در کامپیوتر از مبنای ۲ استفاده می گردد در حالی که مبنای ۱۰، مبنای مورد استفاده انسانها در ریاضیات است. در ریاضیات متداول هر عدد N بصورت زیر تفسیر می گردد:

$$N = (a_{n-1}a_{n-2}\cdots a_2a_1a_0)_{10} = a_0 \times 10^0 + a_1 \times 10^1 + a_2 \times 10^2 + \cdots a_{n-1} \times 10^{n-1}$$

مثال: عدد ۳۴۸۲ بصورت زیر تفسیر می گردد:

$$(3482)_{10} = 2 \times 10^0 + 8 \times 10^1 + 4 \times 10^2 + 3 \times 10^3.$$

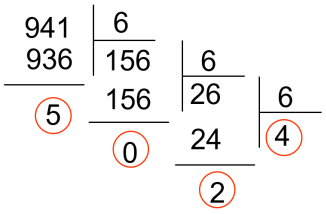
نکته: در سیستم دهدهی نیاز به ۱۰ رقم (از ۰ تا ۹) داریم.
می توان اعداد را در هر مبنای دلخواه دیگری مانند b نیز نشان داد در اینصورت هر عدد مانند N در مبنای b بصورت زیر تفسیر می گردد:

$$N = (a_{n-1}a_{n-2}\cdots a_2a_1a_0)_b = a_0 \times b^0 + a_1 \times b^1 + a_2 \times b^2 + \cdots a_{n-1} \times b^{n-1}$$

کاملا واضح است که در مبنای b نیاز به b رقم (از ۰ تا b-۱) خواهیم داشت. به عنوان مثال یک عدد در مبنای ۶ از ارقام ۰ تا ۵ تشکیل می گردد، بنابراین $(341)_6$ یک عدد درست است اما $(592)_6$ غیر قابل قبول می باشد.

۲.۵.۱ تبدیل مبناها

برای تبدیل یک عدد از مبنای ۱۰ به هر مبنای دلخواه b، از روش تقسیمات متوالی استفاده می گردد.
مثال: $(?)_6 = (941)_{10}$



شکل ۳.۱: $(941)_{10} = (4205)_6$.

برای تبدیل از مبنای b به مبنای ۱۰ کافی است ارقام عدد مورد نظر را در ارزش مکانی آنها ضرب و سپس با یکدیگر جمع کنیم.

مثال: $(?)_{10} = (4205)_6$

$$(4205)_6 = 5 \times 6^0 + 0 \times 6^1 + 2 \times 6^2 + 4 \times 6^3 = 5 + 0 + 72 + 864 = (941)_{10}$$

۳.۵.۱ مبنای ۲ و اهمیت آن

مبنای ۲ اهمیت بسیار زیادی در کامپیوترهای دیجیتال دارد. چراکه:

- در مبنای ۲ تنها به ۲ رقم نیاز داریم، یعنی ۰ و ۱
- این دو رقم را می توان توسط هر ابزاری که دارای دو حالت باشد نشان داد. مثلا یک لامپ که خاموش بودن لامپ به معنای ۰ و روشن بودن آن به معنای ۱ می باشد.
- این همان ایده ای است که کامپیوترهای دیجیتال از آن استفاده می کنند.
- همانطور که قبلا نیز گفته شد واحد نگهداری اطلاعات در کامپیوتر بیت می باشد که هر بیت قادر به نگهداری ۰ و یا ۱ است. با کنار هم قرار دادن بیتها، بایتها تشکیل می گردند و بدینوسیله اطلاعات مورد نظر در قالب بایتها تشکیل می گردند.

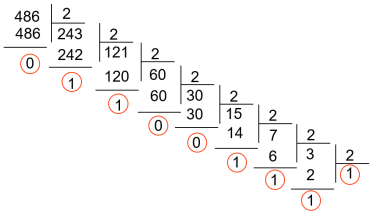
تبدیل اعداد از مبنای ۱۰ به ۲ و بالعکس بسیار ساده و همانند سایر مبنا ها است.

مثال: $(?)_{10} = (11001001)_2$

$$(11001001)_2 = 1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 1 \times 2^6 + 1 \times 2^7$$

$$= 1 + 0 + 0 + 8 + 0 + 0 + 64 + 128 = (201)_{10}$$

مثال: $(?)_2 = (486)_{10}$



شکل ۱.۴: $(486)_{10} = (111100110)_2$.

مبناهای ۸ و ۱۶ و کاربرد آنها

مشکل اصلی در مبنای ۲ اندازه بزرگ اعداد است. بعنوان مثال عدد ۴۸۶ که در مبنای ۱۰ تنها ۳ رقم دارد، تبدیل به یک عدد ۹ رقمی در مبنای ۲ شده است. این مسئله باعث می شود که محاسبه در مبنای ۲ برای انسانها بسیار مشکل شود و معمولا برنامه نویسان علاقه چندانی به مبنای ۲ ندارند. مبنای ۸ نیز همانند سایر مبناها می تواند مورد استفاده قرار گیرد و در ظاهر تفاوتی با سایر مبناها ندارد. اما ویژگی جالب این مبنا در تبدیل ساده آن به مبنای ۲ و بالعکس است. البته برای همواره میتوان برای تبدیل مبنا از ۲ به ۸، عدد را از مبنای ۲ به مبنای ۱۰ و بعد از مبنای ۱۰ به مبنای ۸ منتقل کرد. ولی راه آسانتری برای انتقال سریع تر وجود دارد.

همانطور که می دانیم در مبنای ۸ تنها ارقام ۰ تا ۷ استفاده می شوند. از طرف دیگر اگر یک عدد در مبنای ۲ با حداکثر ۳ رقم را درنظر بگیریم، در می یابیم که می توان

$$000 = 0, 001 = 1, 010 = 2, 011 = 3, 100 = 4, 101 = 5, 110 = 6, 111 = 7$$

را با آن نشان داد. بنابراین می توان نتیجه گرفت که هر ۳ رقم در مبنای ۲، برابر است با ۱ رقم در مبنای ۸ و بالعکس. این نتیجه گیری تبدیل این دو مبنا را به یکدیگر ساده می کند.

مثال: $(?)_8 = (10101110)_2$

$$10_2 = 2, 101_2 = 5, 110_2 = 6 \longrightarrow (10101110)_2 = (256)_8.$$

مثال: $(?)_2 = (271)_8$

$$2 = 10_2, 7 = 111_2, 1 = 001_2 \longrightarrow (271)_8 = (10111001)_2$$

اکثر برنامه نویسان کامپیوتر ترجیح می دهند از مبنای دیگری بنام مبنای ۱۶ استفاده نمایند. این مبنا نیز همانند مبنای ۸ بسادگی قابل تبدیل به مبنای ۲ است، اما اعداد آن به ارقام کمتری نیاز دارند. در این مبنا نیاز به ۱۶ رقم داریم درحالیکه ارقام موجود فقط ۱۰ تا است. بهمین دلیل از حروف A تا F برای ارقام ۱۰ تا ۱۵ استفاده می گردد.یعنی ارقام عبارتند از:

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$$

تبدیل اعداد از مبنای ۲ به ۱۶ و بالعکس ازهمان روش گفته شده برای مبنای ۸ استفاده می نماییم با این تفاوت که هر رقم در مبنای ۱۶ معادل ۴ رقم در مبنای ۲ است.

مثال: $(?)_{16} = (11010111001)_2$

$$(110)_2 = 6_{16}, (1011)_2 = B_{16}, (1001)_2 = 9_{16} \longrightarrow (11010111001)_2 = (6B9)_{16}$$

مثال: $(?)_2 = (A3E)_{16}$

$$A_{16} = (1010)_2, 3_{16} = (0011)_2, E_{16} = (1110)_2 \longrightarrow (A3E)_{16} = (101000111110)_2$$

نمایش اعداد صحیح

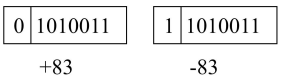
اعداد صحیح در کامپیوتر با استفاده از مبنای ۲ نمایش داده می شوند. برای نمایش اعداد صحیح از ۱ یا ۲ بایت و یا بیشتر (بسته به اندازه عدد) استفاده می گردد. چنانچه قصد ذخیره اعداد صحیح مثبت را داشته باشیم، با استفاده از ۱ بایت می توان اعداد ۰ تا ۲۵۵ را ذخیره کرد. بنابراین برای ۱ بایت، بزرگترین عدد قابل ذخیره برابر است با $2^8 - 1 = 255$.

با استدلال مشابهی چنانچه از ۲ بایت یا ۱۶ بیت استفاده گردد، بزرگترین عدد قابل ذخیره برابر $2^{16} - 1 = 65535$ خواهد بود.

اما مشکل این است که اعداد منفی را چگونه ذخیره نماییم؟ برای این کار چندین روش وجود دارد که هریک را جداگانه بررسی می نماییم.

- استفاده از بیت علامت:

در این روش سمت چپ ترین بیت برای علامت عدد درنظر گرفته می شود و سایر بیتها مقدار عدد رانشان می دهند. ۰ بودن بیت علامت بمعنای مثبت بودن و ۱ بودن آن به معنای منفی بودن عدد می باشد. با داشتن ۸ بیت می توان اعداد بین ۱۲۷- تا ۱۲۷+ را نمایش داد. این روش دو مشکل اصلی دارد. ۱. دو مقدار متفاوت

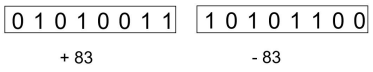


شکل ۵.۱: مثالی از نوع ذخیره سازی در اعداد صحیح در مبنای ۲ به روش استفاده از بیت علامت.

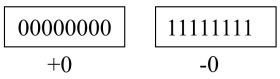
+۰ و -۰ وجود دارد. ۲. برای عمل جمع و تفریق نیاز به دو مدار جداگانه داریم.

- استفاده از متمم ۱:

در این روش اعداد مثبت بصورت معمولی نمایش داده می شوند. اما برای نمایش اعداد منفی، ابتدا قدر مطلق آن را (بصورت عدد مثبت) نمایش داده و سپس کلیه ۰ها را به ۱ و بالعکس تبدیل می نماییم. با توجه به



شکل ۶.۱: مثالی از نوع ذخیره سازی در اعداد صحیح در مبنای ۲ به روش متمم ۱.

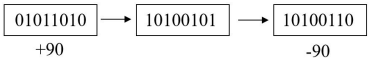


شکل ۷.۱: مقدارهای ۰+ و ۰- در نمایش اعداد صحیح به مبنای ۲ به روش متمم ۱.

محدودیت تعداد بیتها و جلوگیری از تداخل اعداد مثبت و منفی، برای اعداد مثبت باید بیت سمت چپ ۰ باشد و به همین دلیل بازه اعداد مجاز از ۱۲۷- تا ۱۲۷+ است. خوشبختانه اکنون تنها نیاز به یک مدار برای عمل جمع و تفریق است اما هنوز هم دو نمایش مختلف برای ۰+ و ۰- وجود دارد.

- استفاده از متمم ۲:

در این روش برای نمایش اعداد منفی ابتدا متمم ۱ آنها را محاسبه و سپس آن را با ۱ جمع می کنیم. بازه اعداد



شکل ۸.۱: مثالی از نوع ذخیره سازی در اعداد صحیح در مبنای ۲ به روش متمم ۲.

مجاز در این روش از ۱۲۸- تا ۱۲۷+ است.

این روش هردو مشکل روش های قبل را حل می کند. چرا که تنها یک نمایش برای ۰+ و ۰- وجود دارد. برای عمل تفریق می توان از همان مدار جمع استفاده کرد. بدین ترتیب که ابتدا عدد دوم را به روش متمم ۲ منفی کرده و با عدد اول جمع می کنیم.

مثال:

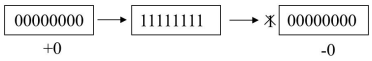
$$53 - 22 = 53 + (-22) = 31$$
$$(00110101) - (00010110) = (00110101) + (11101010) = 1(00011111)$$
$$38 - 60 = 38 + (-60) = -22$$
$$(00100110) - (00111100) = (00100110) + (11000100) = (11101010)$$

۴.۵.۱ نمایش مقادیر اعشاری

نمایش اعداد اعشاری در کامپیوتر مشکلتر است. ما معمولا اعداد اعشاری را به شکل ممیز ثابت نشان می دهیم مانند: ۵۳.۶۴۸ که پنجاه و سه و ممیز ششصد و چهل و هشت خوانده می شود.
اما شکل دیگری نیز وجود دارد که به آن نماد علمی یا ممیز شناور گفته می شود و به شکل زیر است:

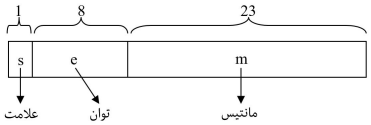
- 53.648×10^0
- 5.3648×10^1
- 0.53648×10^2
- 5364.8×10^{-2}

همانطور که دیده می شود، مکان ممیز در این نمایش شناور است و می تواند در هر نقطه ای قرار گیرد و البته توان نیز باید متناسب با آن تنظیم گردد. نماد علمی نرمال به حالتی گفته می شود که قسمت صحیح فقط دارای یک رقم غیر صفر باشد.



شکل ۹.۱: نمایش یک مقدار برای صفر در مبنای دو با روش متمم ۲.

برای نمایش اعداد اعشاری در کامپیوتر استانداردهای مختلفی وجود دارد که همگی در اصول مشترکند و تنها تفاوتهایی در جزئیات دارند.
ما در اینجا از یک روش استاندارد متداول که توسط انجمن معتبر IEEE ارائه شده است، استفاده می کنیم. این استاندارد بنام IEEE Standard ۷۵۴ Floating Point Numbers شناخته شده است. در استاندارد مورد نظر هر عدد اعشاری در یک کلمه ۳۲ بیتی ذخیره می گردد. ساختار هر عدد بصورت زیر است:



شکل ۱۰.۱: ساختار نمایش اعداد مبنای دو اعشاری با ۳۲ بیت.

برای ذخیره یک عدد اعشاری باید ابتدا آن را به نماد علمی نرمال در مبنای ۲ تبدیل کنیم. سپس آن را به شکل زیر ذخیره می کنیم:

۱. علامت عدد را در s قرار می دهیم (مثبت = ۰ و منفی = ۱)

۲. توان را در قسمت e قرار می دهیم که یک عدد مثبت بدون علامت است (بین ۰ تا ۲۵۵).

نکته: از آنجا که ممکن است توان مثبت یا منفی باشد، ابتدا ۱۲۷ واحد به توان اضافه می کنیم و سپس آن را ذخیره می کنیم. بنابراین e=۱۳۱ باشد بدین معناست که توان برابر ۴ بوده است و e=۱۲۰ به معنای توان برابر ۷- است. البته توانهای e=۲۵۵ و e=۰ برای منظورهای خاصی در نظر گرفته شده اند که بعدا توضیح داده خواهند شد.

۳. قسمت پایه عدد را در مانتیس قرار می دهیم.

نکته: چون در نمایش نرمال قسمت صحیح تنها یک رقم غیرصفر دارد و در نمایش دودویی نیز تنها رقم غیر صفر، رقم ۱ می باشد؛ بنابراین تنها قسمت اعشاری در مانتیس ذخیره می گردد و قسمت صحیح بطور پیش فرض ۱ درنظر گرفته می شود. این باعث می شود که یک بیت در ذخیره سازی صرفه جویی گردد.

مثال: عدد ۴۸۶ را بصورت دودویی اعشاری ذخیره نمایید.
ابتدا آن را به مبنای ۲ تبدیل می کنیم.

$$(486)_{10} = (111100110)_2$$

اکنون داریم:

$$111100110 = 1.11100110 \times 2^8$$

$$s : 0, e : 8 + 127 = 135 = 10000111, m : 11100110$$

بنابراین جواب نهایی بصورت زیر خواهد شد:

$$0|10000111|11100110000000000000000$$

مثال: عدد ۱۲۱.۶۴۰۶۲۵- را بصورت دودویی اعشاری ذخیره نمایید.
ابتدا عدد ۱۲۱ را به مبنای ۲ می بریم:

$$(121)_{10} = (1111001)_2$$

و اما برای تبدیل قسمت اعشاری باید از روش ضربهای متوالی استفاده نماییم. بدین صورت که ابتدا آن را در ۲ ضرب کرده و قسمت صحیح حاصلضرب را ذخیره می کنیم. همین عمل را مجددا برروی قسمت اعشاری حاصلضرب انجام می دهیم و اینکار را تا صفر شدن قسمت اعشاری و یا پر شدن تعداد بیتهای کلمه موردنظر (۲۳ بیت) تکرار می کنیم. در پایان ارقام ذخیره شده را از اول به آخر به ترتیب پس از ممیز قرار می دهیم. داریم :

0.640625 × 2 = 1.28125	قسمت صحیح = 1
0.28125 × 2 = 0.5625	قسمت صحیح = 0
0.5625 × 2 = 1.125	قسمت صحیح = 1
0.125 × 2 = 0.25	قسمت صحیح = 0
0.25 × 2 = 0.5	قسمت صحیح = 0
0.5 × 2 = 1.00	قسمت صحیح = 1

شکل ۱۱.۱: تبدیل مبنای قسمت اعشاری از عدد ۰.۶۴۰۶۲۵ در مبنای ۱۰ به مبنای دو با ضرب های متوالی به ۲.

$$(0.640625)_{10} = (0.101001)_2$$

بنابراین عدد نهایی بصورت زیر حاصل خواهد شد:

$$(121.640625)_{10} = (1111001.101001)_2 = (1.111001101001 \times 2^6)_2$$

$$s : 1, e : 6 + 127 = 133 = 10000101, m : 111001101001$$

بنابراین جواب نهایی بصورت زیر خواهد شد:

$$1|10000101|11100110100100000000000$$

مثال: عدد 431.136964 را به فرمت اعداد اعشاری مدنظر در مبنای ۲ حاصل کنید:

$$\begin{aligned}
(431)_{10} &= (11010111)_2 \\
0.136964 \times 2 &= 0.273928 \\
0.273928 \times 2 &= 0.547856 \\
0.547856 \times 2 &= 1.095712 \\
0.095712 \times 2 &= 0.191424 \\
0.191424 \times 2 &= 0.382848 \\
0.382848 \times 2 &= 0.765696 \\
0.765696 \times 2 &= 1.531392 \\
0.531392 \times 2 &= 1.062784 \\
0.062784 \times 2 &= 0.125568 \\
(431)_{10} &= (1.1010111001000110 * 2^7)_2 \\
s = 0, e &= 7 + 127 = 10000110 \\
01000011010101110010001100000000
\end{aligned}$$

نکته: توجه داشته باشید که به دلیل محدود بودن اندازه مانتیس (۲۳ بیت)، در هنگام تبدیل اعداد از مبنای ۱۰ به مبنای ۲ مجبور هستیم عمل ضرب را تا زمانیکه بیتها پر شوند ادامه دهیم. این مسئله باعث می شود که مقدار تقریبی اعداد در کلمه ۳۲ بیتی ذخیره گردد. بنابراین در هنگام کار با اعداد اعشاری به این مسئله توجه کنید

یک عدد اعشاری ذخیره شده به فرم استاندارد موردنظر به شکل زیر تفسیر می گردد:

- اگر $0 < e < 255$ آنگاه $N = (-1)^s \times (1.m) \times 2^{e-127}$

- اگر e=۰ و m غیر صفر باشد آنگاه $N = (-1)^s \times (0.m) \times 2^{e-126}$

نکته: که به آن عدد غیرنرمال (unnormalized) گفته می شود. دلیل این مسئله آنست که با کوچکترین توان ممکن یعنی ۱۲۶- بتوان کوچکترین مانتیس ممکن (بدون اضافه شدن ۱) را برای نمایش اعداد کوچک داشت.

- اگر e=۰ و m=۰ آنگاه N=۰ . که البته بسته به میزان s مقدار آن +۰ یا -۰ خواهد بود.
- اگر e=۲۵۵ و m غیرصفر باشد آنگاه حاصل یک عدد نیست. به اختصار NaN یا Not a Number نوشته خواهد شد.
- اگر e=۲۵۵ و m=۰ و s=۰ آنگاه عدد برابر مثبت بینهایت است.
- اگر e=۲۵۵ و m=۰ و s=۱ آنگاه عدد برابر منفی بینهایت است.

۵.۵.۱ نمایش کاراکترها در کامپیوتر

در کامپیوترها علاوه بر اطلاعات عددی، گاهی لازم است که حروف و علائم نیز ذخیره گردد که به آنها کاراکتر می گوییم. برای ذخیره سازی کاراکترها به هر یک از آنها یک کد عددی نسبت داده شده است و در حقیقت کد عددی هر کاراکتر در کامپیوتر ذخیره می گردد. در گذشته پر کاربردترین کد مورد استفاده، کد ASCII بود که برای نمایش هر کاراکتر از یک بایت استفاده می کرد. از آنجا که هر بایت می تواند بین ۰ تا ۲۵۵ تغییر کند، بنابراین تا ۲۵۶ کاراکتر قابل تعریف است. از این بین کدهای بین ۰ تا ۱۲۷ بصورت استاندارد برای علائم و حروف انگلیسی تعریف شده است و کدهای بالاتر از ۱۲۷ برای هر کشور خالی گذاشته شده است تا بتوانند حروف خاص زبان خود را تعریف کنند. بعنوان مثال به کدهای ASCII زیر دقت کنید:

$$A = 65, B = 66, C = 67, \cdots, 0 = 48, 1 = 49 \cdots$$

اما امروزه و بدلیل ارتباطات گسترده جهانی از طریق اینترنت، نیاز به تعریف یک کد بین المللی می باشد که کلیه زبانهای جهانی را دربرگیرد. چراکه متنی که در کشور دیگری به زبان فارسی نوشته می شود باید در ایران هم قابل خواندن باشد و لازمه این مسئله یکسان بودن کدها است. بهمین دلیل اخیرا کد بین المللی بنام Unicode ابداع شده است که تقریبا تمام زبانهای زنده دنیا (از جمله زبان فارسی) را دربر می گیرد. البته این کد از ۲ بایت برای نمایش هر کاراکتر استفاده می کند و سیستم عاملهای جدید همگی از آن حمایت می کنند.

۶.۱ زبان

به صفر و یک زبان ماشین نیز می گویند و وابسته به سخت افزار است. سطح بالاتر از سطح زبان ماشین زبان اسمبلی است. زبان اسمبلی راحت تر از زبان ماشین است. این زبان با دستورات پردازنده و حافظه اولیه و حافظه هایی با نام ثبات ها کار میکند. این زبان وابستگی زیادی با CPU دارد. برنامه ای به نام Assembler زبان اسمبلی را به زبان ماشین تبدیل می کند.

ثبات ها: واحدهایی در(CPU) برای ذخیره اطلاعات.

۱.۶.۱ زبان سطح بالا

زبان های سطح بالا به زبان طبیعی انسان نزدیک است. زبان های سطح بالا را کامپایلرها (Compiler) به زبان ماشین تبدیل می کنند. از جمله زبان های سطح بالا را می توان زبان های C, C++, C#, Pascal, Basic, Java, Python, Ruby و غیره را نام برد.

فکر بشر ⟵ زبان طبیعی ⟵ زبان سطح بالا ⟵ زبان اسمبلی ⟵ زبان ماشین

فصل ۲

الگوریتم (Algorithm)

الگوریتم یک رشته مرتب (ordered sequence) از دستورات غیر مبهم و ساخت یافته (structured and well-defined) که کار خاصی را انجام دهد و در زمان متناهی به پایان برسد.
هر الگوریتم باید دارای شرایط زیر باشد:

- ورودی: یک الگوریتم می تواند صفر یا چند ورودی داشته باشد که از محیط خارج تامین می گردد.
- خروجی: الگوریتم باید یک یا چند کمیت خروجی داشته باشد.
- قطعیت: هر دستورالعمل باید واضح و بدون ابهام باشد.
- کارایی: هر دستورالعمل باید قابل اجرا باشد.
- محدودیت: در تمام حالات، الگوریتم باید پس از طی مراحل محدودی خاتمه یابد

در علم کامپیوتر، ما معمولا با یک مسئله مواجهیم که باید آن را حل کنیم. این مسئله می تواند در زمینه های مختلفی همچون علمی، اقتصادی، ریاضی، فنی و ... باشد. معمولا برای حل یک مسئله، مراحل زیر طی می گردد:

- تعریف مسئله بصورت جامع و دقیق (شامل تعریف ورودیها و خروجیها)
- بررسی راه حلهای مختلف برای حل مسئله
- انتخاب مناسبترین راه حل و تهیه یک الگوریتم برای آن
- آزمایش الگوریتم با داده های ورودی و اشکالزدایی آن
- تبدیل الگوریتم به یک زبان برنامه نویسی کامپیوتری (مانند C یا Pascal)
- وارد کردن برنامه به کامپیوتر و تست و اشکالزدایی آن استفاده از برنامه

روش های نمایش الگوریتم

- زبان طبیعی.
- زبان های برنامه نویسی.
- فلوچارت ، نمایش گرافیکی الگوریتم.
- شبکه کد (pseudo-code) ⟵ سودوکد

۱.۲ نمونه مثال هایی از الگوریتم با زبان های طبیعی

هر الگوریتم یک شروع و یک پایان دارد. در ادامه نمونه مثال های از الگوریتم های با زبان طبیعی ارائه شده است.

مثال: الگوریتم روشن کردن لامپ را بنویسید.

۰-شروع ۱-به سمت کلید لامپ حرکت می کنیم. ۲-دستمان را دراز می کنیم. ۳-کلید لامپ را می فشاریم. ۴-پایان

مثال: الگوریتم تلفن زدن را بنویسید.

۰-شروع ۱-گوشی تلفن را برمی داریم. ۲-سکه را درون آن می اندازیم. ۳-شماره گیری می کنیم. ۴-مکالمه را انجام می دهیم. ۵-گوشی را بگذاریم در جای خودش. ۶-پایان.

نکته: الگوریتم ها دارای دستورات ⟵ ۱- متوالی ۲- شرطی ۳- حلقه ای هستند.

مثال: الگوریتم تعویض چرخ پنچر یک ماشین را بنویسید.

۰-شروع ۱-جک را زیر اتومبیل قرار می دهیم. ۲-پیچ پرخ اتومبیل را باز کنید. ۳-چرخ پنچر را از جای خود دراورده. ۴-چرخ سالم را جای آن قرار داده. ۵- پیچ را سفت می کنیم. ۶-اگر پیچ سفت نشده برو به ۵. ۷- جک را باز کن. ۸-پایان.

مثال: چاپ کردن یک عدد.

۰-شروع ۱-عدد a را بگیر. ۲- عدد a را چاپ کن. ۳-پایان.

مثال: تبدیل متر به سانتی متر.

۰-شروع ۱-عدد a را بگیر. ۲- عدد $a = a \times 100$. ۳- عدد a چاپ کن. ۴- پایان.

مثال: دو عدد را بگیرد اگر مجموع بیشتر از ۲۰ است بنویسید بلی وگرنه بنویسید خیر.

۰-شروع ۱- عدد a و b را بگیرد. ۲- $c = a + b$ ۳-اگر $c > 20$ برو به ۶. ۴-چاپ کن خیر. ۵- برو به ۷. ۶- چاپ کن بلی. ۷- پایان

نکته: مراحلی از الگوریتم که اجزای آن بیش از یکبار انجام میشود تشکیل یک حلقه یا (loop) را می دهند.

الگوریتم ۱ الگوریتمی بنویسید که اعداد طبیعی مضارب هفت کوچک تر از پنجاه را با هم جمع و نتیجه حاصله را چاپ کند.

---

start
$N = 0$
$S = 0$
$N = N + 7$
if $N < 50$ **then**
  $S = S + N$
  go to 4
else
  print S
end if
end

فلوچارت

نمایش گرافیکی الگوریتم را فلوچارت می گویند. لیست شکل های اجزاری اصلی فلوچارت را در شکل ۱.۲ مشاهده می شود.

اجزای اصلی فلوچارت:

- شروع و پایان
- ورودی و خروجی
- محاسبات عددی
- تصمیم گیری

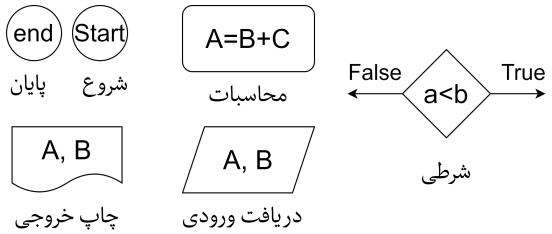
الگوریتم ۲ الگوریتمی بنویسید که هزار عدد از ورودی بگیرد و مینیم آن ها را چاپ کند

0. start
1. get a
2. min = a;
3. i = 1;
4. if i>=1000 goto 10
5. i=i+1
6. get a;
7. if a >= min goto 4
8. min = a;
9. goto 4;
10. print min;
11. end

نکته: به جهت انجام تصمیم گیری از عملگرهای تصمیم گیری زیر می توان استفاده کرد.

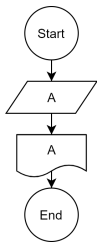
- $>$ برای مفهوم بزرگتر مانند: $a > b$.
- $<$ برای مفهوم کوچکتر مانند: $a < b$.
- $=>$ برای مفهوم بزرگتر و یا مساوی مانند: $a >= b$.
- $<=$ برای مفهوم کوچکتر و یا مساوی مانند: $a <= b$.
- $==$ برای مفهوم مساوی مانند: $a == b$
- $= !$ برای مفهوم مخالف مانند: $a! = b$

نکته: نتیجه عملگرهای محاسباتی همواره درست (True) یا غلط (False) است.
نکته: از عملگرهای شرطی $||$ برای مفهوم یا و از عملگر شرطی $\&\&$ برای مفهوم و استفاده می شود. همچنین برای معکوس کردن شرط از عملگر $!$ استفاده می شود.

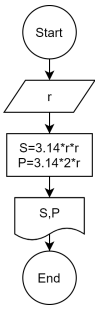


شکل ۱.۲: اجزای نمایش فلوچارت

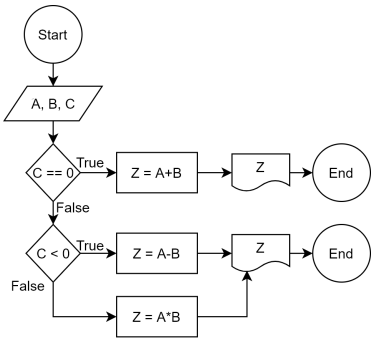
مثال: فلوچارت چاپ کردن یک عدد را بنویسید. جواب در شکل ۲.۲ نشان داده شده است.
مثال: فلوچارتی رسم کنید که شعاع دایره را بگیرد و محیط و مساحت آن را چاپ کند. جواب در شکل ۳.۲ نشان داده شده است.
مثال: فلو چارتی رسم کنید که سه عدد را بگیرد اگر عدد سوم صفر بود جمع دوعدد و اگر عدد سوم منفی بود تفریق دو عدد و اگر عدد سوم مثبت بود ضرب دو عدد را چاپ کند. جواب در شکل ۴.۲ نشان داده شده است.
مثال: فلوچارتی رسم کنید که دو عدد را بگیرد و مقدار آن ها را جا به جا کند. جواب در شکل ۵.۲ نشان داده شده است.
مثال: فلوچارتی رسم کنید که سه ضلع مثلث را بخواند و بگوید که قائمه است یا نه؟ جواب در شکل ۶.۲ نشان داده شده است.



شکل ۲.۲: فلوچارت چاپ کردن یک عدد.



شکل ۳.۲: فلوچارت محاسبه شعاع دایره و محیط و مساحت آن.



شکل ۴.۲: فلوچارت عملیات مربوط به علامت های متغیرهای ورودی.

مثال: فلوچارت بدست اوردن ریشه های معادله درجه دو. جواب در شکل ۷.۲ نشان داده شده است.

نکته: A/B:خارج قسمت تقسیم A به B و $A\%B$: باقی مانده تقسیم AبهB.

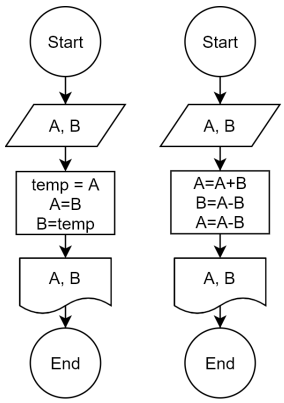
مثال: ده عدد بگیرد و مشخص کند کدام زوج است و کدام فرد. جواب در شکل ۸.۲ نشان داده شده است.

مثال: یک عدد بزرگتر از صفر بگیرد و سپس به تعداد آن عدد اعداد دیگری را خوانده و مجموع و میانگین آن ها را چاپ کند. جواب در شکل ۹.۲ نشان داده شده است.

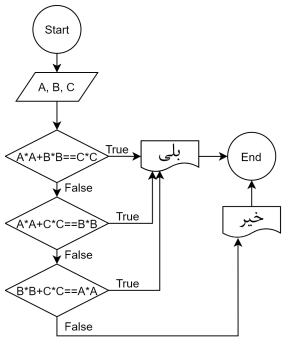
مثال: فلوچارتی رسم کنید که معدل سیزده درس یک دانش آموز را محاسبه و چاپ کند که هردرس دارای سه نمره ثلث اول و ثلث دوم و ثلث سوم است که ضریب ثلث اول و دوم یک است و ضریب ثلث سوم دو است. جواب در شکل ۱۰.۲ نشان داده شده است.

مثال: یک عدد بزرگتر از صفر بگیرد و سپس به تعداد آن عدد اعداد دیگری را خوانده و در نهایت بیشینه مقداری را که دریافت کرده را چاپ کند. جواب در شکل ۱۱.۲ نشان داده شده است.

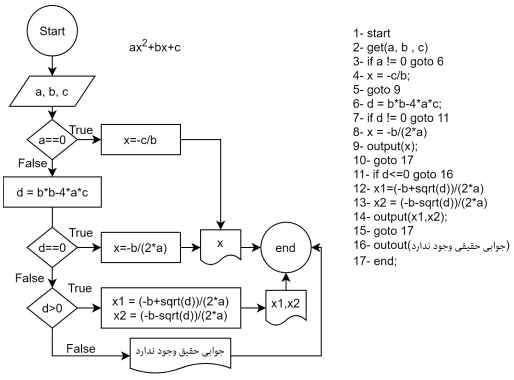
مثال: فلوچارتی رسم کنید که اضلاع یک مثلث را بگیرد و تعیین کند که ایا این سه ضلع میتواند اضلاع مثلث باشد یا خیر؟ جواب در شکل ۱۲.۲ نشان داده شده است.



شکل ۵.۲: فلوچارت برای جابجایی دو متغیر.



شکل ۶.۲: فلوچارت برای تعیین قائم الزاویه بودن سه عدد ورودی.

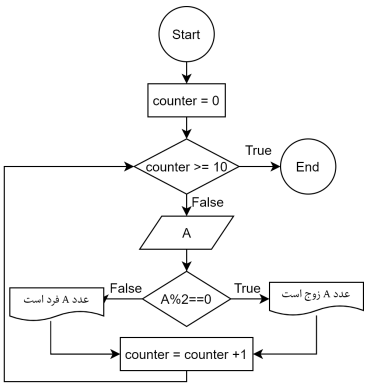


شکل ۷.۲: فلوچارت بدست آوردن ریشه های معادله درجه دو.

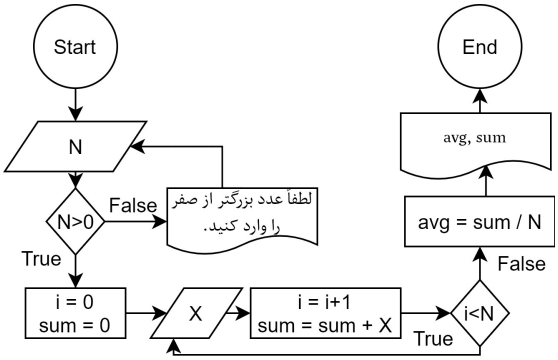
مثال: میخواهیم اعداد زوج یک تا پانزده را چاپ کنیم اشکال فلوچارت ارائه شده در شکل ۱۳.۲ چیست؟
مشکل این الگوریتم این است که هیچ وقت به پایان نمیرسد و شکل درست آن $n < 15$ است.

مثال: فلوچارتی رسم کنید که خارج قسمت و باقی مانده تقسیم Nبر M را چاپ کند. جواب در شکل ۱۴.۲ نشان داده شده است.

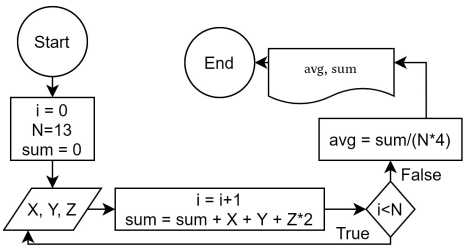
مثال: فلوچارتی طراحی کنید که فاکتوریل یک عدد طبیعی را حساب کند. جواب در شکل ۱۵.۲ نشان داده



شکل ۸.۲: فلوچارت زوج و فرد بودن اعداد.



شکل ۹.۲: فلوچارت برای میانگین اعداد.



شکل ۱۰.۲: فلوچارت برای میانگین معدل نمرات دانشجویی.

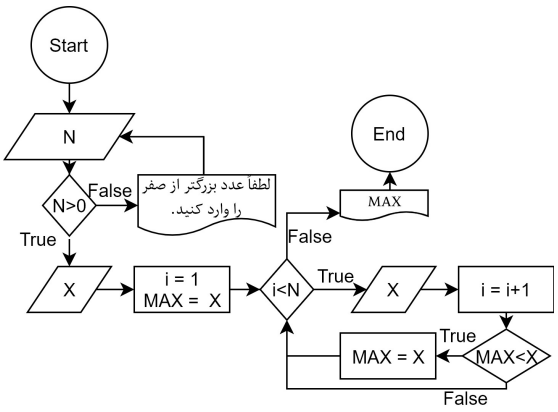
شده است.

مثال: فلوچارتی طراحی کنید که تعداد ارقام یک عدد طبیعی را چاپ کند. جواب در شکل ۱۶.۲ نشان داده شده است.

مثال: فلوچارتی طراحی کنید که مقسوم علیه های طبیعی یک عدد را چاپ کند. جواب در شکل ۱۷.۲ نشان داده شده است.

مثال: فلوچارتی طراحی کنید که تعیین اول بودن یک عدد طبیعی را انجام دهد. جواب در شکل ۱۸.۲ نشان داده شده است.

مثال: فلوچارتی طراحی کنید که n را به عنوان یک عدد طبیعی بخواند و مجموع ارقام آن را نمایش دهد.



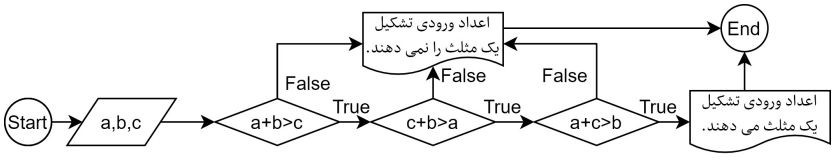
شکل ۱۱.۲: فلوچارت برای پیدا کردم مقدار بیشینه.

الگوریتم ۳ الگوریتمی بنویسید که قیمت نهایی یک کالا را محاسبه کند.

1: start
2: get priceofitem
3: get salestaxrate
4: $salestaxrate = priceofitem * salestaxrate$
5: $finalprice = salestaxrate + priceofitem$
6: **print** finalprice
7: end

الگوریتم ۴ الگوریتمی بنویسید که دستمزد هفتگی یک کارمند را حساب کند و اگر کارمند بیش از چهل ساعت در هفته کار کند به ازای هر ساعت اضافه کاری، یک و نیم برابر ساعات معمولی، حقوق دریافت کند.

1: start
2: get $hoursworked$
3: get $payerate$
4: **if** $hoursworked <= 40$ **then**
5: $finalpay = hoursworked * payrate$
6: **else**
7: $finalpay = 40 * payrate + (hoursworked - 40) * 1.5 * payrate$
8: **end if**
9: **print** $finalpay$
10: end



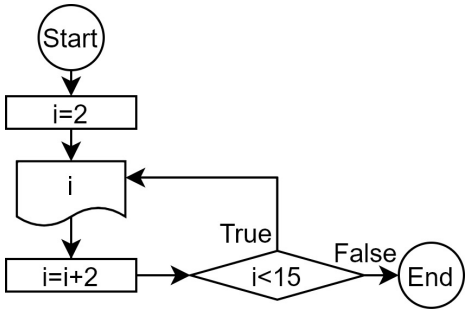
شکل ۱۲.۲: فلوچارت برای تشخیص مثلث ساز بودن سه عدد ورودی.

جواب در شکل ۱۹.۲ نشان داده شده است.

مثال: فلوچارتی بنویسید که n را خوانده و تعیین کند کامل است یا خیر؟ (عددی کامل است که مجموع

الگوریتم ۵ الگوریتمی بنویسید که معدل نمرات کوییز ها را حساب کند.

```
1: start
2: get numofquizzes
3: sum = 0
4: count = 0
5: if count<=numofquizzes then
6:    get quizzgrade
7:    sum = sum + quizzgrade
8:    count = count + 1
9:    go to 5
10: else
11:    average = sum/numofquizzes
12: end if
13: print average
14: end
```



شکل ۱۳.۲: فلوچارت برای تشخیص مثلث ساز بودن سه عدد ورودی.

مقسوم علیه های کوچکتر از خودش با خودش برابر باشد).

جواب در شکل ۲۰.۲ نشان داده شده است.

مثال: الگوریتمی بنویسید که مقلوب یک عدد طبیعی را محاسبه کند.

۰-شروع. ۱-n را بگیر. ۲-reverse=0 ۳-تا وقتی که n<=0 برو به ۷. ۴-revers=revers*10+n%10 ۵- n=n/10 ۶- برو به ۳. ۷- revers را چاپ کن. ۸- پایان.

البته شکل فلوچارت این مسئله نیز در شکل ۲۱.۲ نشان داده شده است.

مثال: e^x را با دقت 0.000001 حساب کنید.

$$e^x = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

۰-شروع. ۱-x را بگیر. ۲-s=0, t=1, i=1 ۳-تا وقتی که t>0.000001 هست ادامه بده و غیر این صورت برو به ۸. ۴-$t = t * (\frac{x}{i})$ ۵. s=s+t ۶-i=i+1. ۷- برو به ۳ ۸-چاپ کن s را. ۹- پایان.

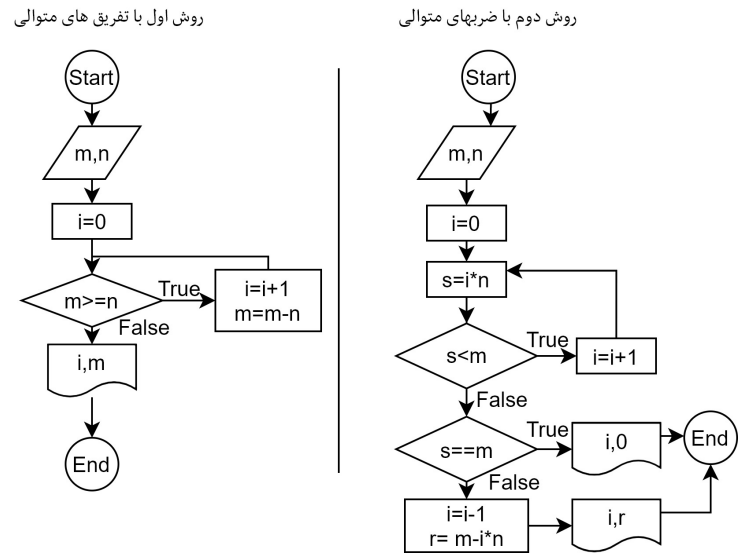
البته شکل فلوچارت این مسئله نیز در شکل ۲۲.۲ نشان داده شده است.

نکته: تعریف آرایه: مجموعه ای از داده های همنوع است که تحت یک نام مشترک ذخیره می گردند.

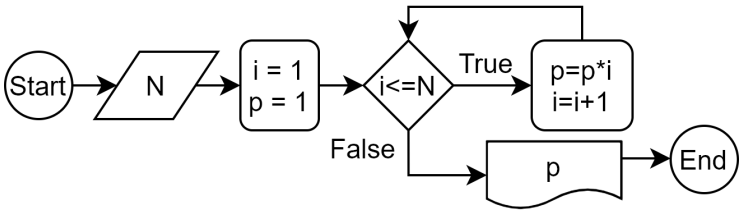
نکته: برای دسترسی به هر یک از اعضا یا عناصر آرایه از نام آرایه بعلاوه یک اندیس استفاده می شود. بنابراین هر عنصر آرایه درحقیقت یک متغیر مستقل از همان نوع مورد نظر است.

نکته: یک آرایه پیش از آنکه استفاده گردد باید اعلان شود. اعلان آرایه شامل نام آرایه و اندازه آن است. عناصر آرایه برای سهولت در دسترسی (معمولا) در خانه های پشت سرهم حافظه ذخیره می گردند.

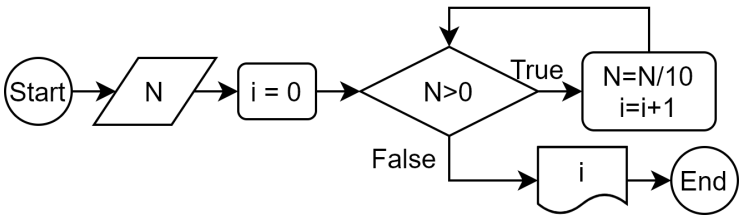
مثال: آرایه A را با ۱۰۰ عضو از اعداد صحیح درنظر بگیرید. به شکل ۲۳.۲ توجه کنید.



شکل ۱۴.۲: فلوچارت برای محاسبه خارج قسمت و باقیمانده تقسیم M بر N.



شکل ۱۵.۲: فلوچارت فاکتوریل یک عدد طبیعی.

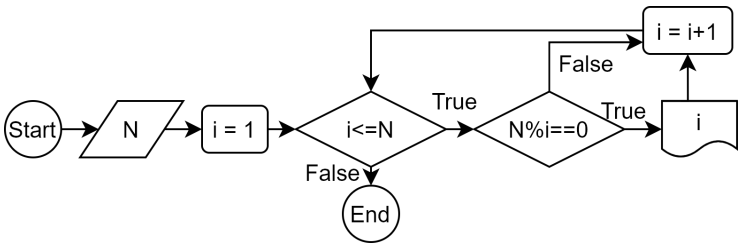


شکل ۱۶.۲: فلوچارت چاپ تعداد ارقام یک عدد طبیعی.

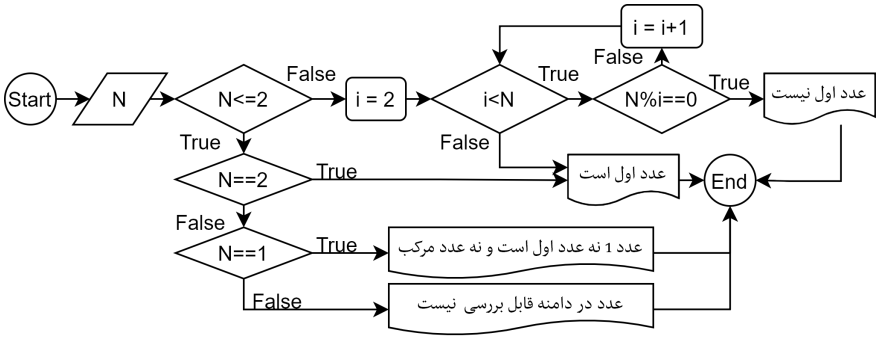
مثال: الگوریتمی طراحی کنید تا مجموع اعداد یک لیست را چاپ کنید.
۰-شروع. ۱- یک لیست array به نام a از ۱ تا n را بگیر. ۲-$s=0, i=1$ ۳- تا وقتی که $i<=n$ ، ۱.۳-$s=s+a[i]$. ۲.۳-$i+1$. ۴- چاپ کن s را. ۵-پایان.
البته شکل فلوچارت این مسئله نیز در شکل ۲۴.۲ نشان داده شده است.

مثال: الگوریتمی طراحی کنید تا بیشینه یا max یک لیست ورودی را چاپ کند.
۰-شروع. ۱-یک لیست به نام a از ۱ تا n بگیر. ۲-$i=2$ ۳-$max=a[i]$ ۴-تا وقتی که $i<=n$، ۱.۴-اگر $a[i]$ بزرگتر از max باشد آنگاه $max=a[i]$ ۲.۴-$i=i+1$ ۵-maxرا چاپ کن ۶-پایان.

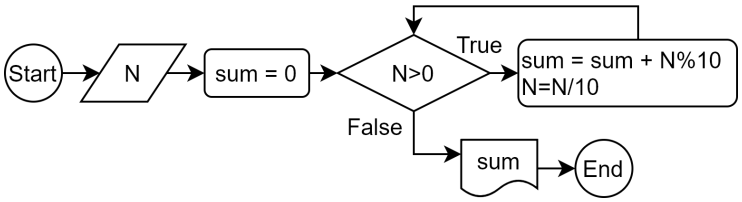
مثال: الگوریتمی بنویسید جهت یافتن جای index عددی که به عنوان ورودی دریافت میگردد.



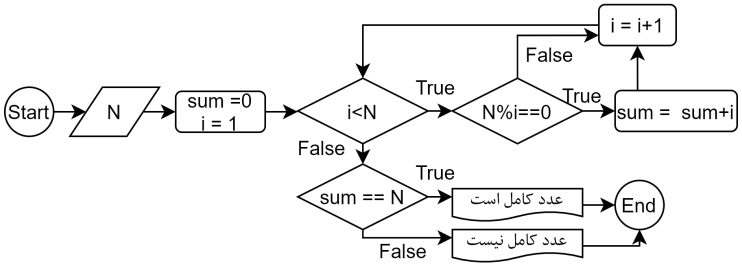
شکل ۱۷.۲: فلوچارت چاپ مقسوم علیه های یک عدد طبیعی.



شکل ۱۸.۲: فلوچارت تعیین اول بودن یک عدد طبیعی.



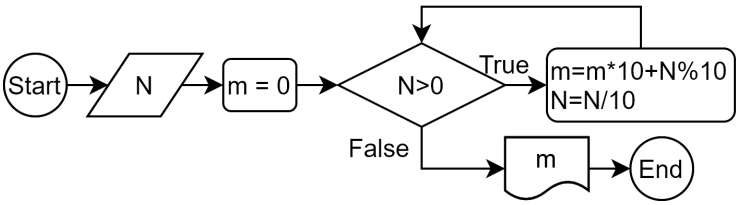
شکل ۱۹.۲: فلوچارت محاسبه مجموع ارقام یک عدد طبیعی.



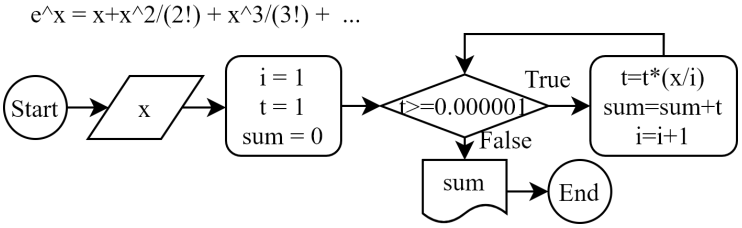
شکل ۲۰.۲: فلوچارت محاسبه کامل بودن یک عدد طبیعی.

۰-شروع. ۱-یک لیست به نام a از ۱ تا n بگیرد. ۲-k را بگیر. ۳-index=0. ۴- اگر index>=n برو به ۸. ۵-اگر a[index] برابر با k باشد آنگاه برو به ۶-۱. ۶-index=index+1. ۷- برو به ۴. ۸- ۱- را چاپ کن و برو به ۱۰. ۹- index .را چاپ کن ۱۰-پایان.

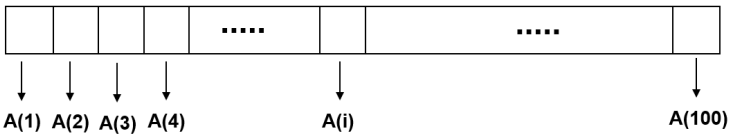
مثال: فلوچارتی طراحی کنید که یک لیست را دریافت و آن را معکوس کند.



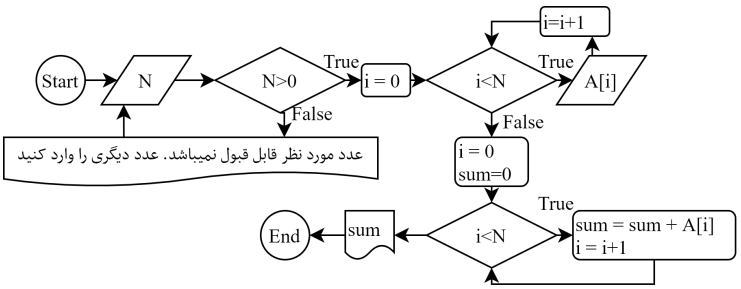
شکل ۲۱.۲: فلوچارت محاسبه مقلوب یک عدد طبیعی.



شکل ۲۲.۲: فلوچارت محاسبه e^x با دقت ۱.۰۰۰۰۰۰



شکل ۲۳.۲: نمونه آرایه ای با ۱۰۰ خانه از نوع عدد.



شکل ۲۴.۲: فلوچارت محاسبه مجموع اعداد یک لیست

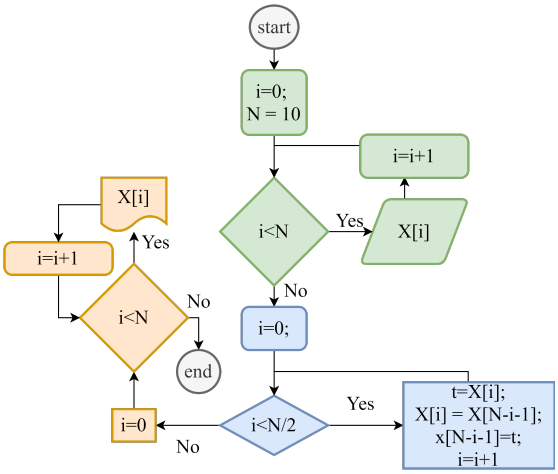
جواب در شکل ۲۵.۲ نشان داده شده است.

مثال: الگوریتمی طراحی کنید تا جذر یک عدد طبیعی را حساب کند.

n=a/i -۷ i=i-1 -۶ deghat=0 -۵ if i*i<a:go to 3 -۴ i=i+1 -۳ i=1.0 -۲ get a -۱ start -۰
goto -۱۳ if deghat==4: go to 14 -۱۲ deghat = deghat+1 -۱۱ i=s -۱۰ s=s/2 -۹ s=n+i -۸
finish -۱۵ print i -۱۴ 7

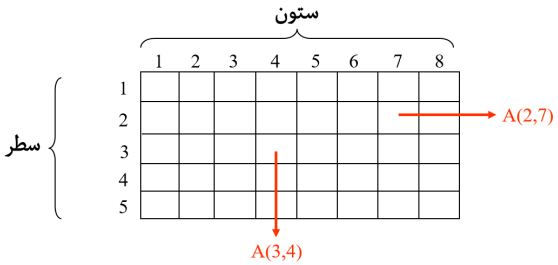
نکته: مسائلی که تاکنون حل شدند نیاز به آرایه های یک بعدی داشتند. هر عنصر از این آرایه ها تنها با یک اندیس مشخص می گردد. اما گاهی در مسائل پیچیده تر نیاز به آرایه هایی است که هر عضو آنها نیاز به بیش از یک اندیس دارد، که به آنها آرایه های چند بعدی گفته می شود.

نکته: چنانچه هر عنصر آرایه به ۲ اندیس نیاز داشته باشد، به آن آرایه دو بعدی می گوییم. برای تعریف یک



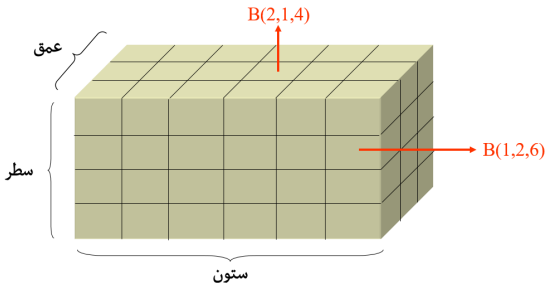
شکل ۲۵.۲: فلوچارت محاسبه معکوس کردن یک لیست.

آرایه دوبعدی باید تعداد سطرها و ستونهای آن را مشخص کنیم. معمولا یک آرایه دو بعدی بصورت $m \times n$ اعلان می گردد که m تعداد سطرها و n تعداد ستونها است. جهت مشاهده ساختار به شکل ۲۶.۲ توجه کنید.



شکل ۲۶.۲: مثالی از آرایه های دو بعدی با 5×8

نکته: برای آرایه های سه بعدی نیز مفاهیم مشابهی قابل طرح است. در این آرایه ها هر عنصر نیاز به ۳ اندیس دارد و برای تعریف آنها را بصورت $p \times m \times n$ اعلان می کنیم که p عمق، m تعداد سطرها و n تعداد ستونها است. بعنوان مثال چنانچه آرایه B بعنوان یک آرایه سه بعدی به ابعاد $3 \times 4 \times 6$ در شکل ۲۷.۲ قابل مشاهده است.



شکل ۲۷.۲: مثالی از آرایه سه بعدی با $3 \times 4 \times 6$ خانه

فصل ۳

برنامه نویسی به زبان C و C++

در سال ۱۹۶۷ مارتین ریچاردز زبان BCPL را برای نوشتن نرم افزارهای سیستم عامل و کامپایلر در دانشگاه کمبریج ابداع کرد. در سال ۱۹۷۰ کن تامپسون زبان B را بر مبنای ویژگیهای زبان BCPL نوشت و از آن برای ایجاد اولین نسخه های سیستم عامل Unix در آزمایشگاههای بل استفاده کرد. زبان C در سال ۱۹۷۲ توسط دنیس ریچی از روی زبان B و BCPL در آزمایشگاه بل ساخته شد و ویژگی های جدیدی همچون نظارت بر نوع داده ها نیز به آن اضافه شد. ریچی از این زبان برای ایجاد سیستم عامل Unix استفاده کرد اما بعدها اکثر سیستم عامل های دیگر نیز با همین زبان نوشته شدند. این زبان با سرعت بسیاری گسترش یافت و چاپ کتاب "The C Programming Language" در سال ۱۹۷۸ توسط کرنیگان و ریچی باعث رشد روزافزون این زبان در جهان شد.

$$BSCP \xrightarrow[\text{برای نوشتن سیستم عامل unix}]{1970} B \xrightarrow[\text{در آزمایشگاه BELL}]{۱۹۷۸} C$$

در سال ۱۹۸۳ انستیتوی ملی استاندارد آمریکا (ANSI) کمیته ای موسوم به X۳J۱۱ را مامور کرد تا یک تعریف فاقد ابهام و مستقل از ماشین را از این زبان تدوین نماید. در سال ۱۹۸۹ این استاندارد تحت عنوان ANSI C به تصویب رسید و سپس در سال ۱۹۹۰، سازمان استانداردهای بین المللی (ISO) نیز این استاندارد را پذیرفت و مستندات مشترک آنها تحت عنوان ANSI/ISO C منتشر گردید.

در سالهای بعد و با ظهور روش های برنامه نویسی شی گرا نسخه جدیدی از زبان C بنام C++ توسط بیارنه استراوستروپ در اوایل ۱۹۸۰ در آزمایشگاه بل توسعه یافت. در C++ علاوه بر امکانات جدیدی که به زبان C اضافه شده است، خاصیت شی گرایی را نیز به آن اضافه کرده است.

$$C \xrightarrow[\text{توسعه شی گرایی}]{1980} C++$$

شرکت سان مایکروسیستمز در سال ۱۹۹۵ میلادی زبان Java را برمبنای C و C++ ایجاد کرد که هم اکنون از آن در سطح وسیعی استفاده می شود و برنامه های نوشته شده به آن بر روی هر کامپیوتری که از Java پشتیبانی کند (تقریبا تمام سیستم های شناخته شده) قابل اجرا می باشد. شرکت مایکروسافت در رقابت با شرکت سان، در سال ۲۰۰۲ زبان جدیدی بنام C# (سی شارپ) را ارائه داد که رقیبی برای Java بشمار می رود.

$$Java \xleftarrow[\text{چند سیستم عاملی}]{1995} C++ \xrightarrow[\text{توسعه امکانات}]{2002} C\#$$

۱.۳ برنامه نویسی ساختیافته

در دهه ۱۹۶۰ میلادی توسعه نرم افزار دچار مشکلات عدیده ای شد. در آن زمان سبک خاصی برای برنامه نویسی وجود نداشت و برنامه ها بدون هیچگونه ساختار خاصی نوشته می شدند. فعالیتهای پژوهشی در این دهه باعث بوجود آمدن سبک جدیدی از برنامه نویسی بنام روش ساخت یافته گردید؛ روش منظمی که باعث ایجاد برنامه هایی کاملا واضح و خوانا گردید که اشکال زدایی و خطایابی آنها نیز بسیار ساده تر بود. اصلی ترین نکته در این روش عدم استفاده از دستور پرش (goto) است. تحقیقات بوهم و ژاکوپینی نشان داد که می توان هر برنامه ای را بدون دستور پرش و فقط با استفاده از ۳ ساختار کنترلی ترتیب، انتخاب و تکرار نوشت.

- ساختار ترتیب، همان اجرای دستورات بصورت متوالی (یکی پس از دیگری) است.
- ساختار انتخاب به برنامه نویس اجازه می دهد که براساس درستی یا نادرستی یک شرط، تصمیم بگیرد کدام مجموعه از دستورات اجرا شود.
- ساختار تکرار نیز به برنامه نویسان اجازه می دهد مجموعه خاصی از دستورات را تا زمانیکه شرط خاصی برقرار باشد، تکرار نماید.

هر برنامه ساختیافته از تعدادی بلوک تشکیل می شود که این بلوکها به ترتیب اجرا می شوند تا برنامه خاتمه یابد (ساختار ترتیب). هر بلوک می تواند یک دستور ساده مانند خواندن، نوشتن یا تخصیص مقدار به یک متغیر باشد و یا اینکه شامل دستوراتی باشد که یکی از ۳ ساختار فوق را پیاده سازی کنند. نکته مهم اینجاست که درمورد دستورات داخل هر بلوک نیز همین قوانین برقرار است و این دستورات می توانند از تعدادی بلوک به شرح فوق ایجاد شوند و تشکیل ساختارهایی مانند حلقه های تودرتو را دهند. نکته مهم اینجاست که طبق قوانین فوق یک حلقه تکرار یا بطور کامل داخل حلقه تکرار دیگر است و یا بطور کامل خارج آن قرار می گیرد و هیچگاه حلقه های روی هم افتاده نخواهیم داشت.

از جمله اولین تلاش ها در زمینه ساخت زبان های برنامه نویسی ساختیافته، زبان پاسکال بود که توسط پروفسور نیکلاس ویرث در سال ۱۹۷۱ برای آموزش برنامه نویسی ساختیافته در محیط های آموزشی ساخته شد و به سرعت در دانشگاه ها رواج یافت.

کمی بعد زبان C ارائه گردید که علاوه بر دارا بودن ویژگی های برنامه نویسی ساختیافته بدلیل سرعت و کارایی بالا مقبولیتی همه گیر یافت و هم اکنون سالهاست که بعنوان بزرگترین زبان برنامه نویسی دنیا شناخته شده است.

۲.۳ مراحل اجرای یک برنامه

همانطور که در شکل ۱.۳ نشان داده شده است، مراحل نوشتن در C++ به شرح زیر است.

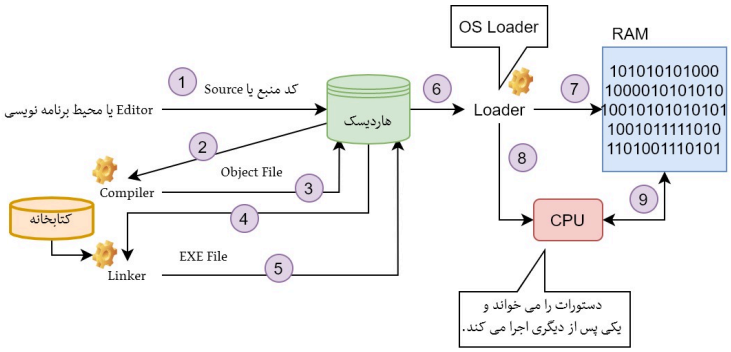
۱. نوشتن برنامه یا (editing) : source برنامه ها را با استفاده از محیط های با نام editor می توان نوشت.

۲. کامپایل کردن برنامه با زبان C : تا برای ماشین قابل فهم باشد و در نهایت object file را برای ما ایجاد خواهد کرد.

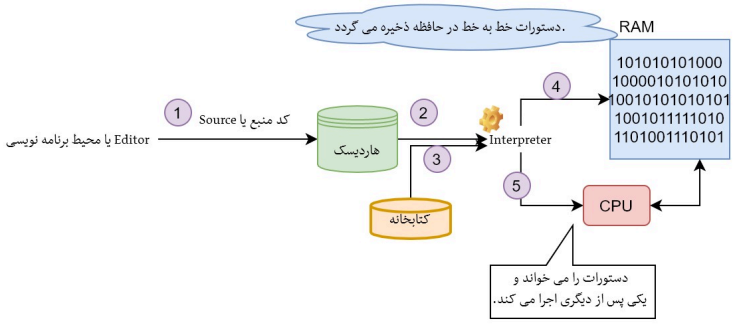
۳. linking:بسیاری از زبان های کامپایلری، دارای روتین های از پیش نوشته شده ای هستند که در کارخانه ی سازنده ی کامپایلر نوشته می شوند و ما میتوانیم در برنامه خود از آن ها استفاده کنیم.

از جمله این کتابخانه ها در زبان C کتابخانه Standard Template Library (STL): تو این مرحله ست که فایل text.exe تولید میشود.

۴. اجرا کردن برنامه.



شکل ۱.۳: مراحل نوشتن و تولید کد و اجرای برنامه در زبان C و C++ .



شکل ۲.۳: مراحل اجرای برنامه در مفسرها.

مسلما طی این مراحل برای برنامه نویسان بسیار زمانبر است. راه حل این مشکل استفاده از محیط مجتمع توسعه نرم افزار یا IDE (Integrated Development Environment) است.
یک IDE شامل مواردی همچون:

- ویرایشگر متن با امکانات ویژه برای زبان.
- امکان کامپایل و پیوند دادن برنامه یا linking.
- امکان اشکال زدایی (Debug) برنامه.

چند محیط معروف برنامه نویسي عبارتند از:

- Microsoft Visual C++
- Dev-C++

۳.۳ خطاهای برنامه نویسی

بنظر می رسد خطاها جزء جداناپذیر برنامه ها هستند. بندرت می توان برنامه ای نوشت که در همان بار اول بدرستی و بدون هیچگونه خطایی اجرا شود.
بطور کلی خطاها به دو دسته تقسیم می شوند:

- خطاهای نحوی (خطاهای زمان کامپایل): این خطاها در اثر رعایت نکردن قواعد دستورات زبان C و یا تایپ اشتباه یک دستور بوجود می آیند و در همان ابتدا توسط کامپایلر به برنامه نویس اعلام می گردد. معمولا این قبیل خطاها خطر کمتری را در بردارند.
- خطاهای منطقی (خطاهای زمان اجرا): این دسته خطاها در اثر اشتباه برنامه نویس در طراحی الگوریتم درست برای برنامه و یا گاهی در اثر درنظر نگرفتن بعضی شرایط خاص در برنامه ایجاد می شوند. متاسفانه این دسته خطاها در زمان کامپایل اعلام نمی شوند و در زمان اجرای برنامه خود را نشان می دهند. ممکن است یک برنامه نویس خطای منطقی برنامه خود را تشخیص ندهد و این خطا پس از مدتها و تحت یک شرایط خاص توسط کاربر برنامه کشف شود. به همین دلیل این دسته از خطاها خطرناکتر هستند. خود این خطاها به دو دسته تقسیم می گردند:
  - خطاهای مهلک: در این دسته خطاها کامپیوتر بلافاصله اجرای برنامه را متوقف کرده و خطا را به کاربر گزارش می کند. مثال معروف این خطاها، خطای تقسیم بر صفر می باشد.
  - خطاهای غیرمهلک: در این دسته خطا، اجرای برنامه ادامه می یابد ولی برنامه نتایج اشتباه تولید می نماید. بعنوان مثال ممکن است در اثر وجود یک خطای منطقی در یک برنامه حقوق و دستمزد، حقوق کارمندان اشتباه محاسبه شود و تا مدتها نیز کسی متوجه این خطا نشود!

بسیار مهم است که در ابتدا سعی کنید برنامه ای بنویسید که حداقل خطاها را داشته باشد، در گام دوم با آزمایش دقیق برنامه خود هرگونه خطای احتمالی را پیدا کنید و در گام سوم بتوانید دلیل بروز خطا را پیدا کرده و آنرا رفع نمایید. هر سه عمل فوق کار سختی بوده و نیاز به تجربه و مهارت دارد. در اصطلاح برنامه نویسی به هر گونه خطا، bug و به رفع خطا repair گفته می شود. برنامه های ترمیم شده با استفاده از patch های که شما جهت رفع خطا نوشتید بروزرسانی می شوند و مجددا قابل استفاده خواهد بود. همچنین به اجرای خط به خط و مشاهده اطلاعات درون برنامه در لحظه اجرا را debug می گویند. دیباگ کردن توسط دیباگرها قابل انجام است. دیباگرها برنامه هایی هستند که اجرای خط به خط دستورات و مشاهده اطلاعات وضعیت پردازنده ها و حافظه برای برنامه را مقدور می سازد.

۴.۳ کاراکترهای مجاز و کلمات کلیدی

زبان C یک زبان case sensitive است. یعنی بزرگی و کوچکی حروف برای آن اهمیت دارد. از جمله کاراکترهای مجاز را می توان در جدول ۱.۳ مشاهده کنید.

جدول ۱.۳: کاراکترهای مجاز در زبان C و C++ .

A-Z	(	)	+	-	=	>	<
a-z	۹-۰	*	{	}	/	|	&
space	_	»	›	:	.	$	#

جدول ۲.۳: کلمات کلیدی مجاز در زبان C و C++ .

const	char	case	break	auto
else	double	do	default	continue
goto	for	float	extern	enum
return	register	long	int	if
struct	static	sizeof	signed	short
void	unsigned	union	typedef	switch
			while	volatile

زبان C شامل ۳۲ تا keyword یا کلمه کلیدی دارد و همگی lower case هستند. منظور از lower case بودن این است که این کلمات همه با حروف کوچک نوشته می شوند.

زبان C دستورات خیلی ابتدایی دارد. خسته کننده و ملالت آور است حتی دستوری برای چاپ روی خروجی ندارد. البته که کتابخانه‌ای از توابع استاندارد STL دارد که می توانیم از آنها استفاده کنیم.

۵.۳ شناسه ها

شناسه (identifier) نامی است که به یک قسمت از برنامه مانند متغیر و تابع اختصاص داده می شود. در زبان C برای انتخاب شناسه ها فقط می توان از علائم حروف انگلیسی کوچک و بزرگ و اعداد و علامت خط پایین یا _ استفاده می شود. البته یک شناسه نمی تواند با یک رقم شروع شود.

- چند شناسه مجاز: sum ، average ، name۲ وstudent_average
- چند شناسه غیرمجاز: ۲name و یا student average

نکته: در برنامه نویسی امروزی پیشنهاد می شود بجای شناسه هایی همانند student_average از studen-tAverage استفاده گردد.

نکته: نکته مهم دیگری که باید به آن اشاره کرد آن است که زبان C برخلاف بسیاری از زبانهای دیگر به کوچک و بزرگی حروف حساس است (case sensitive). در نتیجه شناسه های زیر با یکدیگر متفاوتند: SUM≠Sum≠sum

نکته: در هنگام انتخاب شناسه نمی توانید از کلمات کلیدی که برای منظورهای خاص در زبان C رزرو شده اند استفاده کنید.

Hello world ۶.۳

تابع ای با نام main آغاز کننده برنامه است. برای شروع هر دستوری که درون این تابع نوشته شود، اجرا خواهد شد. برای ایجاد این تابع یک فایل متنی ایجاد کنید و دستور زیر را در آن بنویسید و بعد کامپایل کنید.

```
int main(){
        // code block
}
```

دستورات اجرایی درون تابع main نوشته می‌شوند. ساده ترین برنامه "Hello world!":

```
int main(){
        cout <<"Hello World!";
}
```

در دستور بالا درون تابع main از تابع cout استفاده شده است. این تابع یکی از توابع کتابخانه ای است. کتابخانه ای که این تابع را در خود دارد iostream نام دارد.

در دستور بالا، رشته های متنی را در بین دو نماد « قرار میدهیم. هر چه که بین دو تا « قرار بگیرد به عنوان متن در نظر گرفته می شود. این رشته در خروجی نمایش داده شده است.

برای پیوست کردن یک کتابخانه به برنامه از دستور زیر در ابتدای فایل متنی ای که ایجاد کردیم می نویسیم. در مثال زیر یک کتابخانه iostream جهت بهره برداری از توابع دریافت اطلاعات و نمایش اطلاعات مورد استفاده قرار می گیرد.

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

دستور using namespace std; به compiler اطلاع میدهد که اجازه دارد از همه توابع موجود در این کتابخانه استفاده کند. جهت کاهش حجم برنامه می توانید بخشی این کتابخانه را با ذکر کردن نام تابع مد نظر خودتان استفاده کنید. یعنی به جای این دستور using std::function_name مورد نوشته خواهد شد. هماهنند using std::cout .

comment ۷.۳

برنامه نویسان علاقه دارند در مورد بخش هایی از کد توضیحاتی ایجاد کنند. این توضیحات در روند اجرایی برنامه نباید قرار بگیرند. به این توضیحاتی که برنامه نویسان جهت اطلاع از الگوریتم کد مطرح میکنند و در روند اجرای کد قرار نمیگیرند و اجرا نمی شوند کامنت یا comment می گویند. کامپایلر کامنت ها را به صورت رشته هایی یک خطی یا چند خطی میبیند. و کامپایلر آن را کامپایل نمی‌کند و این کار با قرار دادن علامت // در ابتدای جمله ی یک خطی ایجاد می شود.

اگر خواستیم چندین خط را کامنت کنیم و آنقدر تعداد این خط ها زیاد شد که ما نتوانیم علامت // را بگذاریم می توانیم ازعلامت /* و*/ استفاده کنیم :

```
//.......single line comment..........
/* ......multi line comment..........
.................*/
```

تعریف متغیرها و انواع داده ای ۸.۳

برای نگهداری اطلاعات در حافظه از متغیر یا variable استفاده می شود. قبل از استفاده از یک متغیر اول باید متغیر را تعریف کنیم. در تعریف یک متغیر نوع داده ای و نام متغیر تعیین میگردد. در واقع، متغیر یک منطقه ی نام گذاری شده از حافظه برای نگهداری مقادیر تکی (عدد یا کاراکتر) می‌باشد.

در زبان C++ انواع داده ای زیر وجود دارد.

- int ⇐ نگهداری مقادیر عددی صحیح
- float ⇐ نگهداری اعداد اعشاری
- double ⇐ نگهداری اعداد اعشاری با دقت بیشتر
- char ⇐ نگهداری کاراکتر

جدول ۳.۳: انواع داده ای در زبان C و C++ .

نوع داده	توضیح	اندازه (بیت)	محدوده
char	کاراکتر	۸	از ۱۲۸- تا ۱۲۷
int	عدد صحیح	۱۶	از ۳۲۷۶۸- تا ۳۲۷۶۷
float	عدد اعشاری	۳۲	از ۳.۴ e-۳۸ تا ۳.۴ e+۳۸
double	عدد اعشاری با دقت مضاعف	۶۴	از ۱.۷ e-۳۰۸ تا ۱.۷ e+۳۰۸

اندازه int در محیط های ۱۶ بیتی مانند DOS برابر ۱۶ بیت است. اما در محیط های ۳۲ بیتی همانند Windows اندازه آن ۳۲ بیت می باشد که در اینصورت محدوده ای برابر ۲،۱۴۷،۴۸۳،۶۴۸- تا ۲،۱۴۷،۴۸۳،۶۴۷+ را پوشش می دهد.

در بعضی از کامپایلرهای C ، نوع داده bool نیز وجود دارد که می تواند یکی از مقادیر true (درست) یا false (غلط) را نشان دهد. اما در نسخه های اولیه C از همان نوع داده صحیح یا int برای اینکار استفاده می شد. بدین صورت که ۰ نشان دهنده false و هر عدد غیر صفر (معمولا ۱) نشان دهنده true است.

از آنجا که برای ذخیره سازی کاراکترها کد اسکی آنها ذخیره می گردد، از یک متغیر کاراکتری یا char می توان بعنوان یک عدد صحیح کوچک نیز استفاده کرد و اعمال ریاضی برروی آنها نیز مجاز است.

علاوه براین چندین اصلاح کننده نیز وجود دارد که به ما اجازه می دهد نوع داده مورد نظر را با دقت بیشتری برای نیازهای مختلف استفاده نماییم. این اصلاح کننده ها عبارتند از: short, long, signed, unsigned.

تمام این اصلاح کننده ها می توانند به نوع داده int اعمال شوند و اثر آنها بستگی به محیط دارد. در یک محیط ۱۶ بیتی، short int بازهم برابر ۱۶ بیت است ولی long int برابر ۳۲بیت می باشد. همچنین، unsigned int باعث می شود که یک عدد ۱۶ بیتی بدون علامت داشته باشیم که بازه بین ۰ تا ۶۵۵۳۵ را پوشش می دهد. signed int نیز همانند int معمولی بوده و تفاوتی ندارد.

ترکیب این اصلاح کننده ها نیز ممکن است. مثلا unsigned long int یک عدد ۳۲ بیتی بدون علامت است که بازه ۰ تا ۴۲۹۴۹۶۷۲۹۵ را پوشش می دهد.

برروی نوع داده char فقط اصلاح کننده های signed و unsigned را می توان اعمال کرد. معمولا از اصلاح کننده unsigned وقتی استفاده می شود که قصد داشته باشیم از char بعنوان یک عدد صحیح مثبت بین ۰ تا ۲۵۵ استفاده کنیم.

بر روی نوع داده double تنها اصلاح کننده long قابل اعمال است و دراینصورت عدد اعشاری با ۸۰ بیت را خواهیم داشت که قادر است هر عددی در بازه
۳.۴ e-۴۹۳۲ تا ۱.۱ e+۴۹۳۲ را در خود نگاه دارد.

۹.۳ تعریف متغیرها

در زبان C++ برای تعریف متغیر از دستور زیر استفاده می شود. <type> <variable-list>; که type یکی از نوع داده های گفته شده و variable-list لیستی از متغیرها است که با کاما از یکدیگر جدا شده اند. بعنوان مثال :

```
int main(){
        int    sum;   // Datatype nameofVariables
        float  average;
        long int   a, b, c ;
        unsigned long int  i, j, k ;
```

```
        i=1;        // Assignment
        int d = 0;
}
```

در دستور بالا، یک متغیر از نوع short به اندازه دو بایت (با فرض اینکه بیت سمت چپ برای علامت است) از ۳۲۷۶۸- تا ۳۲۷۶۸ را می تواند بشمارد.

نکته: هر { و } یک block از برنامه می‌باشد که باید دستورات را درون آن بنویسیم.

نکته: تعریف متغیرها طبق اصول زبان C می تواند درهرجایی از برنامه صورت پذیرد، و متغیرهای تعریف شده از همان خط به بعد قابل استفاده خواهد بود. اما معمولا توصیه می گردد که تعریف متغیرها در همان خطوط ابتدایی صورت پذیرد.

نکته: زبان C به متغیرها مقدار اولیه نمي دهد (حتي ۰) و برنامه نویس خود باید اینکار را صریحا انجام دهد، درغیر اینصورت مقدار اولیه متغیر، نامعین خواهد بود.

نکته: به اولین مقداری که به یک متغیر منتسب می‌شود مقدار دهی اولیه آن متغیر (initialization) می‌گویند.

۱۰.۳ ثابت ها

ثابت ها مقادیر ثابتی هستند که مقدار آنها در حین اجرای برنامه تغییر نمی یابد. ثابت ها می توانند از هریک از نوع داده های اصلی باشند.

- ثوابت عددی صحیح: برای نمایش این دسته از ثوابت، از دنباله ای از ارقام بعلاوه علامت + یا - استفاده می کنیم. بعنوان مثال ۴۵- و یا ۳۴۸۹ ثوابت صحیح هستند.

  در حالت عادی C هر عدد را در کوچکترین نوع داده ای که می تواند قرار می دهد. مثلا عدد ۸۵ در یک int قرار می گیرد، اما عدد ۱۴۵۳۹۸ در یک long int قرار خواهد گرفت.

  اگر قصد دارید یک عدد کوچک بعنوان long محسوب گردد، می توانید از پسوند L در انتهای آن استفاده کنید. مثلا ۲۴۵L یک عدد long محسوب می شود. همچنین پسوند U در انتهای عدد نیز نشانه بدون علامت بودن آن است.

```
long int a = 20L;
```

درصورت لزوم ثوابت صحیح خود را در مبنای ۸ یا ۱۶ نیز که از مبناهای متداول در برنامه نویسی هستند، نیز می توانید بنویسید. برای نوشتن عدد در مبنای ۸ باید آن را با ۰ آغاز کنید، مثلا ۰۳۴۲ یک عدد در منای ۸ محسوب می گردد. همچنین برای نوشتن یک عدد در مبنای ۱۶ باید آن را با 0x آغاز نمایید، مانند 0x27A4 .

- ثوابت عددی اعشاری: برای نمایش اعداد اعشاری، باید از نقطه اعشار استفاده کنیم، مانند ۴۵.۲۳۷

  اما نکته جالب آنستکه می توانید از نماد علمی نیز برای نمایش اعداد اعشاری استفاده کنید. برای اینکار کافی است از حرف e برای نمایش قسمت توان استفاده نمایید. بعنوان مثال:

$$-23.47 \times 105 = -23.47e5$$

$$42.389 \times 10 - 3 = 42.389e - 3$$

  دقت کنید که قسمت توان، حتما یک عدد صحیح است.

  نکته جالب اینجاست که برخلاف مورد قبل، کامپایلر بطور پیش فرض داده های اعشاری را از نوع double فرض می کند.چنانچه دوست دارید ثابت شما ازنوع float درنظر گرفته شود، در انتهای آن F قرار دهید. ضمنا پسوند L نیز داده اعشاری را از نوع long double درنظر می گیرد.

- ثوابت کاراکتری: برای نشان دادن ثوابت کاراکتری، آنها را در داخل ' قرار می دهیم. بعنوان مثال 'A' یک ثابت کاراکتری است.

```
char ch = 'S';
```

دقت کنید که همانطور که قبلا گفته شد، کد اسکی کاراکترها در متغیر ذخیره می گردد، بنابراین می توان از آن بعنوان یک عدد صحیح نیز استفاده کرد. نکته مهم دیگر آنستکه به تفاوت عدد ۵ و کاراکتر '5' دقت داشته باشید. در حقیقت '5' برابر است با عدد ۵۳ که همان کد اسکی آن است.

روش فوق، برای نمایش کاراکترهای قابل چاپ مناسب است، اما بعضی کاراکترها مانند enter قابل نمایش نبوده و برای آنها شکل خاصی وجود ندارد. در چنین مواردی، زبان C از ترکیب علامت \ به همراه یک کاراکتر دیگر، برای نمایش این قبیل کاراکترها استفاده می نماید. بعنوان مثال، کاراکتر \n نشان دهنده خط جدید یا همان enter می باشد. جدول ۴.۳ لیست ای از کاراکترهای ترکیبی و معادل آنها را نشان می دهد.

جدول ۴.۳: کاراکترهای ثابت در زبان C و C++ .

نام کاراکتر	نحوه نمایش در C	کد اسکی
صدای بوق کامپیوتر	\a	۷
حرکت به عقب backspace	\b	۸
شروع صفحه form feed	\f	۱۲
سطر جدید line feed	\n	۱۰
برگشت به ابتداي سطر carriage return	\r	۱۳
فاصله افقی horizontal tab	\t	۹
فاصله عمودی vertical tab	\v	۱۱
علامت سوال	\?	۶۳
علامت '	\'	۳۹
علامت "	\"	۳۴
علامت \	\\	۹۲
علامت تهی	\0	۰
کد اسکی یک کاراکتر با ۳ رقم در مبنای ۸	\000	
کد اسکی یک کاراکتر با ۲ رقم در مبنای ۱۶	\x00	

- ثوابت رشته ای: رشته، دنباله ای از کاراکترها است که در داخل " قرار می گیرند.
به عنوان مثال "this is a test" یک رشته است. دقت کنید که 'a' یک کاراکتر است، اما "a" یک رشته است که فقط شامل یک کاراکتر می باشد.

۱۱.۳ عملگرها

عملگر، نمادی است که به کامپایلر می گوید تا عملیات محاسباتی یا منطقی خاصی را برروی یک یا چند عملوند، انجام دهد.

به عملگرهایی که فقط یک عملوند دارند، عملگر یکانی می گوییم و همواره عملگر در سمت چپ عملوند قرار می گیرد(مانند عدد ۱۲۵-).

اما عملگرهایی که برروی دو عملوند اثر می کنند را عملگر دودویی نامیده و عملگر را بین دو عملوند قرار می دهیم (مانند ۸۶+۲۳).

هر ترکیب درستی از عملگرها و عملوندها را یک عبارت می نامیم.

۱.۱۱.۳ عملگرهای محاسباتی

این عملگرها، همان اعمال متداول ریاضی هستند که در زبان C مورد استفاده قرار می گیرند. لیست کامل این عملگرها در جدول ۵.۳ آورده شده است.

این عملگرها بر روی همه انواع داده های C عمل می کنند، بجز عملگر % که فقط برروی نوع داده های صحیح عمل می کند.

اما نکته مهمی که باید به آن اشاره کرد، نحوه کار عملگر تقسیم برروی نوع داده های مختلف است. درصورتیکه هردو عملوند صحیح باشند، تقسیم بصورت صحیح بر صحیح انجام خواهد شد. اگر یکی یا هر دو عملوند اعشاری باشند، تقسیم بصورت اعشاری انجام خواهد پذیرفت.

جدول ۵.۳: عملگرهای محاسباتی در زبان C و C++ .

عمل	نوع	عملگر
منفی کردن عملوند سمت راست	یکانی	-
جمع دو عملوند	دودویی	+
تفریق دو عملوند	دودویی	-
ضرب دو عملوند	دودویی	*
تقسیم دو عملوند	دودویی	/
محاسبه باقیمانده تقسیم دو عملوند	دودویی	%

```
int main(){
        int  a,b ;
        float  c,d ;
        a = 10 ;   b = 4 ;
        c = 8.2;   d = 4.0;
        b = a / b;    //=>   2
        c = c / d;    //=>   2.05
        a = a / d;    //=>   2
        c = a / 4;    //=>2
}
```

چنانچه به آخرین مورد توجه کنید، متوجه می شوید که از آنجا که a یک متغیر صحیح است و ۴ نیز یک ثابت صحیح در نظر گرفته شده، درنتیجه تقسیم بصورت صحیح بر صحیح انجام گرفته است. چنانچه بخواهیم تقسیم بصورت اعشاری صورت پذیرد، به دو شکل می توانیم عمل کنیم.

- این عبارت را بصورت a / 4.0 بنویسیم.
- از عملگر قالب ریزی استفاده کنیم.

عملگر قالب ریزی می تواند یک نوع را به نوع دیگری تبدیل نماید. شکل کلی آن به شکل

(<type>) <expression>

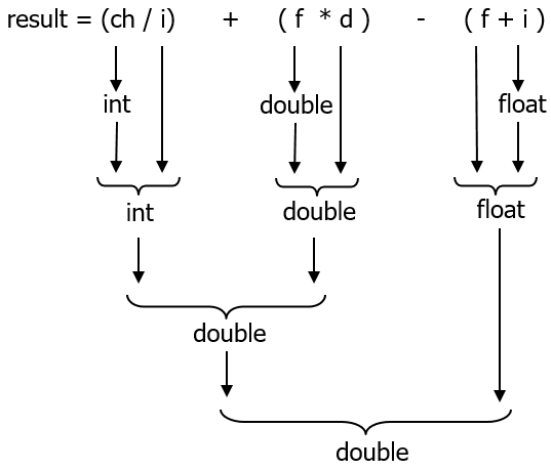
است که type نوع مورد نظر است که قصد تبدیل عبارت expression به آن نوع را داریم. بعنوان مثال در مورد قبلی می توان به شکل

(float) a / 4

عمل کرد. در این حالت از کامپایلر خواسته ایم که ابتدا عدد a را به اعشاری تبدیل کند (البته بصورت موقت) و سپس آن را بر ۴ تقسیم نماید، که مسلما حاصل اعشاری خواهد بود.

بطورکلی هرگاه ثابتها و متغیرهایی از انواع مختلف در یك عبارت محاسباتی داشته باشیم، کامپایلر C همه آنها را به یك نوع یکسان که همان بزرگترین عملوند موجود است تبدیل خواهد کرد. بعنوان مثال به کد زیر و ترتیب تبدیل اجرایی آن که در شکل ۳.۳ ارائه شده است توجه کنید.

```
int main(){
        char ch='a';
        int i=10;
        float f = 12.6;
        double d = 43.2;
        double result = (ch/i)+(f *d)-(f+i);
}
```



شکل ۳.۳: نمونه مثالی از ترتیب تبدیل و اجرای عبارت محاسباتی

اولویت عملگرها: در عبارتی که شامل چندین عملگر است، کدامیک در ابتدا اعمال خواهد گردید. اولویت عملگرهای محاسباتی از بالا به پایین بشرح زیر است:

۱. عملگر یکانی -

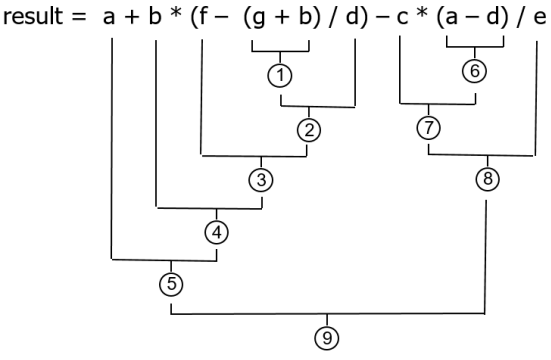
۲. عملگرهای * و / و %

۳. عملگرهای + و -

نکته: چنانچه اولویت دو عملگر یکسان باشد، این عملگرها از چپ به راست محاسبه خواهند شد.

نکته: چنانچه بخواهیم یک عمل با اولویت پایین زودتر انجام شود، باید از پرانتز استفاده کنیم. بنابراین اولویت عملگر پرانتز از همه موارد فوق بیشتراست. در مورد پرانتزهای متداخل، ابتدا پرانتز داخلی محاسبه می شود؛ اما در مورد پرانتزهای هم سطح، ابتدا پرانتز سمت چپتر محاسبه می گردد.

مثال: برای عبارت $result = a + b * (f - (g + b)/d) - c * (a - d)/e$ اولویت های اجرایی عملگرها را مشخص کنید. جواب در شکل ۴.۳ نشان داده شده است.



شکل ۴.۳: مثالی از اولویت عملگرها برای عبارت $result = a + b * (f - (g + b)/d) - c * (a - d)/e$

۲.۱۱.۳ عملگر انتساب

در زبان C برای انتساب چندین عملگر وجود دارد. ساده ترین عملگر انتساب، همان عملگر = است که در بسیاری از زبانها استفاده می شود. بعنوان مثال :

```
a = 5;
b = c + 2 * d;
```

این عملگر باعث می شود که عبارت سمت راست در عبارت سمت چپ قرار گیرد.
نکته: توجه کنید که مقدار سمت چپ باید عبارتی باشد که بتوان به آن یک مقدار را نسبت داد (مانند یک متغیر) که به آن Lvalue گفته می شود، بنابراین یک ثابت نمی تواند در سمت چپ قرار گیرد.
نکته: نکته دیگر اینکه اولویت عملگر = از عملگرهای ریاضی کمتر است و درنتیجه ابتدا آن عملیات انجام شده و در پایان حاصل در عبارت سمت چپ ریخته می شود.
نکته: لازم به ذکر است که در هنگام انتساب، درصورت لزوم نوع عبارت سمت راست به نوع عبارت سمت چپ تبدیل می شود. مثال:

```
int   a;
a = 2.5 * 5.0;
```

که دراینصورت عدد ۱۲ در a ذخیره خواهد شد.
شرکت پذیری این عملگر از راست به چپ می باشد، بدین معنا که چنانچه چندین عملگر نسبت دهی داشته باشیم، این عملگرها از راست به چپ محاسبه می شوند. مثلا پس از اجرای دستور زیر، مقدار هر ۳ متغیر برابر ۱۰ خواهد شد.

```
int a,b,c;
a = b = c = 10;
```

نکته: نکته جالب درمورد زبان C آنستکه دارای یک سری عملگرهای انتساب خلاصه شده است که باعث می شوند در بعضی موارد بتوانیم عبارات کوتاهتری را بنویسیم. این عملگرها در جدول ۶.۳ لیست شده اند. عملگرهای اول و دوم، عملگرهای یکانی هستند که بترتیب عملگر افزایش و عملگر کاهش نامیده می شوند.

جدول ۶.۳: عملگرهای انتسابی اختصاری در زبان C و C++ .

عملگرد	مثال	عبارت انتساب معادل
++	a++;	a = a + 1;
- -	a- -;	a = a - 1;
+=	a+=5;	a = a + 5;
-=	a -= 8;	a= a - 8;
*=	a *= 10;	a= a * 10;
/=	a /= 2;	a = a / 2;
%=	a %= 10;	a = a % 10;

۳.۱۱.۳ عملگرهای مقایسه ای

این عملگرها دو عبارت را بایکدیگر مقایسه کرده و نتیجه را باز می گردانند. نتیجه می تواند درست (true) یا غلط (false) باشد.
نتیجه این عملگرها یک عدد صحیح است که درصورت درست بودن ۱ و درصورت غلط بودن ۰ باز می گردانند. این عملگرها در جدول ۷.۳ ارائه شده اند.
نکته: نکته مهمی که باید به آن دقت کرد عملگر مساوی (==) است، چرا که یک اشتباه بسیار متداول برنامه نویسان C استفاده اشتباه از عملگر انتساب (=) بجای عملگر تساوی (==) است که باعث ایجاد خطا در برنامه می شود.
اولویت این عملگرها از بالا به پایین بشرح زیر است:

جدول ۷.۳: عملگرهای مقایسه ای در زبان C و C++ .

عملگر	مفهوم عملگر	مثال
>	بزرگتر (>)	a > b
<	کوچکتر (<)	a < b
>=	بزرگتر یا مساوی (≥)	a >= b
<=	کوچکتر یا مساوی (≤)	a <= b
==	مساوی (=)	a == b
!=	نامساوی (≠)	a != b

- عملگرهای <، > ، =< و =>

نکته: لازم بذکر است که در هنگام مساوی بودن اولویت چند عملگر، این عملگرها از چپ به راست محاسبه می گردند.

۴.۱۱.۳ عملگرهای منطقی

این عملگرها به شما اجازه می دهند که شرطهای ساده ای را که با استفاده از عملگرهای مقایسه ای ایجاد شده اند را با یکدیگر ترکیب نموده و شرطهای پیچیده تری را بسازید. این عملگرها در جدول ۸.۳ ارائه شده اند.

جدول ۸.۳: عملگرهای مجاز منطقی در زبان C و C++ .

نام	نام انگلیسی	نماد	شرح
قرینه	not	!	ارزش منطقی عملوند خود را معکوس می‌کند.
یا	or	||	این عملگر به دو عملوند نیاز دارد دارد. خروجی این عملگر زمانی دارای ارزش منطقی false خواهد بود که هر دو عملوند آن دارای ارزش false باشند. در غیر این صورت خروجی دارای ارزش منطقی true است.
و	and	&&	این عملگر دارای دو عملوند است و مقدار بازگشتی از این عملگر در صورتی دارای ارزش true خواهد بود که، ارزش منطقی هر دو عملوند آن true باشد. در غیر این صورت ارزش منطقی مقدار بازگشتی false خواهد بود.

۵.۱۱.۳ عملگرهای بیتی

عملگرهای بیتی به شما اجازه می دهند که شکل باینری انواع داده ها را دستکاری کنید. در جدول ۹.۳ عملگرهای مجاز دودویی لیست شده اند.

۱۲.۳ جریان کنترلی

به دنباله اجرای برنامه، جریان کنترلی برنامه گفته می شود. در ادامه جریان های کنترلی مختلف مانند جریان های شرطی و حلقه ای بیان خواهد شد.

۱.۱۲.۳ تابع cout

تابع cout از کتابخانه iostream ، برای نمایش اطلاعات از کاربر می تواند مورد استفاده قرار بگیرد.

مثال: برنامه ای بنویسید که مقدار یک متغیر را چاپ کند.

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

جدول ۹.۳: عملگرهای مجاز بیتی در زبان C و C++ .

نام	نام انگلیسی	نماد	شرح
قرینه	not	!	عملگر قرینه، یک عملوند از نوع عدد دارد و بیت های عملوند خود را معکوس میکند. صفرها را یک و یک ها را صفر خواهد کرد.
یا	or	|	این عملگر به دو عملوند با تعداد بیت های مساوی و از نوع عدد نیاز دارد. هر یک از بیت های دو عملوند را با همدیگر یای منطقی می کند. یعنی نتیجه همواره یک خواهد بود مگر وقتی که هر دو بیت مد نظر از دو عملوند صفر باشند.
و	and	&	این عملگر به دو عملوند با تعداد بیت های مساوی و از نوع عدد نیاز دارد. هر یک از بیت های دو عملوند را با همدیگر وی منطقی می کند. یعنی نتیجه همواره صفر خواهد بود مگر وقتی که هر دو بیت مد نظر از دو عملوند یک باشند.
	xor	^	این عملگر به دو عملوند با تعداد بیت های مساوی و از نوع عدد نیاز دارد. در صورتی که عملوندهای دو طرف این عملگر هر دو صفر یا هر دو ی؛ باشند، نتیجه صفر و در غیر اینصورت نتیجه یک می شود.

```
int main(){
        int i; /*define a variable with int
        datatype.*/
        i=10; /*assinemebt the variable with the
        value of 10.*/
        cout<<i;  /*show the variable value in
        output. */
        return 0;// finishe the main function
}
```

روش های مختلف برای چاپ چند عبارت:

```
cout<<"salam"<<"bye";
cout<<"salam"<<endl<<"bye";
cout<<"salam\n"<<"bye";
```

در دستور دوم و سوم کلمه ی bye در خط بعد چاپ می‌شود. endl و \n برای رفتن به خط بعد مورد استفاده قرار میگیرد.

```
cout<<"sal\nam";
```

نمایش می‌دهد:

```
sal
am
```

۲.۱۲.۳ تابع cin

تابع cin از کتابخانه iostream ، برای دریافت اطلاعات از کاربر می تواند مورد استفاده قرار بگیرد. در کد زیر متغیر a تعریف شده است و مقدار اولیه آن را از کاربر دریافت خواهد کرد.

```
#include<iostream>
using namespace std;
int main() {
        int a;
        cout << "please enter a number:";
        cin >> a;
        return 0;
}
```

در دستور بالا مقداری که از ورودی دریافت می کند در متغیر a قرار می‌دهد.

۳.۱۲.۳ دستورات شرطی

```
if(condition){/*code block*/}
```

اگر بخواهیم عبارت "Hello world!" را پنج بار بنویسیم میتوانیم از شرط (if) استفاده کنیم. البته برای ایجاد حلقه که پنج بار تکرار شود، لازم است تا دستورات ایجاد کننده حلقه استفاده کنیم. برای ایجاد حلقه از دستور goto استفاده شده است. دستور goto یک دستور پرش است به محلی که باید نام مشخص می شود. در کد زیر نمونه از از طریقه کاربرد آن نشان داده شده است.

```
#include <iostream>
using namespace std ;
int main(){
        int i = 1;
        l:cout <<"Hello world!\n";
        i = i+1;
        if(i <= 5){goto l;}
}
```

اگر در استفاده از شرط (if) یک دستور داشتیم می‌توانیم {} را حذف کنیم.
از دستور goto بهتر است استفاده نکنید. چون ساختار برنامه نویسی را از بین میبرد و همچنین خوانایی کد را کاهش میدهد.

۴.۱۲.۳ شرط ها

شرط ها وضعیت ها کنترل برنامه را تغییر خواهند داد. هر شرط دستوری منطقی یا مقایسه ای بر روی اعداد یا کاراکترها می باشد. در جدول ۱۰.۳ عملگرهای مقایسه ای لیست شده اند. همچنین برای ادغام منطقی چندین شرط میتوانید

جدول ۱۰.۳: عملگرهای مجاز برای مقایسه در زبان C و C++ .

عملگر	مفهوم	عملگر	مفهوم	عملگر	مفهوم
==	مساوی	<=	کوچکتر یا مساوی	<	کوچکتر
=!	نامساوی	>=	بزرگتر یا مساوی	>	بزرگتر

با دستور || به معنای یای منطقی و همچنین && به معنای وی منطقی مورد استفاده قرار میگیرد. به عنوان نمونه در صورت وجود دو شرط در یک دستور if در زیر مشاهده می شود.

```
if (condition1 && condition2){/*code block*/}
else {/*code block*/}
```

همانطور که ملاحظه می شود، دستور if با دستور else به معنای در غیر این صورت ادامه خواهد یافت. در بالا ساختار کلی آن مشاهده می شود.

مثال: بررسی کنید در چه شرایط هایی ok نمایش داده خواهد شد؟

```
#include <iostream>
using namespace std;
int main(){
        int a=1, b=3;
        if ((a==1)&&(b==4||b==3))
                cout<<"ok";
        return 0;
}
```

مثال: برنامه ای بنویسید که یک عدد از ورودی بگیرد و اگر زوج بود چاپ کند «زوج» و اگر فرد بود چاپ کند «فرد» .

```
#include <iostream>
using namespace std;
int main(){
        int a;
        cin>>a;
        if (a%2==0)
                cout<<"even";
        else
                cout<<"odd";
        return 0;
}
```

نوع داده ای bool

نوع داده ای bool که دو مقدار true یا false را به خود اختصاص می دهد، برای نگهداری نتیجه گزاره های منطقی استفاده خواهد شد. از طرفی نوع داده ای bool در واقع از نوع عدد است و اگر برابر با مقدار false باشد یعنی مقداری برابر با صفر دارد و اگر عددی غیر صفر باشد در اینصورت به معنی true خواهد بود.

```
#include <iostream>
using namespace std;
int main(){
        int a=1;
        bool b;
        if (a==1) b = true;
        if(b){cout<<!b;} // output is 0;
        return 0;
}
```

عملگرهای مرتبط با عملیات دودویی در جدول ۸.۳ نمایش داده شده اند.

۵.۱۲.۳ دستور while

از این دستور برای اینکه تا زمانی که شرط مقداری برابر با true دارد، کاری را انجام دهد، استفاده می شود.

```
while(condition){/*code block*/}
```

مثال: برای ۵ بار نوشتن "hello world" می توان از این روش استفاده کرد.

```
#include <iostream>
using namespace std;
int main(){
        int i=0;
        while(i<5){
                cout<<"hello world"<<endl;
                i=i+1;
        }
}
```

۶.۱۲.۳ حلقه do-while

گاهی لازم می شود، برخلاف دستور while ، اول دستورها اجرا و بعد شرط بررسی شود. این قابلیت را دستور do-while در اختیار ما قرار میدهد. ساختار کلی این دستور را در زیر مشاهده می کنید.

```
do{
        // code block
} while(condition);
```

مثال: برای ۵ بار نوشتن "hello world" می توان از این روش استفاده کرد.

```
#include <iostream>
using namespace std;
int main(){
        int i=0;
        do{
                cout<<"hello world"<<endl;
                i=i+1;
        }while(i<5);
}
```

۷.۱۲.۳ مقدار دهی شرطی

در صورتی که به صورت شرطی میخواهید مقداردهی به متغیری داشته باشید میتوانید از دستور زیر استفاده کنید.

```
int a = (condition) ? {b} : {c};
```

این دستور معادل دستور زیر است.

```
int a;
if(condition) {a = b;} else {a = c;}
```

۸.۱۲.۳ اضافه کننده و کم کننده ها

برای اضافه یا کم کردن از عملگر ++ یا -- میتوان استفاده می‌کنیم. این دو عملگر از یک واحد اضافه و کم می کنند. در زیر موارد استفاده ای از این عملگرها مشاهده می شود.

```
int i=10;
cout<<++i<<endl;
```

```
cout<<i++<<endl;
cout<<i<<endl;
i--;
cout<<i<<endl;
```

عملگر i++ و ++i از لحاظ نتیجه تقریبا یکسان هستند. تفاوت آنها در مقدار خروجی خواهد بود. ++i یک واحد به متغیر اضافه می کند و بعد مقدار را برای خروجی ارسال میکند ولی i++ در ابتدا مقدار را به خروجی ارسال می کند و بعد یک واحد به متغیر i اضافه می کند. خروجی حاصله از دستورات بالا را در زیر مشاهده می کنید.

```
11
11
12
11
```

۹.۱۲.۳ حلقه for

برای ایجاد حلقه میتوانید از دستور for نیز استفاده کنید. البته که حلقه for بیشتر از موارد دیگر مورد علاقه برنامه نویسان است چون در قسمت اول متغیرهایی لازم را تعریف میکنند، در قسمت دوم شرط را می نویسند و در در قسمت سوم عملگرهای مربوط به افزایش و کاهش را می نویسند.

```
for(statement 1; statement 2; statement 3){/*code block*/
    }
```

مثال: ۵ بار HelloWorld را با استفاده از حلقه for چاپ کنید.

```
#include<iostream>
using namespace std;
int main(){
        for(int i=0;i<5;i++){cout<<"HelloWorld"<<endl;}
}
```

مثال: برنامه‌ای بنویسید که تمام مضارب هفت از ۷ تا ۷۷ را چاپ کند.

```
#include<iostream>
using namespace std;
int main(){
        for(int i=7;i<=77;i+=7){cout<<i<<endl;}
}
```

مثال: تمام مضارب ۲ را از ۲۰ به ۲ چاپ کند.

```
#include<iostream>
using namespace std;
int main(){
        for(int i=20;i>=2;i-=2){cout<<i<<endl;}
}
```

مثال: تمام مضارب ۱۱ را از ۹۹ تا صفر چاپ کند.

```
#include<iostream>
using namespace std;
int main(){
```

```
        for(int i=99;i>=0;i-=11){cout<<i<<endl;}
}
```

کاری که دستور for انجام می‌دهد:

۱. مقدار اولیه

۲. ارزیابی شرط

۳. اجرای دستورات در صورت درست بودن شرط

۴. اجرای دستور سوم حلقه for

۵. بازگشت به ۲

۱۰.۱۲.۳ انتخاب مسیر با switch-case

گاهی بهتر است به جای استفاده از دستورات شرطی متوالی از دستور switch-case استفاده کنید. از لحاظ پیاده سازی خوانایی و اجرای بهتری خواهد داشت. در زیر ساختار کلی این دستور را مشاهده میکنید.

```
switch(expression) {
        case x:
        // code block
        break;
        case y:
        // code block
        break;
        default:
        // code block
}
```

مثال: برنامه ای بنویسید که مقداری را از ورودی بگیرد اگر زوج بود چاپ کند (even) و اگر فرد بود چاپ کند (odd).

```
#include<iostream>
using namespace std;
int main(){
        int a;
        cin>>a;
        if(a%2==0) {cout<<"even";}else{cout<<"odd";}
}
```

البته که میتوان این دستور را با switch-case ایجاد کرد. در زیر می توان دستورات معدل دستورات بالا را مشاهده کرد. البته باید بیان کرد که switch-case در زبان C و C++ از نوع داده ای عدد را فقط قبول میکند. زبان های بروزتر مانند java و غیره، از باقی نوع های داده ای نیز می تواند استفاده شود.

```
#include<iostream>
using namespace std;
int main(){
        int a;
        cin>>a;
        switch(a%2)
        {
                case 0:
```

```
                cout<<"even";
                break;
        case 1:
                cout<<"odd";
                break;
    }
}
```

مثال: برنامه ای بنویسید که نمره دانشجویان را بگیرد و ارزیابی کند.

```
#include<iostream>
using namespace std;
int main(){
    int grade;
    cin>>grade;
    if(grade==20) cout<<"well";
    else if(grade==18) cout<<"good";
    else cout<<"unknown";
}
```

حال اگر بخواهیم همین برنامه را بدون switch-case انجام دهیم:

```
#include<iostream>
using namespace std;
int main(){
    int grade;
    cin>>grade;
    switch(grade)
    {
        case 20:
        cout<<"well";
        break;
        case 18:
        cout<<"good";
        break;
        default: cout<< "unkown";
    }
}
```

۱۱.۱۲.۳ دستورات کنترلی درون حلقه ها

از دستورات کنترلی زیر برای توقف یا ادامه حلقه می توان استفاده کرد.

- break ← متوقف کردن حلقه ← while; do-while; for; switch-case
- continue ← بازگشت به ابتدای حلقه ← for; while; do-while

مثال: برنامه‌ای بنویسید که اعداد زوج ۰ تا ۱۰۰ را چاپ کن.

```
#include<iostream>
using namespace std;
int main(){
```

```
        int i=-1;
        while(i<100){
                i++;
                if(i%2!=0) continue;
                cout<<i;
        }
}
```

مثال: برنامه‌ای بنویسید که ۱۰۰۰ داده را از ورودی بگیرد و min آنها را چاپ کند.

```
#include<iostream>
using namespace std;
int main() {
        int i = 0, a,min;
        cout << "Please enter the first number:";
        cin >> a;
        min = a;
        for (i = 1; i < 1000; i++) {
                cout << "Please enter "<<(i+1)<<"th
                    number:";
                cin >> a;
                if (min > a) min = a;
        }
        cout << "the min number is:" << min;
        return 0;
}
```

مثال: برنامه‌ای بنویسید که سه ضلع یک مثلث را از ورودی بگیرد و بگوید قائم‌الزاویه هست یا خیر.

```
#include<iostream>
using namespace std;
int main(){
        int a,b,c;
        cin>>a>>b>>c;
        if(a*a==(b*b+c*c)) cout<<"yes";
        else if(b*b==(a*a+c*c)) cout<<"yes";
        else if(c*c==(a*a+b*b)) cout<<"yes";
        else cout<<"no";
        return 0;
}
```

مثال: برنامه‌ای بنویسید که مقسوم‌علیه‌های طبیعی عدد n را چاپ کند.

```
#include<iostream>
using namespace std;
int main(){
        int n;
        cin>>n;
        while(n<1){
                cout<<"please enter a natural number";
                cin>>n;
```

```
        }
        for(int i=1;i<=n;i++){
                if(n%i==0) cout<<i<<endl;
        }
        return 0;
}
```

مثال: برنامه‌ای بنویسید که فاکتوریل عدد طبیعی n را چاپ کند.

```
#include<iostream>
using namespace std;
int main(){
        int n,fact=1;
        cin>>n;
        for(int i=1;i<=n;i++)
                fact=fact*i;
        cout<<fact;
        return 0;
}
```

مثال: برنامه‌ای بنویسد که تعیین کند عدد طبیعی n اول است یا خیر.

```
#include<iostream>
using namespace std;
int main(){
        int n;
        cin>>n;
        for(int i=2;i<=n/2;i++)
                if(n%i==0){
                        cout<<"noprime";
                        return 0;
                }
        cout<<"prime";
        return 0;
}
```

مثال: برنامه‌ای بنویسید که عدد n را خوانده و مجموع ارقام آن را نمایش دهد.

```
#include<iostream>
using namespace std;
int main(){
        int n,sum=0;
        cin>>n;
        while(n>0){
                sum=sum+(n%10);
                n=n/10;
        }
        cout<<sum;
        return 0;
}
```

مثال: برنامه ای بنویسید که مجموع عبارت e^x را تا هزار جمله چاپ کند.

$$e^x = 1 + x + \frac{x^2}{2!} + \dots$$

```
#include <iostream>
using namespace std;
int main(){
        int x;
        float a=1, sum=1;
        cout<<"enter the power"<<endl;
        cin>>x;
        for (int i = 1;i < 1000;i++){
                a = a * x / i;
                sum += a;
        }
        cout<<"the answer is:" <<sum;
        return 0;
}
```

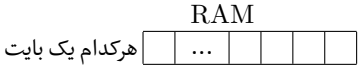
نکته: اگر در if ، i=۰ باشد i را برابر با صفرقرار می دهد ولی اگر i==۰ باشد این یک شرط محسوب می شود.
نکته: اینکه ناخواسته از یک برنامه بیرون بیاید میگویند Error Run-Time مثل : N/۰ (N تقسیم بر صفر)

مثال: برنامه ای بنویسید که دوعدد از کاربر دریافت کرده و ب.م.م (بزرگترین مقسوم علیه مشترک) آن دو عدد را در خروجی چاپ کند. (ب.م.م) = (g.c.d)

```
#include <iostream>
using namespace std;
int main(){
int n, m, min = 0, gcd = 1;
cout<<"enter two numbers"<<endl;
cin>>n>>m;
if (n>m)
        min = m;
else min = n;
for (int i = 1; i <= min; i++)
        if (n%i == 0)
                if (m%i == 0)
                                gcd = i;
cout << "g.c.d=" << gcd;
return 0;
}
```

۱۳.۳ آرایه ها

گروهی متوالی از خانه های حافظه که نوع همه ی آنها از نوع داده ای یکسان باشد، آرایه گفته می شود.
نکته: آرایه ای به طول ۱۰ از نوع int که هر خانه دارای ۴ بایت است، به اندازه ۴۰ بایت کنار هم حافظه نیاز دارد.($int\ b[10]$)



مثال: برنامه ای بنویسید که پنج عدد از ورودی بگیرد و بگوید کدام بیشتر از همه تکرار شده است.

```
#include <iostream>
using namespace std;
int main(){
        int a1, a2, a3, a4, a5;
        int c1=0, c2=0, c3=0, c4=0, c5=0;
        cout << "enter five numbers"<<endl;
        cin >> a1 >> a2 >> a3 >> a4 >> a5;
        if (a1 == a2){
                c1++;
                c2++;
        }
        if (a1 == a3){
                c1++;
                c3++;
        }
        if (a1 == a4){
                c1++;
                c4++;
        }
        if (a1 == a5){
                c1++;
                c5++;
        }
        if (a2 == a3){
                c2++;
                c3++;
        }
        if (a2 == a4){
                c2++;
                c4++;
        }
        if (a2 == a5){
                c2++;
                c5++;
        }
        if (a3 == a4){
                c3++;
                c4++;
        }
        if (a3 == a5){
                c3++;
                c5++;
        }
        if (a4 == a5){
                c4++;
                c5++;
        }
        if (c1 == c2 && c2 == c3 && c3 == c4 && c4 == c5)
```

```
                cout << "All are same";
        else
        if (c1 >= c2 && c1 >= c3 && c1 >= c4 && c1 >= c5)
                cout<<"the answer is:"<<a1;
        else
        if (c2 >= c1 && c2 >= c3 && c2 >= c4 && c2 >= c5)
                cout<<"the answer is:"<<a2;
        else
        if (c3 >= c1 && c3 >= c2 && c3 >= c4 && c3 >= c5)
                cout<<"the answer is:"<<a3;
        else
        if (c4 >= c1 && c4 >= c2 && c4 >= c3 && c4 >= c5)
                cout << "the answer is:" << a4;
        else
        if (c5 >= c1 && c5 >= c2 && c5 >= c3 && c5 >= c4)
                cout <<"the answer is:"<< a5;
        return 0;
}
```

۱.۱۳.۳ تعریف آرایه

نوع آرایه نام آرایه [سایز آرایه]

نکته: محل قرار گرفتن عناصر را index یا subscrip می نامند.
نکته: i اُمین عنصر آرایه، در index $i-1$ اُم قرار دارد.

$$\text{index } i\text{th element} = i - 1$$

نکته: دسترسی به داخل هریک از index ها با [index] نام آرایه امکان پذیر است. به نمونه زیر دقت کنید:

$c[6]$	$c[5]$	$c[4]$	$c[3]$	$c[2]$	$c[1]$	$c[0]$
↓	↓	↓	↓	↓	↓	↓
۳۲	۱۹	۱۷	۱۴	۵۱	۱۲	۱۰

در نمونه بالا فرض شده است یک آرایه از نوع عدد با نام c به اندازه ۷ داریم. هر خانه با یک مقداری، مقداردهی شده است. ایندکس های هر خانه از صفر شروع و با ششمین خانه خاتمه یافته است. قطعه کد زیر را برای تولید داده های بالا می توان داشت.

```
int main(){
        int c[7];
        c[0]=10; c[1]=12; c[2]=51; c[3]=14;
        c[4]=17; c[5] =19; c[6] = 32;
        int a=3;
        int b=1;
        c[a+b]=c[4]=17;
}
```

نکته: از آنجایی که طول آرایه به اندازه ۷ خانه از نوع عدد است، نمیتوان به خانه های غیر از صفرم تا ششم دسترسی داشت. در صورتی که به خانه بیشتر از ششم مانند هشتم دسترسی بخواهید داشته باشید با خطای زمان اجرا شاید برخورد کنیم. البته اگر زبان های دیگری به غیر از C یا C++ باشد حتما خطای اجرایی را خواهید داشت.

مثال: برنامه ای بنویسید که آرایه ای از ده عدد را از کاربر گرفته و چاپ کند.

```
#include <iostream>
using namespace std;
int main(){
        int c[10];
        for(int i=0;i<10;i++){
                cout<<"enter␣c[">>i<<"]"<<endl;
                cin>>c[i];
        }
        for(int i=0;i<10;i++)
                cout<<c[i]<<"␣␣␣";
        return 0;
}
```

مثال: برنامه ای بنویسید که آرایه ای ده تایی را از کاربر گرفته و مجموع اعداد را چاپ کند.

```
#include <iostream>
using namespace std;
int main(){
        int c[10];
        int sum=0;
        for(int i=0;i<10;i++){
                cout<<"enter␣c[">>i<<"]"<<endl;
                cin>>c[i];
                sum=sum + c[i];
        }
        cout<<"sum␣=␣"<<sum;
        return 0;
}
```

۲.۱۳.۳ مقدار دهی اولیه به آرایه ها

به اولین مقداردهی اولیه ای که به خانه های آرایه اختصاص داده می شود، مقداردهی اولیه به آرایه گفته می شود.
در زیر ساختار کلی تعریف یک آرایه با مقداردهی اولیه را مشاهده میکنید.

نوع آرایه نام آرایه [تعداد اعضای آرایه] = {مقدار اول ، مقدار دوم ، ... }

مثال: آرایه ای را تعریف و مقداردهی اولیه کنید.

```
int main(){
        int c[7] = {10,12,51,14,17,19,32};
}
```

برابر است با:

```
int main(){
        int c[7];
        c[0]=10; c[1]=12; c[2]=51; c[3]=14;
        c[4]=17; c[5]=19; c[6]=32;
}
```

نکته: در صورتی که اندازه داده های ورودی برابر با طول آرایه نباشد، باقی مقدارها را صفر در نظر می گیرد.
به نمونه زیر توجه کنید.

$int\ a[3] = \{1, 4\}$

برابر خواهد بود با یک آرایه با طول ۳ با دو مقدار اول یک و چهار، و مقدار سوم صفر. در زیر شکل حافظه را مشاهده می کنید.

1	4	0

در کد زیر آرایه ای از کاراکترها را مشاهده می کنید.

```
int main(){
        char c[]={'s','a','l','a','m','\0'};
        char s[]={"salam"};
}
```

۱۴.۳ متغیرهای ثابت

متغیرهای ثابت، متغیرهایی هستند که مقدار آنها ثابت است و تغییری نمی کند. همانطور که در کد زیر مشاهده می شود، کلمه const موجب می شود تا یک متغیر از نوع مقدار ثابت باشد.

```
int main(){
        const int array_size = 10;
        int myArray[array_size];
}
```

استفاده از متغیرهای ثابت موجب می شود تا از اعمال تغییرات بر روی اندازه ها راحتر و با دقت بیشتری باشد.
مثال: برنامه ای بنویسید که پنج عدد از ورودی بگیرد و بگوید کدام بیشتر از همه تکرار شده است. این مثال را یک بار بدون آرایه حل کردیم ولی اکنون با آرایه میخواهیم این برنامه را ایجاد کنیم.

```
#include <iostream>
using namespace std;
int main(){
        const int size=5;
        int a[size], c[size];
        for (int i=0; i<size; i++){}
                cin>>a[i];
                c[i] = 0;
        }
        for (int i=0; i<size; i++)
                for (int j=0; i<size; j++)
                        if(a[i]==a[j])
                                c[i]++;
        int index=0;
        int max=c[0];
        for (int i=1; i<size; i++)
                if (c[i]>max){
                        max=c[i];
                        index=i;
                }
        cout<<a[index]<<endl;
        cout<<index<<endl;
        cout<<c[index]<<endl;
```

```
        return 0;
}
```

۱۵.۳ کتابخانه iomanip

کتابخانه iomanip جهت زیبا سازی در صفحه نمایش مورد استفاده قرار می گیرد. به نمونه کد زیر توجه کنید.

```
#include<iostream>
#include<iopamip>
using std::setfill;
using std::setw;
using std::cout;
int main(){
        cout<<setw(10)<<endl;
        cout<<setw(10)<<"salam"<<endl
        cout<<setw(2)<<"1"<<endl;
        cout<<setw(2)<<"123"<<endl;
        cout<<setfill('a')<<setw(5)<<"bbb"<<endl;
}
```

خروجی:

```
␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣salam
␣1
123
aabbb
```

۱۶.۳ مرتب سازی حبابی

مرتب سازی حبابی یا bubble sort یکی از چندین الگوریتمی است که جهت مرتب سازی مورد استفاده قرار می گیرد.

نمونه عددی زیر را حتما دقت کنید. اجازه بدهید یک آرایه از عددهای (۸، ۲، ۴، ۵، ۱) اختیار کنیم و آن را به ترتیب صعودی با استفاده از مرتب‌سازی حبابی مرتب کنیم. در هر مرحله عناصری که در حال مقایسه شدن با یکدیگر هستند پر رنگ تر نشان داده شده‌اند:

گذر اول	گذر دوم	گذر سوم	گذر چهارم
(۸، ۲، ۴، ۵، ۱)	(۱، ۸، ۲، ۴، ۵)	(۱، ۲، ۸، ۴، ۵)	(۱، ۲، ۴، ۸، ۵)
↓	↓	↓	↓
(۸، ۲، ۴، ۱، ۵)	(۱، ۸، ۲، ۴، ۵)	(۱، ۲، ۸، ۴، ۵)	(۱، ۲، ۴، ۵، ۸)
↓	↓	↓	↓
(۸، ۲، ۱، ۴، ۵)	(۱، ۸، ۲، ۴، ۵)	(۱، ۲، ۴، ۸، ۵)	(۱، ۲، ۴، ۵، ۸)
↓	↓	↓	↓
(۸، ۱، ۲، ۴، ۵)	(۱، ۲، ۸، ۴، ۵)	(۱، ۲، ۴، ۸، ۵)	(۱، ۲، ۴، ۵، ۸)
↓	↓	↓	↓
(۱، ۸، ۲، ۴، ۵)	(۱، ۲، ۸، ۴، ۵)	(۱، ۲، ۴، ۸، ۵)	(۱، ۲، ۴، ۵، ۸)

در نهایت آرایه مرتب شده‌است و الگوریتم می‌تواند پایان پذیرد.
n-1مرحله ⇐ n-1امین عنصر ماکزیمم به انتهای لیست میرود.
برنامه ای که این کار را انجام می دهد به شکل زیر می باشد.

```
#include <iostream>
int main(){
        const int len=10;
        int a[len],i,j,temp;
        for(i=0;i<len;i++)
                std::cin>>a[i];
        for(i=len-2;i>=0;i--)
                for(j=0;j<=i;j++)
                        if(a[j]>a[j+1])
                        {
                                temp=a[j];
                                a[j]=a[j+1];
                                a[j+1]=temp;
                        }
        for(i=0;i<len;i++)
                std::cout<<a[i]<<'\t';
        return 0;
}
```

۱۷.۳ آرایه های دوبعدی

آرایه های دو بعدی مجموعه ای پشت سر هم از آرایه های یک بعدی هستند. نحوه تعریف آرایه به صورت زیر است.

datatype arrayname[rows][colums]

نحوه دسترسی به اعضای آرایه به صورت زیر خواهد بود.

arrayname[i][j] = 0

مثال: یک آرایه دو بعدی را بگیرد و ان را چاپ کند

```
#include <iostream>
using namespace std;
int main(){
        const int rowsize=3;
        const int colsize=4;
        int a[rowsize][colsize];
        for(int i=0; i<rowsize; i++)
                for(int j=0; j<colsize; j++)
                        cin>>a[i][j];
        for(int i=0; i<rowsize; i++){
                for(int j=0; j<colsize; j++)
                        cout<<a[i][j]<<'\t';
                cout<<endl;
        }
        return 0;
}
```

مثال: جمع دو ماتریس

```
#include <iostream>
using std::cout;
using std::endl;
#include<iomanip>
using std::setw;
int main(){
        int A[3][4];
        int B[3][4];
        int C[3][4];
        for(int i=0; i<3;i++)
                for(int j=0; j<4;j++)
                        cin>>A[i][j];
        for(int i=0; i<3;i++)
                for(int j=0; j<4;j++)
                        cin>>B[i][j];
        for(int i=0; i<3;i++)
                for(int j=0; j<4;j++)
                        C[i][j]=A[i][j]+B[i][j];
        for(int i=0; i<3; i++){
                for(int j=0; j<4; j++)
                        cout<<C[i][j]<<'\t';
                cout<<endl;
        }
        return 0;
}
```

۱.۱۷.۳ مقداردهی اولیه به آرایه دو بعدی

جهت مقداردهی اولیه همانند آرایه های یک بعدی عمل میکنیم. در زیر دو نمونه از مقداردهی های اولیه نشان داده شده است.

```
int A[2][2]={{1,2},{3,4}};
```

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
char A[3][3]={{'a','b','c'},{'x','y','z'},{'r','s','t'}};
```

$$A = \begin{bmatrix} a & b & c \\ x & y & z \\ r & s & t \end{bmatrix}$$

مثال: برنامه ای بنویسید که ماتریس $A_{2\times3}, B_{3\times4}$ را بگیرد و در نتیجه $C_{2\times4} = A \times B$ را چاپ کند.

```
#include<iostream>
using std::cout;
using std:: endl;
#include<iomanip>
using std::setw;
```

```
int main(){
        int A[2][3];
        int B[3][4];
        int C[2][4];
        for(int i=0; i<2;i++)
                for(int j=0; j<4;j++)
                        C[i][j]=0;
        for(int i=0; i<2;i++)
                for(int j=0; j<3;j++)
                        cin>>A[i][j];
        for(int i=0; i<3;i++)
                for(int j=0; j<4;j++)
                        cin>>B[i][j];
        for(int k=0; k<2;k++)
                for(int i=0; i<4;i++)
                        for(int j=0; j<3;j++)
                                C[k][i]+=A[k][j]*B[j][i];
        for(int i=0; i<2;i++){
                for(int j=0; j<4;j++)
                        cout<<setw(5)<<C[i][j];
                cout<<endl;
        }
        return 0;
}
```

مثال: یک آرایه به طول ۱۰ بگیرد و آن را برعکس کند.

```
#include <iostream>
using namespace std;
int main(){
        const int size = 10;
        int A[size];
        int temp;
        for(int i = 0; i < size; i++)
                cin>>A[i];
        for(int i = 0; i < size/2; i++){
                temp = A[size-i-1];
                A[size-i-1] = A[i];
                A[i] = temp;
        }
        return 0;
}
```

۱۸.۳ مرتب سازی انتخابی

مرتب سازی انتخابی (selection sort) یکی از چندین روش مرتب سازی است که در ادامه روند اجرای آن و کد آن شرح داده خواهد شد.

۲	۱۰	۴	۵۶	۳۴۱
۳۴	۱۰	۴	۵۶	۱۲
۳۴	۱۰	۵۶	۱۴	۲
۳۴	۵۶	۱۱۰	۴	۲
۵۶	۱۳۴	۱۰	۴	۲
۱۵۶	۳۴	۱۰	۴	۲

```
#include <iostream>
using namespace std;
int main(){
        const int Arraysize = 10;
        int min, temp;
        int A[Arraysize];
        for(int i = 0; i < Arraysize; i++)
                cin>>A[i];
        for(int i = 0; i < Arraysize-1; i++){
                min = i;
                for(int j = i+1; j < Arraysize; j++)
                        if(A[j]<A[i])
                                min = j;
                temp = A[min];
                A[min] = A[i];
                A[i] = temp;
        }
        for(int i = 0; i < Arraysize; i++)
                cout<<A[i]<<'\t';
        return 0;
}
```

۱۹.۳ طریقه آدرس دهی درون آرایه ها

آدرس یک خانه آرایه در حافظه توسط نام آرایه نگهداری می شود. در ارایه های یک بعدی، کامپایلر آدرس شروع آرایه و نوع عناصر را می داند و نام آن را (base Address) در نظر می گیرد.

A[i] آدرس = base Address + (i*element size)

برای آرایه دو بعدی:

A[Rows][Cols]
A[i][j] آدرس = base Address + ((i*Cols+j)*element size)

در آرایه دو بعدی برای استفاده از این روش فقط در c++ از Rows استفاده نشده است.
برای آرایه سه بعدی:

A[Rows][Cols][Hi]
A[i][j][k] آدرس = base Address + ((i*Cols+j*Hi+k)*element size)

به همین صورت برای آرایه های با ابعاد بیشتر نیز ادامه پیدا می کند.

۲۰.۳ مرتب سازی درجی

مرتب سازی درجی (Insertion sort)، یکی از چندین روش مرتب سازی است که در ادامه روند اجرای آن و کد آن شرح داده خواهد شد.

۰	۲	۱۰	۴	۵۶	| ۳۴
۰	۲	۱۰	۴	| ۵۶	۳۴
۰	۲	۱۰	| ۵۶	۳۴	۴
۰	۲	| ۵۶	۳۴	۱۰	۴
۰	| ۵۶	۳۴	۱۰	۴	۲
۵۶	۳۴	۱۰	۴	۲	۰

```
#include <iostream>
using namespace std;
int main() {
        const int arraysize = 5;
        int j, insert;
        int A[arraysize];
        bool change = false;
        for (int i = 0; i < arraysize; i++)
        cin >> A[i];
        for (int i = 1; i < arraysize; i++) {
                insert = A[i];
                j = i-1;
                change = false;
                while (j >= 0 && insert < A[j] ) {
                        A[j+1] = A[j];
                        j--;
                        change = true;
                }
                if (change) {
                        j++;
                        A[j] = insert;
                }
        }
        for (int i = 0; i < arraysize; i++)
        cout << A[i] << '\t';
        return 0;
}
```

۲۱.۳ تبدیل نوع

در محاسبات با اعداد لازم است به نوع اعداد توجه شود. به عنوان نمونه به کد زیر توجه کنید. در تقسیم عدد ۱۰ بر ۳ نتیجه ۳.۳۳ خواهد شد. ولی از آنجایی که متغیر a از نوع int است، عدد صحیح را در خود نگهداری می کند و لذا خروجی حاصله در برابر با ۳ خواهد بود. همچنین برای ضرب عدد ۱۰ در۲.۵، در ابتدا عدد ۱۰ به عدد اعشاری ۱۰.۰۰ تبدیل می شود و بعد در عدد ۲.۵ ضرب خواهد شد و نتیجه ۲۵.۰۰ حاصل خواهد شد. از آنجایی که عدد a یک نوع داده ای اعداد صحیح است، عدد ۲۵ را فقط در خود نگهداری خواهد کرد.

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

```
int main(){
        int a = 10/3; // a = 3
        cout << a;
        a = 10*2.5; //10.0*2.5=25.00
        cout << a; //a = 25
```

خروجی:

```
3
25
```

این تبدیل ها به صورت خودکار توسط کامپایلر انجام می شود ولی شما میتوانید کامپایلر را مجبور هم کنید. مثلا به کد زیر توجه کنید.

مثال: تمام کاراکتر های صفر تا ۲۵۵ را برای ما چاپ کند این گونه عمل می کنیم.

```
#include <iostream>
using namespace std;
int main(){
        for(unsigned char c = 0; c < 255; c++)
                cout <<(int)c <<'␣␣'<< c << endl;
        return 0;
}
```

enum ۲۲.۳

به مثال زیر توجه کنید.

مثال: مقطع دیپلم داشت بنویسید good و اگر فوق لیسانس داشت بنویسید well .

```
#include <iostream>
using namespace std;
int main(){
        int a;
        cout<<"Diploma=0,␣Master=1";
        cin>>a;
        if(a==0) cout << "good";
        if(a==1) cout << "well";
        else cout << "Incorrect␣number";
        return 0;
}
```

enum در خوانایی کد به ما کمک میکند. همانطور که در مثال بالا مطرح شد، درک اینکه یک کدام بود و صفر کدام بود برای ما سخت خواهد بود. ما با استفاده از تعریف نوع داده ای جدید به نام enum می توانیم این شماره گزاری ها را خودکار داشته باشیم. این نوع داده ای جدید که ما تعریف خواهیم کرد از نوع عدد اعداد ثابت خواهد بود. این کار با کلمه کلیدی enum انجام خواهد شد.

ساختار کلی تعریف enum به صورت زیر بیان میگردد.

enum نام { نام های مختلف مدنظر که با ویرگول از هم جدا شده اند. }

```
enum week {Sunday, Monday, Tuesday,
                Wednesday, Thursday, Friday, Saturday};
```

```
enum boolean { false, true };
enum season {
        spring = 0,
        summer = 4,
        autumn = 8,
        winter = 12
};
```

اگر بخواهیم برنامه ارائه شده در مثال قبلی را با دستور enum بنویسیم،

```
#include<iostream>
using namespace std;
enum Degree{Diploma, BS, MS};
int main(){
        Degree d;
        cin >> d;
        if(d==Diploma) cout << "good";
        else if(d==BS) cout << "very good";
        else if(d==MS) cout << "well";
        else cout << "Incorrect number";
}
```

۲۳.۳ تابع

مثال: برنامه ای بنویسید که ۳ عدد از کاربر گرفته، ماکسیمم آنها را چاپ کند سپس ۳ عدد دیگر گرفته به ترتیب در اعداد اول ضرب کرده و سپس ماکسیمم مضارب آنها را چاپ کند.

```
#include<iostream>
using namespace std;
int main(){
        int a, b, c;
        cin>>a>>b>>c;
        if(a>=b && a>=c) cout << a;
        if(b>=a && b>=c) cout << b;
        if(c>=a && c>=b) cout << c;
        int x, y, z;
        int px, py, pz;
        cin>>x >>y >>z;
        px = x*a;
        py = y*b;
        pz = z*c;
        if(px>=py && px>=pz) cout << px;
        if(py>=px && py>=pz) cout << py;
        if(pz>=px && pz>=py) cout << pz;
}
```

تابع یا function یک گروه از دستورات است که یک وظیفه خاصی را انجام میدهند. هر برنامه به زبان C یا C++ حداقل یک تابع را داراست که نام آن main است. شما میتوانید کد های خود را به قسمت های با معنی تقسیم کنید و هر قسمت را درون یک تابع قرار دهید. هر تابع یک مجموعه پارامتر ورودی و یک خروجی دارد. عموما وظیفه هایی را که قرار است به تکرار انجام شوند درون یک تابع قرار میدهیم تا نخواهیم در نوشتن، کد را تکرار کنیم.

۱.۲۳.۳ تعریف یک تابع

return_type function_name(parameter list separated with ',')
{/*body of the function...*/}

از مزایای استفاده از توابع:

- خوانایی بیشتر کد.
- کاهش حجم کد.
- شکستن برنامه به بخش های کوچک تر، نوشتن کد و تست کردن آن را نجام میدهد.

هر تابع می تواند انواع خروجی مختلفی داشته باشد: int, float, double, char, ..., void .
از void وقتی که خروجی ای نداریم استفاده میکنیم.

همچنین، ممکن است تابعی پارامتر ورودی نداشته باشد.

void f()→void f(void)

مثال: تابعی تعریف کنید که سلام را چاپ کند.

```
#include<iostream>
using namespace std;
void printhello(){
        cout<<"Hello";
}
int main(){
        printhello();
        return 0;
}
```

مثال: تابعی تعریف کنید که جمع دو عدد ۱۰ و ۲۰ را چاپ کند.

```
#include<iostream>
using namespace std;
void sum(int a,int b){
        cout<<a+b;
}
int main(){
        int x=10,y=20;
        sum(x,y);
        return 0;
}
```

۲.۲۳.۳ مقداردهی اولیه به تابع

گاهی لازم است تا تابع مقدار دهی پیشفرض داشته باشد. برای این کار جلوی تعریف پارامتر، متغیر را برابر با مقدار پیشفرض قرار می دهیم.

```
#include<iostream>
using namespace std;
int boxvolume (int,int.int);
int main(){
```

```
        cout<<boxvolume();//output is: 1*1*1=1\
        cout<<boxvolume(10);//output is: 10*1*1=10
        cout<<boxvolume(10,5);//output is: 10*5*1=50
        cout<<boxvolume(10,5,2);//output is: 10*5*2=100
}
int boxvolume (int length=1, int width=1, int height=1){
        return length*width*height;
}
```

۳.۲۳.۳ متغیرهای عمومی و محلی

در حالت کلی دو نوع متغیرهای محلی و عمومی در برنامه ها وجود دارد. متغیرهای محلی یا local متغیرهایی هستند که درون توابع تعریف و فقط درون تابع قابل دسترسی هستند. متغیرهای global یا عمومی متغیرهایی هستند که از همه بخش های برنامه مورد استفاده قرار می گیرند و در دسترس عموم است.

حال مثالی را که در اول این بحث مطرح کردیم را مجددا با تعریف تابع ایجاد میکنیم تا خصوصیات تابع را بهتر دریابید.

مثال: برنامه ای بنویسید که ۳ عدد از کاربر گرفته، ماکسیمم آنها را چاپ کند سپس ۳ عدد دیگر گرفته به ترتیب در عدد اول ضرب کرده و سپس ماکسیمم مضارب آنها را چاپ کند.

```
#include<iostream>
using namespace std;
void printmax(int a,int b,int c){
        if (a>=b && a>=c) cout<<a;
        else if(b>=a && b>=c) cout<<b;
        else if(c>=a && c>=b) cout<<c;
}
int main(){
        int a,b c;
        cin>>a>>b>>c;
        printmax(a,b,c);
        int x,y,z;
        cin>>x>>y>>z;
        printmax(x*a,y*b,z*c);
}
```

اگر بخواهیم از تابع خروجی بگیریم به جای void از int استفاده میکنیم. میتوانیم از خروجی داده شده در توابع دیگر استفاده کنیم. مثال بالا را مجددا اینگونه تغییر میدهیم:

```
#include<iostream>
using namespace std;
int printmax(int a,int b,int c)
{
        int max;
        if (a>=b && a>=c) max=a;
        else if(b>=a && b>=c) max=b;
        else if(c>=a && c>=b) max=c;
        return max;
}
int main(){
        int a,b c;
```

```
        cin>>a>>b>>c;
        cout<<printmax(a,b,c);
        int x,y,z;
        cin>>x>>y>>z;
        cout<<printmax(x*a,y*b,z*c);
}
```

مثال: تابع توان را بنویسید

```
#include<iostream>
using namespace std;
int pow(int x,int y){
        int result=1;
        for(int i=0;i<y;i++)
                result*=x;
        return result;
}
int main(){
        cout<<pow(2,3);
        return 0;
}
```

مثال: برنامه ای بنویسید که فاکتوریل یک عدد را محاسبه کند.

```
#include <iostream>
using namespace std;
int factorial(int x);
int main(){
        int x;
        cin>>x;
        cout<<"factorial is:"<<factorial(x);
}
int factorial(int a){
        int result=1;
        for(int i=1;i<=a;i++)
                result=result*i;
        return result;
}
```

مثال: تابعی که تشخیص دهد یک عدد اول است یا خیر.

```
#include <iostream>
using namespace std;
bool prime(int a){
        int s=0;
        for(int i=2;i<a;i++){
                if(a%i==0)
                        s++;
        }
```

```
        if (s==0)
                return true;
        else
                return false;
}
int main(){
        int x;
        cin>>x;
        bool result = prime(x);
        if(reult) cout<<x<<" is a prime number";
        else cout<<x<<" is not a prime number";
        return 0;
}
```

مثال: برنامه ای بنویسید که اعداد اول ۱۰۰ تا ۱۰۰۰ را چاپ کند.

```
#include <iostream>
using namespace std;
bool prime(int a){
        int s=0;
        for(int i=2;i<a;i++){
                if(a%i==0)
                s++;
        }
        if (s==0)
        return true;
        else
        return false;
}
int main(){
        for(i=100;i<1000;i++)
                if(prime(i))
                        cout<<i<<'\t';
        return 0;
}
```

داده های static

داده ها یا عمومی هستند و یا محلی. ولی در کنار این نوع دسترسی خاصیت static بودن را نیز میتوانند داشته باشند. داده هایی که از نوع static تعریف می شوند حتی اگر از بلاک اجرایی خارج شوند، باز هم مقدار خود را حفظ خواهد کرد. به نمونه کد زیر توجه کنید.

```
#include <iostream>
using std::cout;
using std::endl;

void uselocal();
void usestaticlocal();
void useglobal();
```

```
int x=1;

int main(){
        {
                int x=5;
                cout<<x;
                x+=3;
                cout<<x;
        }
        cout<<x;
        uselocal();
        usestaticlocal();
        useglobal();
        cout<<x;
        uselocal();
        usestaticlocal();
        useglobal();
        cout<<x;
        return 0;
}
void uselocal(){
        int x=25;
        cout<<x;
        x++;
        cout<<x;
}
void usestaticlocal(){
        static int x=50;
        cout<<x;
        x++;
        cout<<x;
}
void useglobal(){
        cout<<x;
        x*=10;
        cout<<x;
}
```

لازم به توجه است که برای دسترسی به تابع به متغیر x عمومی از درون تابع uselocal میتوان از عملگر :: استفاده کرد.

مثال: خروجی کد زیر را تخمین بزنید.

```
#include <iostream>
using namespace std;
int number=7;
int main(){
        int number=10;
        cout<<number<<'|'<<::number;
        return 0;
```

```
}
```

خروجی:

```
10|7
```

مثال: یک عدد به توان دو برسد.

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
int pow2(int);
int main(){
        int a=10;
        cout<< "squared is:"<< pow2(a);
        return 0;
}
int pow2(int x){
        return x*x;
}
```

۴.۲۳.۳ توابع برخط

توابع برخط یا inline function برای جلوگیری از فراخوانی های متعدد مورد استفاده قرار میگیرد. برخلاف خصوصیت توابع که موجب کاهش حجم می شد، توابع برخط به دلیل کپی کردن تابع در مکانی که فراخوانی می شد موجب افزایش حافظه خواهد شد. چون فراخوانی تابع برای سیستم‌عامل هزینه دارد و برای اینکه سیستم عامل در مواقعی که تعدادشان زیاد می‌شود می‌توان از inline function استفاده کرد.

مثال: حجم یک مکعب را حساب کنید.

```
#include<iostream>
using namespace std;
inline double cuble(double size){
        return size*size*size;
}
int main(){
        double size;
        cin>>size;
        cout<<cuble(size);
        return 0;
}
```

توابع inline سرعت را بهبود می بخشد و حجم تولید شده توسط کامپایلر را افزایش می‌دهد.

ما برنامه نویسان، توابعی که صدا می‌زنند تابعی دیگر را caller و توابعی که صدا زده می شوند را callee نام گذاری می کنیم.

۵.۲۳.۳ انواع ارسال پارامتر به تابع

ما دو نوع ارسال پارامتر به توابع را داریم. این برای همه زبان های برنامه سازی یکسان است. نوع اول ارسال کپی است و نوع دوم ارسال با ارجاع است. در زبان C و C++ برای تحقق این مفهوم روش های زیادی وجود دارد.

در ارسال با کپی یا pass by value از متغیر کپی گرفته می شود و در اختیار تابع قرار داده می شود. که اگر مقدار متغیرها زیاد باشد باعث میشود که این تعداد دو برابر شود و حجم زیادی از توابع را بگیرد. تا کنون هر آنچه نوشته شد ارسال با کپی بوده است. برای نمونه به کد زیر توجه کنید.

```
/*pass by value example*/
#include<iostream>
using namespace std;
double f(int x){ /*codes*/}
int main(){
        int a = 10;
        f(a);
        return 0;
}
```

در ارسال با ارجاع به جای ارسال مقدار کپی از خوده متغیر استفاده میکند و متغیر را به درون تابع ارسال میکند. باید توجه داشت که اگر تغییراتی بر روی داده، اعمال شود، تغییر پس از برگشت از تابع نیز باقی خواهد ماند.

```
/*pass by reference example*/
#include<iostream>
using namespace std;
int squarebyvalue(int);
void squarebyreference(int&);
int main(){
        int x=2;
        int z=4;
        cout<<squarebyvalue(x);
        cout<<x;
        squarebyreference(z);
        cout<<z;
        return 0;
}
int squarebyvalue(int a){
        return a*a;
}
void squarebyreference(int&a){
        a=a*a;
}
```

عملگر ارجاع با نماد & بر روی تعریف پارامتر تابع نشان میدهد که این پارامتر باید از نوع ارسال با ارجاع در دسترس تابع قرار بگیرد. اگر به کد زیر توجه کنید عملگر ارجاع موجب می شود دو متغیر به یک مکان ارجاع دهند. وقتی متغیرهایی داشته باشیم که به یک مکان از حافظه ارجاع می دهند، متغیرهای همنام یا alias نام دارند. اگر به کد زیر توجه کنید متغیرهای CRef و count همنام هستند.

```
#include<iostream>
using namespace std;
int main(){
        int count=1;
        int &CRef = count;
```

```
        CRef ++;
        return 0;
}
```

مثال: خروجی برنامه زیر را تعیین کنید.

```
#include<iostream>
using namespace std;
int main(){
        int x=3;
        int  &y = x;
        cout<<x<<','<<y;
        y=7;
        cout<<x<<','<<y;
        return 0;
}
```

خروجی:

```
3,3
7,7
```

۶.۲۳.۳ توابع همنام

نوشتن تابع، همنام ولی آرگمان های متفاوت (نوع و مقدار) را توابع همنام یا function overloading می گویند. در صورت نوشتن توابع هم نام ولی پارامترهای ورودی متفاوت، کامپایلر خودش متوجه می شود که باید کدام را فراخوانی کند. ولی امکان وجود توابع همنام با پارامترهای یکسان وجود ندارد.

```
#include<iostream>
using namespace std;
int square(int x){
        return x*x;
}
double square(double x){
        return x*x;
}
int main(){
        cout<<square(5)<<endl;
        cout<<square(2.5)<<endl;
        return 0;
}
```

خروجی:

```
25
6.25
```

۷.۲۳.۳ توابع بازگشتی

توابع بازگشتی یا Recursive functions توابعی هستند به صورت مستقیم یا غیر مستقیم خودشان را فراخوانی می کنند. نمونه ای از فراخوانی بازگشتی مستقیم را کد زیر مشاهده می کنید.

```
void f(x){
        /*codes*/
        f(x);
        /*codes*/
}
```

همچنین نمونه ای از کد تابع بازگشتی غیر مستقیم را در زیر مشاهده می کنید.

```
void f(x){
        /*codes*/
        g(x);
        /*codes*/
}
void g(x){
        /*codes*/
        f(x);
        /*codes*/
}
```

توابع بازگشتی باید حتما شرط خاتمه را داشته باشند وگرنه هیچ گاه این برنامه پایان نخواهد یافت. از این نوع برنامه برای بدست آوردن رابطه بازگشتی ریاضی استفاده می شود.

مثال: برای محاسبه فاکتوریل تابع بازگشتی بنویسید.

```
#inlcude<iostream>
using namespace std;
int fact(int n){
        if(n==0) return 1;
        else return n*fact(n-1);
}
int main(){
        cout<<fact(4); // output is 4!=24
}
```

مثال: تابع بازگشتی ای ایجاد کنید که a را به توان b برساند.

```
#inlcude<iostream>
using namespace std;
int mypow(int a, int b){
        if(b==1) return a;
        else return a*mypow(a,b-1);
}
int main(){
        cout<<mypow(4,4); // output is 256
}
```

مثال: تابع بازگشتی ای برای محاسبه ب.م.م به روش نبردبانی ایجاد کنید.

```
#inlcude<iostream>
using namespace std;
int gcd(int a, int b){
        if (a%b==0) return b;
        else return gcd(b, a%b);
```

```
}
int main(){
        cout<<gcd(72,80);//output is 8
}
```

مثال: یک تابع بازگشتی بنویسید که سری فیبوناچی را حساب کند.

```
#inlcude<iostream>
using namespace std;
int fib (int);
int main(){
        int x;
        cin>>x;
        cout<<fib(x);
        return 0;
}
int fib (int a){
        if (a == 0||a==1) return a;
        else return fib(a-1)+fib(a-2);
}
```

۸.۲۳.۳ ارسال آرایه به تابع

برای ارسال یک آرایه به تابع باید از نماد [] استفاده کرد. همچنین در زبان C و C++ در صورت نیاز به اندازه آرایه، باید اندازه آرایه را نیز به تابع ارسال کنیم. در زبان های دیگر ویژگی های دیگری درون خود آرایه وجود دارد و می توان از آن ویژگی ها استفاده کرد ولی در زبان C آن ویژگی ها وجود ندارد و شما وظیفه دارید همراه با ارسال آرایه به تابع آن اندازه آن را نیز به تابع ارسال کنید. در زیر نمونه ای از کد ارسال آرایه به یک تابع را مشاهده می کنید.

```
#include<iostream>
using namespace std;
void print_array(int a[], int size)
{       for(int i = 0;i<size;i++)
        cout<<a[i];
}
int main(){
        const int max_size = 10;
        int a[max_size];
        print_array(a,max_size);
}
```

نکته: ارسال آرایه به تابع همواره از نوع ارسال با ارجاع است.

۹.۲۳.۳ توابع templates

وقتی قرار است یک وظیفه را برای توابع مختلف با پارامترهای مختلف overloading کنید، بهتر است از توابع tem-plates استفاده کنید. پس برای جلوگیری از کدهای زیاد از این روش برای نوشتن چند تابع که برای int و double و char مانند زیر هستند، استفاده می شود.

```
int max (int x1, int x2, int x3){
        int max;
        /*some code*/
```

```
        return max;
}
double max(double x1, double x2, double x3){
        double max;
        /*some code*/
        return max;
}
char max(char x1, char x2, char x3){
        char max;
        /*some code*/
        return max;
}
```

می‌توانیم از یک تابع که چند نوع از تابع ها را یکی می‌کند استفاده کرد که داده های ورودی متفاوت را در خود جای دهد و مکمل ۳ تابع بالا باشد.

```
template <class T>
T max (T x1, T x2, T x3){
        T max;
        /*some code*/
        return max;
}
```

در بالا به جای T می توان هر نوع داده ای می تواند قرار بگیرید. البته که T یک نام است و شما هر نام دیگری را میتوانید جایگزین کنید. همچنین به هر تعداد که خواستار هستید میتوانید متغیر از این نوع تعریف کنید. به نمونه زیر توجه کنید:

```
template <class T, class S>
f (T a, ...,S b)
```

۲۴.۳ کتابخانه cmath

کتابخانه cmath از کتابخانه های زبان C++ است که در روند اجرای کند می توان از توابع اماده شده ریاضی آن استفاده کرد. از جمله توابعی که میتوانید استفاده کنید ceil است که عدد اعشاری دریافت می کند و عدد صحیح حد به بالا را برای ما ایجاد خواهد کرد. همچنین تابع floor عدد صحیح حد پایین را محاسبه خواهد کرد. توابع sin و cos و غیره نیز در این کتابخانه هستند که می توانید مورد استفاده قرار دهید.

#include<iostream>
#include<Cmath>
ceil(x)→[x]→[۲.۴]=۳
floor(x)→[x]→[۲.۴]=۲
sin(x) cos(x)→x :برحسب رادیان
tan(x)
fabs(x) :قدر مطلق
fmod(x) :باقی مانده اعشاری
log(x)→log(۲.۶۱۸۲۸۲)=۱۰
log۱۰(x)=log۱۰(۱۰۰۰)=۲
$pow(x, y) \rightarrow x^y$
sqrt(x)→ $\sqrt[2]{x}$

۲۵.۳ کتابخانه cstdlib

کتابخانه cstdlib از جمله کتابخانه های پرکاربرد است. ازجمله توابع ایجاد مقدار تصادفی و تبدیلات اعداد به کاراکتر و کاراکتر به اعداد می تواند مورد استفاده قرار بگیرد.

#include<cstdlib>

برای تولید یک عدد تصادفی بین ۰ و RAND-MAX از تابع ()rand می توان استفاده کرد. اگر می خواهید یک عدد بین a تا b را به صورت تصادفی ایجاد کنید باید این تابع را به صورت

$$z = rand()\%(b-a)+a$$

تعریف می شود.

۲۶.۳ کتابخانه cstring

کتابخانه cstring توابعی برای تبدیلات کاراکترها و تعامل با کاراکترها ایجاد شده است.

```
#include <cstring>
```

```
int isdigit(char);
```

عدد باشد غیر صفر وگرنه صفر می شود.

```
int isalph(char);
```

حروف الفبا باشد غیر صفر وگرنه صفر می شود.

```
int isalnum(char);
```

عدد یا حرف باشد غیر صفر وگرنه صفر می شود.

```
int islower(char);
```

اگر حروف کوچک باشد غیرصفر وگرنه صفر می شود.

```
int isupper(char);
```

حروف بزرگ باشد غیر صفر وگرنه صفر می شود.

```
int strlen(char[]);
```

طول آرایه ای از کاراکتر یا همان رشته را می دهد. نمونه ای از کاربر تابع strlen را در ادامه مشاهده می کنید.

```
char c[10] = "Hello";
cout<<strlen(c); //output is 5
```

چسباندن آرایه های کاراکتر با تابع :strcat

```
char c1[] = "Hello" ;
char c2[] = "Bye";
cout<<strcat(c1,c2);   // c1+c2 --> "HelloBye"
```

کپی کردن آرایه دوم در آرایه اول با تابع :strcpy

```
char c1[] = "Hello" ;
char c2[] = "Bye";
strcpy(c1,c2);  // c1 = c2
cout<<c1;  // "Bye"
cout<<c2;  // "Bye"
```

مقایسه دو رشته با int: strcmp

```
char c1[] = "Hello" ;
char c2[] = "Bye";
cout << strcmp(c1,c2);  // output != 0
```

در این تابع در صورتی که دو رشته برابر باشند صفر در غیر این صورت غیر صفر خروجی می دهد.
البته خودتان نیز میتوانید این توابع را همانطور که دوست دارید پیاده سازی کنید. در ادامه چند تابع کار با کاراکترها پیاده سازی شده است تا شما روند انجام کار را مشاهده کنید و یاد بگیرید.

```
bool isALPHA(char a[]){
        for(int i = 0;a[i] != '\0';i++)
                if (!isalph(a[i]))
                        return false;
        return true;
}
```

```
void trancate(char a[],int n){
        a[n] = '\0';
}
int main(){
trancate("Hello",0);
// the 'H' will become '\0' so "Hello" never prints!
}
```

در این تابع از n به بعد رشته را قطع می کند. چون قرارداد داریم که انتهای رشته با ۰ معین می شود. در زیر نیز یک تابع جهت شمردن کاراکتر خاص دریافتی از ورودی به تابع خواهد داشت.

```
int count(char a[],char c){
        int count=0;
        for(int i=0; a[i]!='\0'; i++)
                if(a[i]==c)
                        count++;
        return count;
}
```

تابع زیر نیز به اندازه ای که رشته از طول n کمتر باشد کاراکتر فاصله قرار می دهد تا طول رشته برابر با n شود.

```
void padRight(char a[],int n){
        int lent = strlen(a);
        while(len<n){
                a[len] = ' ';
                len++;
        }
        a[len]='\0';
```

```
}
int main(){
        padRight("hey",5);// a = "hey  "
}
```

جهت حذف فاصله های اضافی در سمت راست متن از کد زیر می تواند استفاده کرد.

```
void trimright(char a[]){
        int last=strlen(a)-1;
        while (a[last]==' ')
                last--;
        a[last+1]='\0';
}
```

مثال: برنامه ای بنویسید که پارامترهای ورودی برای محاسبه یک چند جمله را دریافت کند چند جمله را محاسبه و نتیجه را در خروجی نمایش دهد.

$$a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

```
#include<iostream>
using namespace std;
const int maxDegree=10;
int main(){
        int degree;
        int x, sum=0;
        float coefficients[maxDegree+1];
        float xpowers[maxDegree];
        cin>>degree; // get the degree.
        for(int i=0; i<=degree; i++)
                cin>>coefficients[i]; // get a_i
        cin>>x; // get the x;
        xpower[0]=1;
        for(int i=0; i<=degree;i++)
                sum+=coefficients[i]*xpowers[i];
        cout<<sum;
}
```

۲۷.۳ اشاره‌گرها

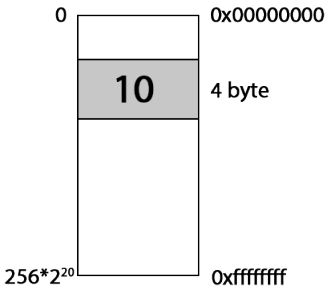
اشاره‌گر متغیری است که آدرس متغیرهای دیگر را در خود نگه می‌دارد. وقتی می‌گوییم حافظه اصلی کامپیوتر 256MB است یعنی:

$$256 \times 2^{10}\ \text{Byte}$$

هر حافظه برای آدرس دهی 4 Byte یا 32 Bit است.

```
cout<<n;
cout<<&n;
```

پس عملگر & برای بدست آوردن آدرس متغیر به کار می‌رود.




```
int i=6;
int *p;
```

با نوشتن این دستور ۴ بایت درون حافظه RAM با نام p تعیین می‌شود که جهت نگهداری آدرس متغیر i مورد استفاده قرار می گیرد.

```
int main(){
        int i=6;
        int *p=&i;
        cout<<i<<&i; //6,0x00001004
        cout<<p;//1004
        cout<<&P;//0x00001008
        cout<<*P;//derefrence
}
```

derefrence میگه آنجایی که دارم اشاره میکنم مقدار درونش چند هست.

```
cout<<*&i;
```

۶ را نمایش می‌دهد.

```
cout<<&*i;
```

ارور می‌دهد زیرا i اشاره‌گر نیست و آدرسی را به ما نمی‌دهد.

```
cout<<*&p;
```

۱۰۰۴ را نمایش می‌دهد.

```
cout<<&*p;
```

۱۰۰۴ را نمایش می‌دهد.

```
int &x=i;
x=*p;
*p=10;
```

با این کار مقادیر x=i=10 می‌شود.

۱.۲۷.۳ بازگشت تابع از نوع ارجاع

جهت ارسال آدرس متغیری که درون تابع ایجاد شده به خارج از آرایه می توان از بازگشت تابع از نوع ارجاع استفاده کرد. در زیر نمونه ای از بازگشت تابع بدون ارجاع نشان داده شده است.

```
int main()
{
        int m=40;
        int n=20;
        cout<<max(m, n); //40
        cout<<m<<n;// 40 20
        max(m,n)=50;// error
        cout<<max(m, n);//40
        cout<<m<<n;//40 20
}
int max (int &m, int &n)
{
        if(m>n)
                return m;
        else return n;
}
```

در صورتی که نیاز به بازگشت به ارجاع باشد به صورت زیر عمل میکنیم.

```
int main(){
        int m=40, n=20;
        cout<<max(m, n)<<m<<n;//40 40 20
        max(m,n)=50; //m=50;
        cout<<max(m, n)<<m<<n;//50 50 20
}
int &max(int &m, int &n){
        if(m>n)
                return m;
        else return n;
}
```

۲.۲۷.۳ عملیات مجاز روی اشاره‌گرها

بر روی اشاره گرها می توان انتساب، عملیات ریاضی و اعداد صحیح، کم کردن دو اشاره گر نسبت به هم را داشت.

۱. انتساب

```
int i=6;
int *p1=&i;
int *p2=p1;
```

۲. عملیات ریاضی و اعداد صحیح

۳. کم کردن دو اشاره‌گر نسبت به هم

```
//p1=1008 , p2=1004;
p1=p1+1; // or p1++
```

```
cout << p1;      // output is 1008
cout << p1-p2; // output is 4
```

مثال: تابعی مثال بزنید که دو متغیر به آن داده شده و جای آنها را با هم عوض کند (به روش اشاره گر ها). در زیر کد مربوطه برای تابع مطرح شده بدون استفاده از اشاره گرها قرار داده شده است ولی اگر توجه کنید پس از بازگشت از تابع مقدارها با هم جابجا نشده اند. چون جابجایی درون تابع رخ داده است و به بیرون تابع منتقل نشده است.

```
void swap(int a, int b){
        int temp = a;
        a=b;
        b=temp;
}
int main(){
        int a, b;
        cin>>a>>b;
        swap(a,b);
}
```

اما با استفاده از اشاره گرها ما میتوانیم به هدفمان برسیم. درادامه کد مربوطه را مشاهده میکنید.

```
void swap(int* a, int* b){
        int temp = *a;
        *a=*b;
        *b=temp;
}
int main(){
        int a, b;
        cin>>a>>b;
        swap(&a,&b);
}
```

۳.۲۷.۳ nullptr

عبارت زیر یک اشاره گر است که به هیچ چیزی اشاره نمیکند. اشاره به نقطع صفر در واقع یک قرار داد است که نشان دهد اشاره گر به جایی اشاره نمیکند. به جای برابر با صفر قرار دادن میتوان مساوی با null یا nullptr یا NULL نیز قرار بگیرد. همه اینها معادل همدیگر هستند.

```
int * p = nullptr;// p=0
```

۴.۲۷.۳ رابطه اشاره گر با آرایه

اسم آرایه یک اشاره گر ثابت است که به خانه اول آرایه اشاره می کند. برای درک بهتر به مثال زیر توجه کنید.

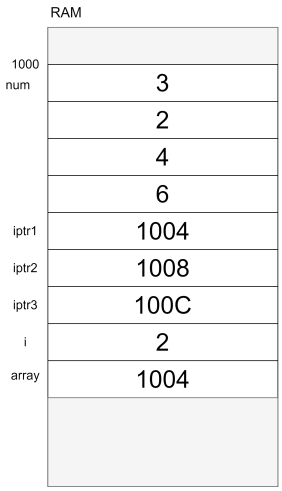
مثال: ابتدا شکل حافظه را برای خودتان ترسیم کنید و سپس خروجی کد زیر را بدست آورید.

```
#include<iostream>
using namespace std;
int main(){
        const int num = 3;
        int array[num]={2, 4, 6};
```

```
        int *iptr;
        int *iptr2;
        int *iptr3;
        int i;
        iptr = array;
        iptr3 = array;
        for(int i = 0; i < num-1 ; i++){
                iptr2 = iptr + i ;
                cout<<i<<array[i]<<*iptr2<<*iptr3<<*(iptr
                    + i)<<endl;
                iptr3++;
        }
}
```

خروجی:

```
0 2 2 2 2
1 4 4 4 4
```



شکل ۵.۳: شمای حافظه برای اجرای کد ارائه شده در مثال بالا

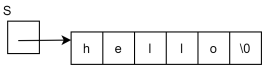
این مفهوم بر روی رشته ها نیز قابل اجراست. جهت درک بهتر به کد زیر توجه کنید.

مثال: شمای حافظه و خروجی را برای کد زیر بدست آورید.

```
#include<iostream>
using namespace std;
int main(){
        char *S = "hello";
        for(int i = 0; i<5 ; i++)
                cout << s1[i]; // or *(s2+i);
}
```

خروجی:

```
hello
```



شکل ۶.۳: شمای حافظه برای اجرای کد ارائه شده در مثال بالا

مثال: خروجی کد زیر را دست آورید.

```
#include<iostream>
using namespace std;
int main(){
        int a[3] = {2, 4, 6};
        int *p = a;
        int *p = a[0];
        cout<<*(p+1)<<p[1]<<a[1]<<*(a+1);
}
```

هر چهار مورد نوشته شده به یک چیز اشاره می کنند و آن خانه دوم آرایه a یعنی ۴ می باشد. خروجی:

```
4 4 4 4
```

مثال: برای کپی گرفتن از رشته یک تابع ایجاد کنید.

```
#include<iostream>
using namespace std;
void copy1(char* , char*);
void copy2(char* , char*);
int main(){
        char string1[10];
        char *string2= "hello" ;
        char string3[10];
        char string 4[]= "goodbye";
        copy1(string1 , string2);
        cout << string1;
        copy2(string3 , string4);
        cout << string2;
        return 0;
}
void copy1(char *string1 , char *string2) {
        for(int i = 0 ; string2[i] != "10" ; i++)
                string1[i] = string2[i];
}
void copy2(char *string1 , char *string2){
        for( ; string2 != "\0" ; string2++ , string1++)
                *string1 = *string2;
}
```

در دو تابع copy۱ و copy۲ در کد بالا یک وظیفه را به دو روش انجام می دهند.

مثال: برای چاپ تک تک خانه های یک رشته یک آرایه ایجاد کنید.

```
#include<iostream>
using namespace std;
void printcharacter(const char*);
int main(){
        const char phrase = "salam";
        printcharacter(phrase);
        return 0;
}
void printcharacter(const char *sptr){
        for ( ; *sptr != '\0' ; sptr++)
                cout << *sptr;
}
```

مثال: خروجی برنامه زیر را بدست آورید:

```
#include<iostream>
using namespace std;
float& component(float *f , int k){
        return f[k-1]; // or *(f+k-1)
}
int main(){
        float f[4];
        for(int i=0; i<4;i++){
                component(f, i) = 1.0/(i+1) ;
                cout<<f[i]<<'\t';
        }
        return 0;
}
```

خروجی:

```
1 1/2 1/3 1/4
```

۵.۲۷.۳ ارسال اشاره گر به تابع

برای ارسال یک اشاره گر به تابع با نماد * بعد از تعریف نوع متغیر استفاده می کنیم. به نمونه زیر توجه کنید. a به عنوان یک اشاره گر مطرح می باشد.

```
void f(int * a) { /*codes*/}
```

گاهی ما برای کم کردن پردازش کپی از اشاره گرها استفاده میکنیم. ولی به خودمان میخواهیم اطمینان دهیم که این پارامتر ورودی بدون تغییر بازخواهد گشت و درون تابع به مقدارهای درون آن کاری نداریم از کلمه const قبل از تعریف پارامتر استفاده میکنیم. به ازای هر پارامتر باید یک بار قبل از هر پارامتر نوشته شود. به نمونه زیر توجه کنید:

```
#include<iostream>
using namespace std;
void printArray(const int* a, int size){
        for(int i=0; i<size; i++)
                cout<<*(a+i);
```

```
}
int main(){
        int* p;
        int a[10]={0};
        printArray(a,10);
        return 0;
}
```

همانطور که مشاهده می شود متغیر a یک اشاره گر به یک آرایه است که از نوع const تعریف شده است. به این معناست که درون تابع printArray مقدارهای درون متغیر a تغییر نخواهد کرد. اگر برنامه نویس به اشتباه یا از قصد بخواهد تغییر دهد، در لحظه کامپایل خطا خواهد داشت. چون کلمه const را به مفهموم اینکه تغییرش نمیدهد نوشته بود. یا نباید کلمه const را می نوشت، یا نباید تغییرش دهد.

۶.۲۷.۳ بازگشت اشاره گر از تابع

برای بازگشت اشاره گر از یک تابع یا خروجی یک تابع از نماد * استفاده می کنیم. به نمونه زیر توجه کنید.

```
int* f() {
                int a=10;
                int* aptr=&a;
                return aptr;
}
int main(){
        int* p = f();
        cout<<*p;
}
```

خروجی:

```
10
```

در حالت کل اگر در یک تابع یک اشاره گر وجود داشته باشد آن فضای مربوطه تغییر نمی کند یعنی از حافظه پاک نمی شود و می توان مقدار آن را تغییر داد. مثال بالا می تواند ادامه پیدا کند:

```
int main(){
        int* p = f();
        cout<<*p;
        *p=20;
        cout<<*p;
}
```

خروجی:

```
10 20
```

۷.۲۷.۳ اشاره گر مقدار ثابت

متغیرهایی از نوع اشاره گر ممکن است داشته باشیم که به مفهوم این است که یک بار که مقداردهی شدند دیگر تغییر نخواهند داشت. فقط یک بار مقداردهی می شوند و تا آخر برنامه فقط به آن یک خانه اشاره می کنند و قابل تغییر نیستند. در صورت مقداردهی مجدد در زمان کامپایل خطا خواهیم داشت. جهت بررسی بیشتر به نمونه کد زیر توجه کنید:

```
int main(){
        int x, y;
        int *const ptr=x;
        *ptr=7;
        ptr=y; // error
        return 0;
}
```

همانطور که مشاهده می شود کلمه const بعد از تعریف نوع متغیر آمده است. اگر قبلش میامد مفهوم فرق می کرد. در نمونه بالا چون const بعد از datatype آمده است می توان مقداری را که ptr به آن اشاره می کند را تغییر داد ولی دیگر نمی توان آن چیز را که اشاره می کند را تغییر داد یعنی اشاره گر فقط به یک چیز اشاره می کند چون ptr به x اشاره می کند و پس دیگر جهت خود را عوض نکرده و به y اشاره نمی کند.
اگر کلمه const را قبل از تعریف نوع متغیر قرار دهیم، مقدار جایی را که اشاره میکند را نمی توان تغییر داد. به نمونه زیر توجه کنید:

```
#include<iostream>
using namespace std;
void main()
{
        int x, y;
        x = 10;
        const int* p = &x;
        x = 11;
        cout << *p;
        p = &y;
        *p = 12; // compiler error
}
```

در نمونه بالا مقدار متغیر را از طریق x نمی توان تغییر داد. البته که شما میتوانید کلمه const را یک بار قبل و یک بار بعد از تعریف نوع نیز قرار دهید. در این صورت نه می توان بیشتر از یک بار مقدار p را تغییر داد و نه می توان مقدار جایی را که p به آن اشاره می کند تغییر داد. جهت بررسی به نمونه زیر توجه کنید:

```
int main(){
        int x=5, y=6;
        const int*const ptr=x;
        cout<<*ptr;
        *ptr=7; // compiler error
        ptr=&y; // compiler error
        return 0;
}
```

حال با توجه به سه نمونه بالا معلوم است که اگر هم بعد و هم قبل از data type ها کلمه const وجود داشته باشد دیگر نه مقدار جهت اشاره گر را می توان عوض کرد و نه مقدار آن چیز را که اشاره می کنیم می توان اشاره کرد.

مثال: تابعی بنویسید که دو آرایه را به عنوان ورودی دریافت کند و دو int از ورودی دریافت کند و بگوید که آیا در بین n عضو اول a و m عضو اول b یافت می شود اگر شد جای خانه ی آخرین خانه را نمایش دهد و اگر نشد ۱- را نمایش دهد.
بدون استفاده از اشاره گرها:

```
int f(a[],b[]n,m){
        for(int i=0;i<n;i++)
                if(a[i]==b[0])
```

```
                for (int j=0;j<m;j++){
                        if(a[i+j] != b[j])
                                break;
                        if(j==m)
                                return j;
                }
        return -1;
}
```

با استفاده از اشاره گرها

```
int* f(int *a,int *b,int n, int m){
        int* end=a+n-1
        for(int* p=a; p<=end; p++)
                if(*p==*b)
                        for(int j=0;j<m;j++)
                                if(p[j]!=b[j])
                                        break;
                                if(j==m)
                                        return p;
        return nullptr;
}
```

۸.۲۷.۳ تخصیص حافظه به صورت پویا

در حافظه مکانی با نام حافظه پویا وجود دارد. نام اصلی آن heap نام دارد. در مقابل حافظه های پویا حافظه های ایستا وجود دارد. تا کنون هرچه خواندیم از نوع حافظه پویا بوده است. در حافظه ایستا حافظه در زمان کامپایل اختصاص داده می شود. این در حالی است که حافظه پویا در لحظه اجرا اختصاص داده می شود. و رفتن و اجازه گرفتن از سیستم عامل جهت برداشتن حافظه کمی تاخیر در کار خواهد داشت. هر دوی حافظه پویا و ایستا در حافظه اولیه یا RAM قرار خواهند گرفت.

```
int main(){
        int * ptr = new int;
        *prt = 5;
}
```

همانطور که در نمونه بالا مشاهده می شود، جهت تعریف یک حافظه به صورت پویا از کلید واژه new استفاده می شود.

یک فضا به اندازه سایز size آن datatype به برنامه ما در زمان اجرا در محفظه حافظه‌ای Heap اختصاص داده می شود و آدرس آن فضا را در نام متغیر نگهداری خواهد کرد. یعنی ما یک اشاره گر با نام متغیری که داریم و این حافظه از نوع ایستا است که به یک حافظه از نوع datatype در Heap اشاره می کند و ما بطور مستقیم به آن دسترسی نداریم.

برای اینکه این متغیرها خود به خود بعد از استفاده پاک نمی شوند و با پایان یافتن برنامه از حافظه پاک می شوند، باید بعد از استفاده از این‌چنین متغیرهایی، حتما آنها را پاک کنیم و پاک کردن آنها به این صورت است:

```
int main(){
        int * ptr = new int;
        *prt = 5;
        delete ptr;
}
```

ساختار کلی جهت تولید و حذف یک متغیر در حافظه پویا به صورت زیر است:

```
datatype * varName = new datatype;
delete varName;
varName=nullptr;
```

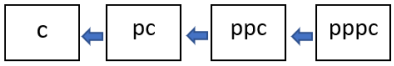
همچنین برای تعریف متغیر پویا آرایه به این صورت است که:

```
datatype *varName = new datatype[size];
delete [] varName;
varName=nullptr;
```

۹.۲۷.۳ اشاره‌گر به اشاره‌گر

به اشاره گری که به اشاره دیگری اشاره میکند اشاره گر به اشاره گر گفته می شود.

```
char c = 'a';
char *pc = &c;
char **ppc = &pc;
char ***pppc = &ppc;
.
.
.
```



شکل ۷.۳: شمای حافظه اشاره گر به اشاره گر

۱۰.۲۷.۳ عملگر sizeof

اپراتور (size of): تعداد بایت های اختصاص یافته به یک متغیر و یا یک نوع داده را مشخص می کند. اندازه را برحسب بایت نمایش خواهد داد.

```
int main(){
        int i;
        cout<<sizeof(i); // output is 4.
}
```

وقتی size of را همراه اسم متغیر به کار ببریم، میتوانیم بدون پرانتز نیز بنویسیم، ولی برای datatype ها باید حتما پرانتز به کار ببریم.

```
int main(){
        int i;
        cout<<sizeof i; // output is 4.
        cout<<sizeof(int); // output is 4.
        cout<<sizeof(char); // output is 1.
        cout<<sizeof(short); // output is 2.
        cout<<sizeof(double); // output is 8.
        cout<<sizeof(float); // output is 4.
        cout<<sizeof(long double); // output is 12.
        cout<<sizeof(unsigned int); // output is 4.
```

```
        cout<<sizeof(signed int); // output is 4.
        cout<<sizeof(double*); // output is 4.
        cout<<sizeof(int*); // output is 4.
}
```

۱۱.۲۷.۳ آرایه‌ای از اشاره‌گرها

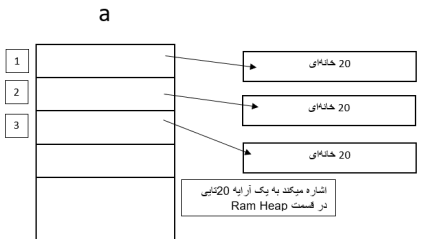
همچون نوع های داده ای معمول، اشاره گرها نیز میتوانند به صورت مجموعه از آرایه مورد بهره برداری قرار بگیرند. در زیر یک نمونه از تعریف آرایه ای از اشاره گرها بیان شده است.

```
int main(){
        const int b=5;
        int z = 10;
        int *a[b];
        a[0] = &z;
        cout<<a[0]; // output is 10.
}
```

با بهره بری از این تکنیک میتوانیم آرایه های چند بعدی را در حافظه های پویا داشت.

```
int main(){
        const int b=5;
        int *a[b];
        for (int i=0; i<b; i++)
                a[i] = new int[20];
        for (int i=0; i<b; i++)
        delete [] a[i];
        delete [] a;
        a=nullptr;
}
```

با این کار برای هر a[i] ها یک آرایه ۲۰تایی از نوع عدد اختصاص می دهیم. توجه کنید که سطرها بصورت static است و ستون ها بصورت پویاست. همچنین میتوان اندازه ای را که ۲۰ تعیین شده را متفاوت در نظر گرفت. در نمونه



شکل ۸.۳: شمای حافظه آرایه های دو بعدی در حافظه های پویا

زیر فقط a در حافظه ایستا قرار می گیرد و باقی در حافظه پویا است.

```
int main(){
        int n, x;
        cin>>n>>x;
        int **a;
```

```
        a = new int*[n];
        for (int i=0; i<n; i++)
                a[i] = new int[x];
        for (int i=0; i<n; i++)
                delete [] a[i];
        delete [] a;
        a=nullptr;
}
```

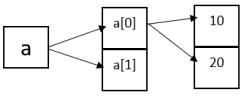
همین عملیات را می توان به با کاراکترها نیز اعمال کرد. اشاره گر به اشاره گرها را بر روی حافظه ایستای و پویا نشان می دهد.

```
int main(){
        char **s = {"Hello", "Bye"};
        char **s = new char*[2];
        s[0] = "Hello";
        s[0][0] = "H"; s [0][5] = '\0';
        s[1] = "Bye";
        delete s;
}
```

مثال: خروجی کد زیر را بدست آورید.

```
int main(){
        int **a = new int*[2];
        a[0]=new int[2];
        a[0][0]=10;
        a[0][1]=20;
        cout<<a<<*a<<&a<<a[0]<<*a[0]<<&a[0]<<&*a[0]<<endl
            ;
        cout<<a[0][0]<<*&a<<&*a<<&a[0][0]<<*a[0][0];
        delete [] a[0];
        delete [] a;
        a=0;
}
```

مورد آخر چون به جایی اشاره نمی کند، پس ارور می دهد.
با این وجود :



شکل ۹.۳: شمای حافظه برای مثال بالا

```
a = *&a = &*a = &a[0] = 0x3e3es8
a[0] = a[0][0] = 10
*a = a[0] = &a[0][0] = &*a[0] = 0x3e3ea0
```

۲۸.۳ اشاره گر Void

اشاره گری است که نوع آن مشخص نیست. چون یک اشاره گر Void Pointer است پس نمی تواند derefrence بر روی آن انجام شود.

```
void * a ;
(*a)++ ; // Compiler Error
```

و این جایی که نخواهیم نوع اشاره گر مشخص باشد کاربرد دارد.

مثال: خروجی کد زیر را تحلیل کنید

```
int main(){
        int a = 10;
        increase(&a ; sizeof(a))
        cout << a ;
}
void increase(void *P,int size)
{
        *P++;
        if (size == size of (int)){
                int *a = (int*)P;
                (*a)++;
        }
        if (size == size of (char)){
                char *a = (char*)P;
                (*a)++;
        }
}
```

خروجی:

compiler error; because of "*P++;" operation within the increase function.

مثال: خروجی کد زیر را بدست آورید.

```
#include<iostream>
using namespace std;
int main(){
        int *P = new int;
        *P = 10;
        inerease (P);
        cout << *P;
        delete P;
}
void inerease(int * a){
        (*a)++;
}
```

خروجی:

11

۲۹.۳ Struct

یک نوع داده گروهی که فضای پیوسته و متوالی از حافظه را اشغال میکند، مانند آرایه ها و فرق اصلی آنها با آرایه ها در این است که تمام داده های آرایه یک نوع باشد ولی در Struct هر نوع از any type هر خانه ای می تواند داشته باشد.

```
struct{
        {structure elements};
};
```

مثال: برای ساعت یک struct ایجاد کنید.

```
struct time
{
        int hour;
        int minute;
        int second;
};
int main(){
        time t ;
        cout << sizeof (t);//3*4 = 12
}
```

مثال: مشخصات یک بانک را در یک Struct قرار دهید .

```
struct account
{
        int account_num;
        char account_type;
        char account_name[50];
        float balance ;
        time last payment;
} accl;
int main()
{
        accl.account_name = 1210;
        accl.account_type = "a";
        accl.balance = 447.5;
        accl.payment.hour = 10;
        for(int i = 0 ; i < 50 ; i++)
        {
                cin >> accl.account_name[i];
        }
}
```

مثال: اطلاعات نام و سن افرادی را دریافت کنید.

```
#include <iostream>
using namespace std;
struct person {
    char name[100];
    int age;
```

```
};
void f(person P)
{
    P.age = 100;
}
void g(person *P)
{
    (*P).age = 100;
}
int main ()
{
    person P1 = {"ali",10};
    cout << P1.age;
    f(P1);
    cout << P1.age;
    person *P2 = new person;
    g(P2);
    cout << (*P2).age;
    person *P = new person[5];
    P->age = 100;
    P[1].age = 10;
    (*(P+1)).age = 10;
    (P+1)->age = 10;
    (&P[1])->age=10;
}
```

Union ۳۰.۳

Union یک نوع داده ای تعریف شده توسط برنامه نویس است که بر خلاف ،Struct همه اعضای آن در یک خانه از حافظه قرار می گیرند. اندازه یک Union بر اساس اندازه بزرگترین عضو آن تعیین می شود. اگر مدنظر دارید دو یا چند متغیر را یکی را استفاده کنید، union یکی از ساختارهای است که این امکان را برای ما فراهم خواهد کرد. ساختار تعریف کردن Union در زیر نوشته شده است.

```
union union_name {
        member definition;
};
```

در اینجا union_name نام دلخواهی است که برنامه نویس انتخاب میکند. member definition; نیز یک مجموعه از متغیرها است.

مثال: خروجی کد زیر را بدست آورید.

```
union Data {
        int i;
        float f;
};
int main( ) {
        Data data, data1;
        cout<<"Memory size occupied by data:"<< sizeof(
            data)<<'\t'<<sizeof(data1);
        return 0;
```

```
}
```

۳۱.۳ پردازش فایل در C

همانگونه که می دانید در زبان های برنامه سازی از جمله C ، کلیه متغیرها در حافظه اولیه (RAM) ذخیره می گردند. اگرچه حافظه اصلی دارای سرعت بسیار بالایی است، اما با پایان یافتن اجرای برنامه اطلاعات آن از بین می روند. با این وجود در بسیاری از برنامه ها، باید اطلاعات برای اجرای بعدی برنامه حفظ شوند. به عنوان مثال در یک سیستم دانشجویی، باید اطلاعات دانشجویان و دروس آنها بر روی یک حافظه پایدار ذخیره گردد، تا با اتمام برنامه اطلاعات آنها از بین نرود. راه حل این مشکلات استفاده از حافظه پایداری به نام حافظه جانبی است. اگرچه حافظه های جانبی سرعت بسیار کمتری نسبت به حافظه اصلی دارند، اما قادرند اطلاعات خود را حتی پس از قطع جریان برق نیز حفظ نمایند.
انواع مختلفی از حافظه های جانبی وجود دارد، مانند: دیسک سخت و نرم، نوار، دیسکهای نوری (CD) و اخیرا نیز Disk Flash ها. واحد ذخیره سازی اطلاعات بر روی حافظه های جانبی فایل است. فایل مجموعه ای از داده ها است که به نحوی با یکدیگر در ارتباط هستند. برای ذخیره اطلاعات یک برنامه، کافی است یک فایل بر روی حافظه جانبی ایجاد کرده و داده ها را در آن ذخیره نماییم. در این بخش با نحوه انجام این کار آشنا خواهیم شد.
فایلها به دو دسته اصلی تقسیم می گردند: فایلهای متنی (text) فایلهای دودویی (binary) .
درک تفاوت بین این دو بسیار مهم است. برای روشن شدن موضوع، به یک مثال ساده توجه کنید. فرض کنید یک متغیر صحیح به صورت زیر تعریف کرده ایم:

```
int a = 9281;
```

همانگونه که می دانید این متغیر در حافظه به صورت یک عدد دودویی ذخیره می گردد. اگر برنامه در محیط DOS باشد، برای این متغیر دو بایت حافظه به صورت شکل زیر در نظر گرفته شده و عدد ۹۲۸۱ به صورت دودویی در آن ذخیره می گردد.

00100100|01000001

حال فرض کنیم یک فایل دودویی به نام test.dat ، و یک فایل متنی بنام test.txt ایجاد کرده و متغیر a را در آنها نوشته ایم. در فایل دودویی test.dat ، متغیر a دقیقا به همان شکلی که در حافظه قرار دارد، ذخیره می گردد. البته در هنگام ذخیره در فایل، ابتدا بایت با ارزش پایینتر و سپس بایت با ارزش بالاتر ذخیره می گردد. بنابراین، این متغیر دو بایت فضا را در فایل اشغال خواهد کرد. اما در فایل متنی test.txt ، کلیه اطلاعات به صورت رشته ای از کاراکترها ذخیره می گردند. بنابراین متغیر صحیح a ، ابتدا به رشته "9281" تبدیل شده و سپس این چهار کاراکتر به ترتیب در فایل نوشته می شوند. در حقیقت در این فایل متنی، به ترتیب کدهای اسکی مربوط به کاراکترهای '9' ، '2' ، '8' و سرانجام '1' نوشته خواهد شد. درنتیجه این متغیر، چهار بایت فضا را در فایل اشغال خواهد کرد.
مزیت فایلهای متنی نسبت به فایلهای دودویی آن است که می توان آنها را توسط هر ویرایشگر متنی (مانند Edit و یا Notepad) باز کرد و محتوای آنها را دید. توجه کنید که یک ویرایشگر متنی، فایل را به صورت رشته ای از کاراکترها می بیند، و آن را به صورت کاراکتر به کاراکتر می خواند. چنانچه یک فایل دودویی توسط یک ویرایشگر متنی باز شود، اطلاعات به شکل نادرستی نمایش داده خواهند شد و کاربر فقط مجموعه ای از کاراکترهای نامفهوم را خواهد دید. به عنوان مثال شکل ۱۰.۳ نحوه نمایش دو فایل test.dat و test.txt را توسط یک ویرایشگر متنی، نشان می دهد.

شکل ۱۰.۳: مثالی جهت نمایش اختلاف نوع های ذخیره سازی به صورت متنی و دودویی

اگر به نحوه نمایش فایل test.dat دقت کنید، می بینید که این فایل حاوی دو کاراکتر (بایت) می باشد. کاراکتر A، دارای کد اسکی ۶۵ است که در مبنای دو برابر ۰۱۰۰۰۰۰۱ می باشد. کاراکتر $ نیز دارای کد اسکی ۳۶ است که در مبنای دو برابر ۰۰۱۰۰۱۰۰ خواهد شد. حال اگر این دو عدد را با نحوه ذخیره متغیر a در حافظه مقایسه کنید، متوجه خواهید شد که این دو همان بایتهای تشکیل دهنده عدد ۹۲۸۱ هستند.

به طور کلی، یک فایل دودویی توسط ویرایشگر متنی قابل خواندن نیست و فقط توسط خود برنامه می تواند خوانده شده و تفسیر گردد.
از آنجا که فایل های متنی توسط هر ویرایشگری قابل خواندن هستند، معمولا از آنها برای ذخیره گزارش و یا اطلاعاتی که باید توسط کاربر قابل مشاهده و چاپ باشد، استفاده می گردد.
به فایل های متنی، فایل های قالب بندی شده نیز گفته می شود، چرا که برنامه نویس می تواند داده ها را در یک قالب دلخواه و با شکل مناسب برای کاربر در فایل قرار دهد. مثلا در صورت لزوم می توان داده ها را در داخل یک جدول با عنوان های مناسب قرار داد، تا کاربر راحت تر بتواند از آنها استفاده نماید.
اما در مقابل، فایل های دودویی دنباله ای از بایت های خام هستند که فقط توسط برنامه می توانند خوانده شده و تفسیر گردند.
یک فایل دودویی از خارج برنامه قابل استفاده نمی باشد، چرا که یک فایل دودویی ۲۰۰ بایتی، می تواند ۱۰۰ عدد صحیح (دو بایتی) و یا ۵۰ عدد اعشاری (۴ بایتی) باشد. بنابراین تنها خود برنامه نویس می داند اطلاعات را چگونه ذخیره کرده است، و به چه نحوی باید آنها را خوانده و تفسیر نماید.
توجه کنید که در فایلهای دودویی مواردی همچون فاصله گذاری بین اطلاعات، رفتن به خط بعد، جدول کشی و ... معنا ندارد، چرا که این فایلها از خارج برنامه قابل مشاهده و چاپ نیستند. اما فایلهای دودویی دارای مزایای متعدی نسبت به فایل های متنی هستند که باعث شده است برنامه نویسان در اغلب موارد از این دسته فایل ها برای ذخیره و بازیابی اطلاعات استفاده نمایند. این مزایا عبارتند از:

- سرعت بالا:
  - این فایلها داده ها را به همان نحوی که در حافظه ذخیره می شوند، نگهداری می کنند.
  - بنابراین در هنگام نوشتن یا خواندن داده ها به/از فایل، نیازی به تبدیل داده ها به رشته کاراکتری و بالعکس نمی باشد. درحالیکه در فایلهای متنی، انجام این تبدیل ها، باعث پایین آمدن سرعت می گردد.
- حجم کمتر:
  - فایلهای دودویی حجم کمتری را نسبت به فایلهای متنی اشغال می کنند، چرا که معمولا تبدیل اعداد صحیح و یا اعشاری به رشته، باعث می شود که تعداد بایتهای بیشتری اشغال گردد.
  - به عنوان مثال عدد اعشاری ۳۲۱.۱۴۵۶ که در فایل دودویی به صورت یک داده چهار بایتی ذخیره می گردد، دریک فایل متنی احتیاج به یک رشته هشت بایتی دارد (برای ذخیره رشته "1456.321").
- امکان دسترسی تصادفی:
  - فرض کنید یک فایل متنی حاوی تعدادی عدد صحیح داریم. حال قصد داریم دهمین عدد این فایل را بخوانیم. برای این کار باید از ابتدای فایل شروع کرده و نه عدد اول را خوانده و کنار بگذاریم تا به عدد دهم برسیم.
  - چرا که تعداد بایتهای تخصیص داده شده به هر عدد مشخص نیست و بستگی به تعداد ارقام آن دارد (مثلا ۱۲ نیاز به دو بایت و ۶۷۱۴ نیاز به چهار بایت دارند).
  - به این روش دسترسی به داده ها، دسترسی ترتیبی گفته می شود.
  - اما در یک فایل دودویی مشابه، می توان بلافاصله به ابتدای دهمین عدد پرش کرده و آن را خواند؛ چرا که هر عدد صحیح دقیقا دو بایت را اشغال می کند، بنابراین چنانچه ۱۸ بایت از ابتدای فایل جلو برویم، به عدد دهم خواهیم رسید.
  - به این روش دسترسی به داده ها، دسترسی تصادفی یا مستقیم گفته می شود. در قسمتهای بعدی، روش دسترسی تصادفی را بررسی خواهیم کرد.
- اطلاعات از بیرون برنامه قابل خواندن و بروزرسانی نیستند.
  - این خاصیت در مواردی که داده ها مهم بوده و نمی خواهیم سایر افراد از بیرون برنامه به اطلاعات دسترسی داشته باشند، مفید است.

۱.۳۱.۳ نحوه دستیابی به فایل ها

برای دستیابی به یک فایل، باید مراحل زیر انجام گردد:

۱. تعریف یک متغیر برای فایل

۲. باز کردن فایل

۳. خواندن و نوشتن در فایل

۴. بستن فایل

کلیه داده ها و توابع لازم برای عملیات فوق در فایل سرآمد stdio.h تعریف شده اند، بنابراین برای کار با فایلها باید این فایل را در برنامه گنجاند.
در ادامه، هریک از موارد فوق مورد بررسی قرار گرفته اند.

۲.۳۱.۳ تعریف فایل

برای تعریف یک فایل، باید از نوع داده ای بنام FILE استفاده نمایید. در حقیقت نوع داده FILE ، یک ساختار است که در فایل stdio.h تعریف شده است. این ساختار حاوی اطلاعاتی همچون :

- نام خارجی فایل بر روی دیسک: نام واقعی فایل در سیستم عامل
- نوع فایل: ورودی، خروجی، دودویی، متنی
- آدرس محل بافر اطلاعات: مکانی از حافظه است که در هنگام عملیات ورودی و خروجی، به عنوان واسط بین برنامه و فایل استفاده می شود.
- اشاره گر به مکان فعلی فایل: شماره بایتی از فایل را نشان می دهد که در اولین عمل خواندن یا نوشتن بعدی، استفاده خواهد شد. از این اشاره گر در قسمتهای بعدی برای دسترسی تصادفی استفاده خواهیم کرد.
- وضعیت خطاها مانند رسیدن به آخر فایل، خطا در خواندن یا نوشتن و ...

توجه کنید که در حقیقت نیازی به دانستن عناصر ساختار FILE و نام آنها نیست، چرا که کلیه عملیات توسط توابع زبان C انجام خواهد شد و موارد گفته شده تنها برای آشنایی بیشتر شما با این ساختار بود.
برای استفاده از یک فایل در برنامه، ابتدا باید متغیری از نوع اشاره گر به ساختار FILE تعریف نمایید. به عنوان مثال به تعریف زیر توجه کنید:

```
FILE   *inputFile ;
```

این تعریف، متغیر inputFile را از نوع اشاره گر به FILE تعریف می نماید.

۳.۳۱.۳ باز کردن فایل

برای استفاده از یک فایل، ابتدا باید آن را باز کنید و یا به عبارت بهتر، آن را به برنامه خود مربوط کنید. برای این کار از تابعی بنام fopen استفاده می شود. اعلان این تابع به صورت زیر است:

```
FILE*   fopen(char fileName[], char mode[]);
```

پارامتر fileName ، رشته ای است که نام خارجی فایل در سیستم عامل را مشخص می کند. این رشته می تواند علاوه بر نام فایل، حاوی مسیر فایل نیز باشد. به عنوان مثال چنانچه این رشته به صورت "letter.txt" مشخص شود، در فه رست جاری به دنبال این فایل جستجو می شود. اما رشته ای مانند "C:\myLetters\letter.txt" ، آدرس فیزیکی فایل را به طور کامل مشخص می نماید.
پارامتر دوم یا mode ، رشته ای است که نحوه باز شدن فایل را نشان می دهد. این رشته از دو قسمت تشکیل می گردد. قسمت اول، عملیات قابل اجرا بر روی فایل را نشان می دهد و یکی از موارد مشخص شده در جدول ۱۱.۳ است.

جدول ۱۱.۳: جدول وضعیت بازکردن فایل ها در زبان C

رشته	عمل قابل اجرا	توضیحات
r	خواندن	فایل باید موجود باشد، در غیر این صورت مقدار NULL بازگردانده می شود.
w	نوشتن	یک فایل جدید ایجاد می شود، اگر فایل از قبل موجود باشد اطلاعات آن از بین خواهد رفت.
a	افزوردن به انتها	اگر فایل موجود باشد، اطلاعات به انتهای آن اضافه می شود در غیر این صورت فایل جدید ایجاد می شود بر روی آن می نویسد.
r+	خواندن و نوشتن	فایل باید موجود باشد، در غیر این صورت NULL بازگردانده می شود.
w+	نوشتن و خواندن	یک فایل جدید ایجاد می شود، اگر فایل از قبل موجود باشد، اطلاعات آن از بین خواهد رفت.
a+	افزودن و خواندن	اگر فایل موجود باشد، اطلاعات به انتهای آن اضافه می گردد . در غیر این صورت ایجاد می گردد.

جدول ۱۲.۳: جدول وضعیت نوع فایل ها در زبان C

رشته	نوع فایل	توضیحات
t	متنی	فایل به صورت متنی باز می شود.
b	دودویی	فایل به صورت دودویی باز می شود.

و اما قسمت دوم رشته ،mode نوع فایل را نشان می دهد و یکی از موارد مشخص شده در جدول ۱۲.۳ است.
این دو قسمت به یکدیگر متصل شده و تشکیل رشته نحوه باز شدن فایل را می دهند. به عنوان نمونه، رشته "wb" به معنای نوشتن بر روی یک فایل دودویی است. لازم به ذکر است که اگر قسمت دوم ذکر نشود، به طور پیش فرض از نوع متنی (t) در نظر گرفته می شود. به عنوان مثال "r+" به معنای خواندن و نوشتن بر روی یک فایل متنی است.
تابع fopen ، ابتدا حافظه لازم برای یک ساختار از نوع FILE را تخصیص داده و سپس با توجه به آرگومانهای ارسالی (نام واقعی فایل و نحوه باز شدن آن)، عناصر آن را مقدار دهی اولیه می کند، و اشاره گری به آن را باز می گرداند. اکنون می توان این اشاره گر بازگشتی را به یک متغیر از نوع اشاره گر به FILE نسبت داده و از آن برای عملیات خواندن و نوشتن در فایل استفاده کرد. به مثال زیر دقت کنید:

```
FILE *inputFile;
inputFile = fopen("letter.txt", "rt");
```

در این مثال، تابع fopen فایلی بنام letter.txt را به صورت متنی و برای خواندن باز کرده و یک اشاره گر به FILE باز می گرداند، که آن را در متغیری به نام inputFile قرار داده ایم. به عنوان یک مثال دیگر، به نمونه زیر دقت کنید:

```
FILE *itemFile;
itemFile = fopen("C:\\data\\items.dat","w+b");
```

این دستور، فایلی از آدرس "C:\data\items.dat" را برای خواندن و نوشتن، به صورت دودویی باز کرده و اشاره گر FILE آن را در متغیر itemFile قرار می دهد.
توجه کنید که چنانچه تابع fopen به هر دلیلی نتواند فایل مورد نظر را باز کند (مثلا به دلیل موجود نبودن فایل یا فهرست مورد نظر، یا پر بودن دیسک و غیره)، مقدار NULL (که برابر با ۰ است) را باز می گرداند. بنابراین بهتر است پس از باز کردن فایل، ابتدا درستی باز شدن آن بررسی گردد. به مثال زیر توجه کنید:

```
FILE *inputFile;
inputFile = fopen("letter.txt", "rt");
if (inputFile == NULL) {
        printf("can't␣open␣file!\n");
```

```
        exit(1) ;
}
```

البته معمولا برنامه نویسان قسمت شرط if را بصورت زیر می نویسند:

```
if (!inputFile)  // ...
```

۴.۳۱.۳ بستن فایل

پس از آنکه عملیات ما بر روی فایل خاتمه پیدا کرد، باید آن را توسط تابع fclose ببندیم. اعلان این تابع به شکل زیر است:

```
int  fclose(FILE *f)
```

این تابع، یک اشاره گر به فایل را به عنوان ورودی دریافت و آن را می بندد. در صورت موفقیت در بستن فایل مقدار ۰، و در صورت عدم موفقیت مقدار EOF بازگردانده می شود. EOF یک ثابت نمادی است که برابر مقدار ۱- تعریف شده است. به عنوان مثال دستور زیر، فایلی که توسط متغیر inputFile به آن اشاره می شود را می بندد:

```
fclose(inputFile);
```

با بستن یک فایل، ارتباط آن با برنامه قطع می شود. چندین دلیل برای بستن فایل وجود دارد:

- تعداد فایلهایی که می توانند به طور همزمان باز باشند، محدود است و در نتیجه هرگاه به فایلی نیاز نداریم، بهتر است آن را ببندیم.
- هنگام عملیات نوشتن در فایل، داده ها ابتدا در یک بافر در حافظه نوشته می شوند (به قسمت تعریف فایل مراجعه کنید)، سپس هنگامی که بافر پر شد، به طور یکجا به فایل ارسال می شوند.با بستن فایل، چنانچه هنوز داده ای در بافر باقی مانده باشد، به فایل ارسال می شود.
- تابع fclose فضای تخصیص یافته به متغیر اشاره گر فایل را آزاد می کند که باعث صرفه جویی در حافظه می شود. علاوه براین می توان از این متغیر مجددا برای باز کردن یک فایل دیگر استفاده کرد.

البته چنانچه اجرای یک برنامه به طور عادی خاتمه پیدا کند، خود کامپایلر تمام فایلهای باز آن را می بندد؛ ولی بهتر است این کار توسط خود برنامه نویس انجام شود.
برای بستن فایل، تابع دیگری نیز وجود دارد. تابع fcloseall تمام فایلهای باز برنامه را می بندد. این تابع هیچ آرگومانی دریافت نمی کند و اعلان آن به صورت زیر است:

```
int  fcloseall();
```

این تابع در صورت موفقیت، تعداد فایلهای بسته شده و در صورت عدم موفقیت، مقدار EOF را باز می گرداند. آخرین نکته ای که به آن اشاره می کنیم، در مورد بافر است. همانطور که گفته شد، با بسته شدن یک فایل، محتوای بافر آن خالی شده و به فایل اصلی ارسال می گردد. اما چنانچه پیش از بستن فایل بخواهیم بافر را خالی کنیم، می توانیم از تابع fflush استفاده نماییم. اعلان این تابع به صورت زیر است:

```
int  fflush(FILE *f)
```

این تابع، بافر فایلی را که توسط پارامتر f به آن ارسال کرده ایم، خالی می نماید. در صورت موفقیت مقدار ۰، و درصورت عدم موفقیت مقدار EOF (یا ۱-) باز گردانده می شود.

۵.۳۱.۳ ورودی و خروجی در فایلها

از آنجا که نحوه خواندن و نوشتن در فایلهای متنی با فایلهای دودویی متفاوت است، هریك از آنها را در یك بخش جداگانه بررسی می نماییم. قبل از اینکه این مبحث را آغاز نماییم، باید اشاره ای به یك متغیر بسیار مهم، یعنی اشاره گر به مکان فعلی فایل داشته باشیم. همان طور که در قسمتهای قبلی گفته شد، ساختار FILE دارای عضوی است که

به مکان فعلی فایل اشاره می کند. این عضو، مکانی از فایل را نشان می دهد که عمل خواندن یا نوشتن بعدی ازیا به آنجا انجام خواهد شد. هنگامی که یك فایل را باز می کنیم، این اشاره گر به اولین بایت فایل اشاره می کند، مگر اینکه فایل را برای اضافه کردن به انتها (با گزینه (a باز کرده باشیم، که در این صورت به انتهای فایل اشاره خواهد کرد. با هر عمل خواندن یا نوشتن، این اشاره گر به تعداد بایتهای خوانده یا نوشته شده، جلو می رود.
حال فرض کنید فایلی را برای خواندن باز کرده ایم. با اولین عمل خواندن یك کاراکتر، اولین بایت آن خوانده خواهد شد. اما از آنجا که اشاره گر مکان فعلی فایل نیز به اندازه یك بایت جلو می رود، در دومین عمل خواندن کاراکتر، دومین بایت آن فایل خوانده خواهد شد (و نه اولین بایت) و عملیات به همین شکل ادامه می یابد.
لازم به ذکر است که کلیه این عملیات توسط خود توابع زبان C انجام خواهد شد، و نیازی نیست که برنامه نویس نگران محل اشاره گر مکان فعلی فایل باشد. البته می توانیم مکان این اشاره گر را توسط توابعی که معرفی خواهیم کرد، تغییر دهیم.
روال نوشتن در فایل نیز مشابه خواندن است؛ به این معنا که با هر بار عمل نوشتن یک داده، اشاره گر مکان فعلی به تعداد بایتهای آن داده جلو خواهد رفت. به این ترتیب هر داده جدید، به طور خودکار در مکان پس از داده قبلی نوشته خواهد شد (و نه بر روی آن).

۶.۳۱.۳ ورودی و خروجی در فایلهای متنی

همانطور که قبلا نیز گفته شد، فایل های متنی توسط هر ویرایشگری قابل خواندن هستند. لذا برنامه نویس باید بتواند داده ها را در قالب دلخواه خود در فایل قرار دهد تا توسط کاربران قابل استفاده باشد. به همین دلیل به فایلهای متنی، فایلهای قالب بندی شده نیز می گویند.
اگر به یاد داشته باشید، عملیات ورودی و خروجی در زبان C از طریق دو تابع scanf و printf انجام می گرفت. برای عملیات ورودی و خروجی در فایلهای متنی نیز دو تابع مشابه وجود دارد: fscanf و .fprintf تابع fprintf که برای نوشتن داده های خروجی در یك فایل استفاده می شود، دارای شکل کلی زیر است:

```
fprintf(<file-pointer>,<control-string>, <variable-list>)
    ;
```

مثال: برنامه ای بنویسید که میزان فروش روزانه یك شرکت را براي تمام روزهاي سال (۳۶۵ یا ۳۶۶ روز) دریافت، و آنها را دریك فایل متني به نام sales.txt قرار دهد. میزان فروش روزانه یك عدد صحیح است.

```
#include <stdio.h>
int main() {
        FILE *outFile ;
        int i, day, sales;
        outFile = fopen("sales.txt", "wt");
        if (!outFile) {
                printf("can't open file!");
                return -1;
        }
        printf("enter number of days : ");
        scanf("%d", &day);
        for (i=0; i<day; i++) {
                printf("enter daily sales (day %d) : ", i
                    +1);
                scanf("%d", &sales) ;
                fprintf(outFile, "%d\n", sales);
        }
        fclose(outFile);
}
```

برای خواندن اطلاعات از یك فایل متنی، از تابع fscanf استفاده می گردد. شکل کلی این تابع بصورت زیر است:

```
fscanf(<file-pointer>,<control-string>, <variable-address
    -list>);
```

تعریف این تابع نیز تقریبا همانند تابع scanf است، با این تفاوت که به عنوان اولین آرگومان، یك اشاره گر به فایل را دریافت می کند که مشخص کننده فایلی است که داده ها باید از آن خوانده شوند. دو آرگومان بعدی دقیقا همانند تابع scanf بوده و نحوه استفاده از آنها نیز کاملا مشابه است. به عنوان نمونه، به یک مثال توجه کنید.

مثال: برنامه ای بنویسید که فایل متنی sales.txt ، مربوط به مثال قبلی را خوانده و میانگین فروش روزانه شرکت را بر روی نمایشگر چاپ نماید.

```
#include <stdio.h>
void main() {
        FILE *inFile ;
        int sales, counter=0 ;
        float average = 0.0;
        inFile = fopen("sales.txt", "rt");
        if (!inFile) {
                printf("can't open file!");
                return ;
        }
        fscanf(inFile, "%d",&sales) ;
        while ( ! feof(inFile) ) {
                average += sales ;
                counter ++;
                fscanf(inFile, "%d",&sales) ;
        }
        fclose(inFile);
        average /= counter ;
        printf("average = %f",average);
}
```

نکته مهم در حل این مسئله آن است که تعداد اعداد فایل sales.txt از قبل مشخص نیست (ممکن است ۳۶۵ یا ۳۶۶ باشد)، بنابراین برنامه باید عمل خواندن را تا زمانی که به انتهای فایل برسد، ادامه دهد.

حتما از مطالب مطرح شده در اول این قسمت به یاد دارید که با هر بار عمل خواندن از فایل، اشاره گر مکان جاری فایل به اندازه تعداد بایتهای خوانده شده جلو می رود. بنابراین باید عمل خواندن را تا زمانی که این اشاره گر به انتهای فایل برسد، ادامه دهیم. اما چگونه می توان فهمید که این اشاره گر به انتهای فایل رسیده است؟ خوشبختانه برای این کار تابعی به نام feof وجود دارد که اعلان آن به صورت زیر است:

```
int   feof(FILE *f);
```

این تابع یك اشاره گر به فایل را دریافت کرده، و در صورتی که اشاره گر مکان فعلی آن به انتهای فایل رسیده باشد، مقدار درست (غیر صفر) و درغیر این صورت مقدار نادرست (صفر) را باز می گرداند.

اما در استفاده از این تابع باید دقت کرد. نکته مهم در اینجا است که این تابع هنگامی مقدار درست باز می گرداند که در آخرین عمل خواندن با شکست مواجه شده و به پایان فایل رسیده باشد. برای روشن شدن موضوع، فرض کنید فایلی دارای ۱۰ عدد صحیح است. چنانچه پس از ۱۰ بار عمل خواندن از فایل، تابع feof را فراخوانی کنید، مقدار صفر را باز خواهد گرداند که نشان می دهد هنوز انتهای فایل را ندیده است. اما چنانچه یك عمل خواندن دیگر (یازدهمین عمل خواندن) انجام دهید، گرچه این عمل ناموفق خواهد بود چرا که داده ای برای خواندن وجود ندارد، اما تابع متوجه می گردد که به انتهای فایل رسیده است. این بار فراخوانی تابع feof مقدار غیرصفر باز خواهد گرداند.

۷.۳۱.۳ ورودی و خروجی در فایلهای دودویی

همانطور که گفته شد، در فایلهای دودویی اطلاعات به صورت بایتهای خام ذخیره می گردند و به همین دلیل توسط ویرایشگرها قابل خوانده شدن نیستند. در این نوع فایلها، داده ها به صورت پشت سرهم ذخیره می گردند و از آنجا که اندازه (تعداد بایتهای) هر داده مشخص است، احتیاجی به استفاده از کاراکترهای جدا کننده مانند فضای خالی نیست.
به طور کلی برای خواندن و نوشتن در این فایلها، احتیاج به ورودی/خروجی قالب بندی شده نداریم، بلکه به توابعی نیاز داریم که بتوانند داده ها را به صورت دنباله ای از بایتهای خام به فایل ارسال کنند و یا از آن بخوانند. به همین دلیل به این فایلها، فایلهای قالب بندی نشده نیز گفته می شود. برای خواندن و نوشتن در فایلهای دودویی، دو تابع fread و fwrite را داریم. اعلان تابع fwrite که برای نوشتن در فایلهای دودویی به کار می رود، به شکل زیر است:

```
size_t  fwrite(void *data, size_t  size, size_t  n, FILE
    *f);
```

ابتدا توجه کنید که نوع داده size_t ، نام دیگری برای عدد صحیح بدون علامت (unsigned int) می باشد. این نوع داده توسط دستور typedef در فایل stdio.h تعریف شده است. پارامتر data ، آدرس داده ای است که قصد نوشتن آن در فایل را داریم. همانطور که پیشتر نیز گفته شد، برای به دست آوردن آدرس يك متغير، از علامت & پيش از نام متغير استفاده می نماییم. پارامتر size، اندازه داده ای که باید نوشته شود را بر حسب بايت مشخص مي نمايد. پارامتر n نیز تعداد داده هایی که باید نوشته شوند را مشخص می کند. در قسمتهاي بعدي خواهيد ديد كه اين تابع مي تواند تعدادي داده را در قالب يك آرايه به طور يكجا در فايل بنويسد. آخرين پارامتر نيز اشاره گر به فايلي است که قصد نوشتن داده ها در آن را داریم. این تابع به عنوان خروجی، تعداد داده های (نه تعداد بایتهای) نوشته شده را باز می گرداند.
مثال: برنامه ای بنویسید که میزان فروش روزانه يك شركت را براي تمام روزهاي سال (۳۶۵ یا ۳۶۶ روز) دریافت و آنها را دريك فايل دودويي بنام sales.dat قرار دهد. میزان فروش روزانه يك عدد صحيح است.

```
#include <stdio.h>
void main() {
        FILE *outFile ;
        int i, day, sales ;
        outFile = fopen("sales.dat", "wb");
        if (!outFile) {
                printf("can't␣open␣file!");
                return ;
        }
        printf("enter␣number␣of␣days␣:␣");
        scanf("%d", &day);

        for (i=0; i<day; i++) {
                printf("enter␣daily␣sales␣(day␣%d):␣", i
                    +1);
                scanf("%d", &sales) ;
                fwrite(&sales, sizeof(int), 1, outFile);
        }
        fclose(outFile);
}
```

براي خواندن داده ها از يك فايل دودويي، از تابع fread استفاده مي شود. اعلان اين تابع بصورت زير است:

```
size_t  fread(void *data, size_t  size, size_t  n, FILE *
    f);
```

پارامترهاي اين تابع همانند تابع fwrite می باشند، با اين تفاوت كه اولين پارامتر، آدرس داده اي است كه بايد از فايل خوانده شود. در ضمن مقدار خروجی اين تابع، تعداد داده های (و نه بايتهای) خوانده شده می باشد. به مثال زير توجه كنيد.

مثال: برنامه ای بنويسيد كه فايل دودويي sales.dat ، مربوط به مثال قبلی را خوانده و ميانگين فروش روزانه شركت را بر روی نمايشگر چاپ نمايد.

```
#include <stdio.h>
void main() {
        FILE *inFile ;
        int sales, counter=0 ;
        float average = 0.0;
        inFile = fopen("sales.dat", "rb");
        if (!inFile) {
                printf("can't open file!");
                return ;
        }
        fread(&sales , sizeof(int), 1, inFile) ;
        while ( ! feof(inFile) ) {
                average += sales ;
                counter ++;
                fread(&sales, sizeof(int), 1, inFile) ;
        }
        fclose(inFile);
        average /= counter ;
        printf("average = %f",average);
}
```

۸.۳۱.۳ سایر توابع ورودی و خروجی فایل

به جز توابع گفته شده، توابع ديگری نيز برای ورودی يا خروجی در فايلها وجود دارند. توابع fgets و fputs برای خواندن و نوشتن رشته ها در فايل بكار می روند. اعلان تابع fgets به شكل زير است:

```
char*  fgets(char string[], int n, FILE *f);
```

اين تابع يک رشته كاراكتری را از فايل f خوانده و در رشته string قرار می دهد. خواندن رشته تا زمانی ادامه می يابد كه يا در فايل به خط جديد (new line) برسيم، و يا به تعداد n-۱ كاراكتر خوانده شود. يک كاراكتر باقيمانده نيز برای $'\backslash 0'$ درنظر گرفته شده است. مقدار بازگشتی اين تابع، آدرس رشته خوانده شده است. اعلان تابع fputs نيز به شكل زير است:

```
int  fputs(char string[], FILE *f);
```

دسته ديگر، توابع fgetc و fputc هستند كه به ترتيب برای خواندن و نوشتن يک كاراكتر در فايل به كار می روند. اين توابع هنگامی استفاده می شوند كه قصد داريد يک فايل را به صورت بايت به بايت (كاراكتر به كاراكتر) پردازش نماييد. اگرچه اين دو تابع معمولا در فايلهای دودويی استفاده می شوند، اما گاهی از آنها برای پردازش فايلهای متنی نيز استفاده می شود. اعلان اين دو تابع بصورت زير است:

```
int  fgetc(FILE *f);
int  fputc(int c, FILE *f);
```

جدول ۱۳.۳: جدول انواع دسترسی در زبان C

مقدار	ثابت نمادی	مفهوم	مقدار مجاز تعداد بایت
۰	SEEK_SET	جابجایی از ابتدای فایل شروع شود.	۰ یا یک عدد مثبت برای حرکت رو به جلو
۱	SEEK_CUR	جابجایی از مکان فعلی اشاره گر شروع شود.	عدد مثبت: حرکت رو به جلو عدد منفی: حرکن رو به عقب
۲	SEEK_END	جابجایی از انتهای فایل شروع شود.	۰ یا یک عدد منفی برای حرکت رو به عقب

۹.۳۱.۳ دسترسی مستقیم به فایلها

به دو شکل مي توان فايلها را مورد پردازش قرار داد:

- پردازش ترتيبي: در اين روش، پردازش از ابتداي فايل شروع مي شود و داده ها يك به يك پردازش مي گردند تا به انتهاي فايل برسيم. اين روش در مواقعي مناسب است كه بخواهيم عملياتي را بر روي كل فايل و يا قسمت بزرگي از آن انجام دهيم.
- پردازش مستقيم (تصادفي): در اين روش مي توان به طور مستقيم به يك مكان (آدرس) مشخص از فايل دسترسي پيدا كرد. اين روش در مواردي موثر است كه بخواهيم عملياتي را بر روي برخي داده هاي مشخص فايل انجام دهيم.

با توجه به مطالب گفته شده قبلي، دسترسي مستقيم فقط در مورد فايلهاي دودويي معنا دارد. چرا كه در اين فايلها با توجه به ثابت بودن اندازه هر داده، مكان هر يك از داده ها كاملا مشخص است. مثلا چنانچه فايلي از اعداد صحيح به صورت دودويي داشته باشيم و فرض كنيم هر عدد صحيح ۲ بايت باشد، براي خواندن دهمين عنصر كافي است اشاره گر مكان جاري فايل را به بايت شماره ۱۸ برده (ابتداي دهمين عدد) و سپس عمل خواندن را انجام دهيم. اما آيا مي توان همين روش را در مورد يك فايل متني از اعداد صحيح نيز بكار برد؟ با توجه به اينكه در چنين فايلي، تعداد بايتهاي تخصيص يافته به هر عدد بستگي به تعداد ارقام آن دارد، چگونه مي توان مكان (آدرس) دهمين عدد را تعيين كرد؟ مسلما تنها روش آن است كه از اول فايل شروع كرده و عدد اول را بخوانيم تا به جداكننده برسيم، سپس عدد دوم و ... تا به دهمين عدد برسيم. بنابراين معمولا در فايلهاي متني از پردازش ترتيبي استفاده مي شود.
به ياد داريد كه هر فايل داراي يك اشاره گر به مكان فعلي فايل است. اين اشاره گر به مكاني از فايل اشاره مي نمايد كه عمل خواندن يا نوشتن بعدي در آن انجام خواهد شد. براي دسترسي مستقيم به داده هاي فايل، نياز به تابعي داريم كه به ما امكان تغيير مكان اين اشاره گر را بدهد. اين تابع fseek نام دارد و اعلان آن به صورت زير است:

```
int  fseek(FILE *f, long int offset, int place);
```

مقادير مجازی برای پارامتر place در جدول ۱۳.۳ آمده است:
توجه كنيد كه دومين پارامتر، يعني offset مي تواند منفي نيز باشد كه به معناي حركت رو به عقب است.
اين تابع در صورت موفقيت مقدار ۰، و در غير اينصورت يك مقدار غير صفر باز مي گرداند. به عنوان نمونه به مثال زير توجه نماييد:
مثال: برنامه اي بنويسيد كه ميانگين فروش ماه مهر و همچنين ميزان فروش در آخرين روز سال را از فايل sales.dat استخراج كرده و چاپ نمايد.

```
#include <stdio.h>
void main() {
        FILE *inFile ;
        int i, sales, lastDaySales ;
        float averageMehr = 0.0;
        inFile = fopen("sales.dat", "rb");
        if (!inFile) {
                printf("can't␣open␣file!");
                return ;
```

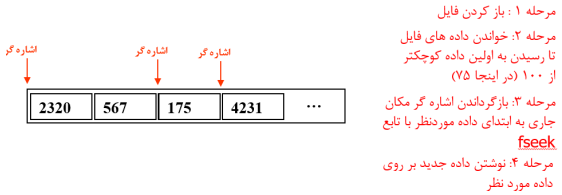
```
        }
        fseek(inFile, 186 * sizeof(int), SEEK_SET);
        for (i=0; i<30; i++) {
                fread(&sales , sizeof(int), 1, inFile) ;
                averageMehr += sales ;
        }
        averageMehr /= 30 ;

        fseek(inFile, -1L * sizeof(int), SEEK_END);

        fread(&lastDaySales , sizeof(int), 1, inFile) ;
        printf("average = %f\n",averageMehr);
        printf("last day sales = %d",lastDaySales) ;
        fclose(inFile);

}
```

در مثال بالا پرش به ابتدای اطلاعات ماه مهر صورت میگیرد. همچنین، حلقه مربوط به خواندن میزان فروش ماه مهر و محاسبه میانگین آنها را خواهد داشت. در ادامه، پرش به میزان فروش آخرین روز سال را خواهد داشت.
يكي از مهمترين كاربردهاي تابع fseek ، بروز رساني (انجام تغييرات) فايلهاي دودويي است. به عنوان مثال فرض كنيد اطلاع پيدا كرده ايم كه در فايل sales.dat، يكي از فروشهاي روزانه به طور اشتباه زير ۱۰۰ عدد وارد شده است، و بايد به آن ۱۰۰ واحد اضافه شود. براي انجام اين كار بايد از ابتداي فايل شروع به خواندن نماييم تا به اولين داده كوچكتر از ۱۰۰ برسيم. اما پس از پيدا كردن داده موردنظر، ابتدا بايد اشاره گر مكان فعلي فايل را به اندازه يك عدد صحيح به عقب بازگردانده و سپس عدد جديد را بنويسيم، در غيراينصورت عدد جديد بر روي فروش روز بعد نوشته خواهد شد. شكل ۱۱.۳ نحوه انجام كار را بر روي يك فايل نمونه نشان مي دهد.



شکل ۱۱.۳: مثالی از نحوه بروزرسانی بر روی فایل های دودویی در زبان C

کد مربوط به شکل ۱۱.۳ را در ادامه مشاهده می کنید.

```
#include <stdio.h>
void main() {
        FILE *updateFile ;
        int sales ;
        updateFile = fopen("sales.dat", "r+b");
        if (!updateFile) {
                printf("can't open file!");
                return ;
        }
        fread(&sales , sizeof(int), 1, updateFile) ;
        while ( ! feof(updateFile) ) {
                if (sales < 100) {
                        sales += 100 ;
```

```
                        fseek(updateFile, -1L * sizeof(
                            int), SEEK_CUR);
                        fwrite(&sales, sizeof(int), 1,
                            updateFile);
                        break;
                }
                fread(&sales, sizeof(int), 1, updateFile)
                    ;
        }
        fclose(updateFile);
}
```

۱۰.۳۱.۳ خواندن و نوشتن ساختارها در فایل

در بسیاری از موارد ممکن است بخواهید یك ساختار را در یك فایل بنویسید، یا آن را از یك فایل بخوانید. نحوه انجام این کار در فایلهای متنی و دودویی کاملا متفاوت است. برای خواندن یا نوشتن یك ساختار در یك فایل متنی، باید هریك از اعضای آن را به طور جداگانه از فایل خواند و یا در آن نوشت. دلیل این مسئله نیز کاملا مشخص است. برای ارسال یك ساختار به یك فایل متنی، باید نحوه و قالب نوشتن هر عضو، به طور جداگانه مشخص شود. مثلا آیا هر عضو در یك ردیف جدا چاپ شود و یا با یك فاصله از عضو بعدی جدا گردد. همین مسئله در هنگام خواندن نیز باید رعایت گردد. بنابراین نمی توان کل یك ساختار را به طور یکجا در فایل نوشت یا از آن خواند.

مثال: برنامه ای بنویسید که مشخصات تعدادی دانشجو را از کاربر دریافت و آنها را دریك فایل متنی بنام stu-dent.txt قرار دهد. اطلاعات هر دانشجو شامل شماره دانشجویی، نام، سن و معدل دانشجو می باشد.

```
#include <stdio.h>
struct student {
        long int id;
        char name[40];
        int age;
        float average;
};
void main() {
        student s;
        FILE *studentFile;
        int i, n;
        studentFile = fopen("student.txt","wt");
        if (! studentFile) {
                printf("can't␣open␣file.");
                return;
        }

        printf("please␣enter␣number␣of␣students:␣");
        scanf("%d", &n);
        for (i=0; i<n; i++) {
                printf("student␣%d:␣enter␣id,␣name,␣age␣
                    and␣average:␣", i+1);
                scanf("%ld␣%s␣%d␣%f", &s.id, s.name, &s.
                    age, &s.average);
                fprintf(studentFile,"%ld␣%s␣%d␣%5.2f\n",s
                    .id, s.name, s.age, s.average);
        }
```

```
        fclose(studentFile);
}
```

اما خوشبختانه روش کار در فایلهای دودویی بسیار ساده تر است. برای خواندن یا نوشتن یك ساختار در یك فایل دودویی، می توان آن ساختار را به صورت یکجا از فایل خواند یا در آن نوشت. هنگامی که یك ساختار را در یك فایل دودویی می نویسیم، کلیه بایتهای عناصر آن بصورت پشت سرهم و بدون هیچ فاصله ای در فایل نوشته می شوند. همانطور که قبلا نیز گفتیم، از آنجا که تعداد بایتهای هر یك از عناصر ساختار کاملا مشخص است، نیازی به استفاده از یك جداکننده نیست. در هنگام خواندن یك ساختار از فایل نیز، همان عناصر (با تعداد بایتهای مشخص) به صورت پشت سرهم از فایل خوانده می شوند.

مثال: برنامه ای بنویسید که مشخصات تعدادی دانشجو را از کاربر دریافت و آنها را دریك فایل دودویی بنام student.dat قرار دهد. اطلاعات هر دانشجو شامل شماره دانشجویی، نام، سن و معدل دانشجو می باشد.

```
#include <stdio.h>
struct student {
        long int id;
        char name[40] ;
        int age;
        float average ;
} ;
void main() {
        student s;
        FILE *studentFile;
        int i, n;
        studentFile = fopen("student.dat","wb");
        if (! studentFile) {
                printf("can't␣open␣file.");
                return ;
        }

        printf("please␣enter␣student␣number␣:␣");
        scanf("%d", &n);
        for (i=0; i<n; i++) {
                printf("student␣%d:␣enter␣id,␣name,␣age␣
                    and␣average:␣", i+1);
                scanf("%ld␣%s␣%d␣%f", &s.id, s.name, &s.
                    age, &s.average);
                fwrite(&s, sizeof(student), 1,
                    studentFile) ;
        }
        fclose(studentFile);
}
```

مثال: برنامه ای برای بروزرسانی اطلاعات دانشجویان بنویسید. این برنامه باید یك شماره دانشجویی را دریافت و در فایل دودویی دانشجویان (student.dat) به دنبال آن جستجو نماید. سپس در صورت پیدا شدن دانشجو، اطلاعات قبلي شامل نام، سن و معدل وی را با اطلاعات جدیدی که از کاربر دریافت می کند، جایگزین نماید.

```
#include <stdio.h>
struct student {
        long int id;
        char name[40] ;
        int age;
```

```
        float average ;
} ;
void main() {
        student s;
        FILE *studentFile;
        int i;
        long int searchId ;
        studentFile = fopen("student.dat","r+b");
        if (! studentFile) {
                printf("can't open file.");
                return ;
        }

        printf("please enter student id : ");
        scanf("%ld", &searchId);
        do
        fread(&s, sizeof(student), 1, studentFile) ;
        while (!feof(studentFile) && s.id != searchId);
        if (feof(studentFile))
                printf("student not found!\n");
        else {
                printf("student found!\n");
                printf("enter new name, age and average :
                    ");
                scanf("%s %d %f", s.name, &s.age, &s.
                    average);
                fseek(studentFile, -1L * sizeof(student),
                     SEEK_CUR);
                fwrite(&s, sizeof(student), 1,
                    studentFile) ;
        }
        fclose(studentFile);
}
```

۱۱.۳۱.۳ خواندن و نوشتن آرایه ها در فایل

یکی دیگر از مسائلی که ممکن است با آن مواجه شوید، خواندن یا نوشتن کل محتویات یک آرایه در یک فایل است. به عنوان یک مثال ساده فرض کنید یک آرایه از اعداد اعشاری به صورت زیر تعریف شده است:

```
float   data[100];
```

پس از آنکه در برنامه به نحوی به این آرایه مقداردهی کرده ایم، قصد داریم آن را به طور کامل در یک فایل ذخیره کنیم. در اینجا نیز نحوه ذخیره آرایه دریک فایل متنی با یک فایل دودویی متفاوت است. برای ذخیره یک آرایه در یک فایل متنی، باید هریک از عناصر آرایه را به طور جداگانه در فایل نوشت. بنابراین برای این کار نیاز به استفاده از یک حلقه داریم. قطعه برنامه زیر نحوه انجام این کار را نشان می دهد:

```
outFile = fopen("data.txt", "wt");
for (i=0; i<100; i++)
fprintf(outFile, "%f", data[i]) ;
```

خواندن آرایه از یک فایل متنی نیز به همین صورت انجام می پذیرد. یعنی عناصر آرایه باید به صورت تک به تک از آرایه خوانده شوند. به عنوان مثال قطعه برنامه زیر مجددا آرایه data را از همان فایل می خواند:

```
inFile = fopen("data.txt", "rt");
for (i=0; i<100; i++)
fscanf(inFile, "%f", &data[i]) ;
```

اما برای نوشتن یک آرایه در یک فایل دودویی، می توان آن را با استفاده از تابع fwrite به صورت یکجا در فایل نوشت و نیازی به استفاده از حلقه نمی باشد.
برای نوشتن آرایه ها در فایل کافی است در هنگام فراخوانی تابع fwrite، نام آرایه را به تنهایی (بدون استفاده از علامت &) به عنوان آرگومان اول و تعداد عناصر آرایه را به عنوان آرگومان سوم (که مشخص کننده تعداد داده ها می باشد) ارسال نماییم. به عنوان مثال، قطعه برنامه زیر آرایه data را در یک فایل دودویی می نویسد:

```
outFile = fopen("data.dat", "wb");
fwrite(data, sizeof(float), 100, outFile);
```

خواندن یک آرایه از یک فایل دودویی نیز به طور مشابه و فقط با یک بار فراخوانی تابع fread انجام می شود. قطعه برنامه زیر آرایه data را از همان فایل دودویی بالا می خواند:

```
inFile = fopen("data.dat", "rb");
fread(data, sizeof(float), 100, inFile);
```

موارد فوق برای سایر نوع داده ها، از جمله نوع داده های تعریف شده توسط کاربر (مانند ساختارها) نیز برقرار است. به عنوان مثال اگر ساختاری مانند ساختار دانشجو را تعریف کرده باشید، می توانید یک آرایه از نوع دانشجو را به طور یکجا در یک فایل دودویی بنویسید و یا از آن بخوانید.