



جزوه درس

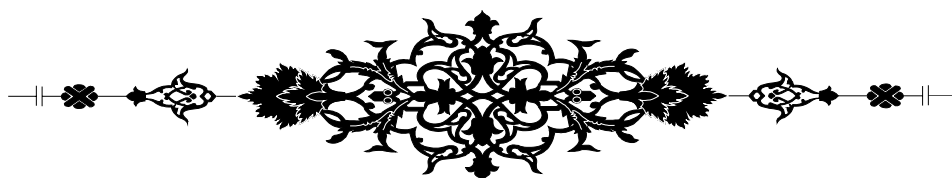
مبانی کامپیوتر و برنامه سازی

تهیه کننده

علی چوداری خسروشاهی

ویژه دانشجویان کارشناسی کامپیوتر - نرم افزار

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



پیشگفتار

ما تلاش کرده ایم در این جزوه تمام سرفصلهای درس پوشش داده و مطالب مفیدی را در اختیار شما قرار دهیم تا شما با خواندن آن کلیاتی را از درس بدست آورید. در مورد این جزوه یادآوری این نکته مهم، ضروری است که این جزوه جهت کمک به دانشجویان در کاهش یادداشت برداری، ترسیم شکلها، مثالها در هنگام تدریس بوده است. بنابر این در کنار این جزوه، هر دانشجو مطابق با ذوق و سلیقه خود، جزوه ای دست نویس خواهد داشت که حاوی یادداشت ها به منظور تکمیل و تفهیم این جزوه است. این جزوه، ادعای جایگزینی مراجع اصلی این درس را ندارد بلکه تنها خلاصه ای از مطالب مهم از مراجع درس است. در گردآوری این جزوه از مراجع زیر استفاده شده است:

- برنامه نویسی به زبان C

نویسنده: عین الله جعفر نژاد قمی

- الگوریتم و فلوچارت

انتشارات مجتمع فنی تهران

در اینجا لازم است از تلاش تمام دانشجویانی که اینجانب را در تدوین این جزوه یاری نموده اند کمال تشکر را داشته باشیم. همچنین در این جزوه احتمال اشتباه وجود دارد، به این جهت در مطالعه مطالب جزوه دقت کافی را داشته باشید. از دانشجویان عزیز تقاضا می شود در مطالعه مطالب آن دقت کافی را داشته باشند و وجود هرگونه ایراد، و همچنین نظرات و پیشنهادات خود را به آدرس پست الکترونیکی اینجانب اطلاع دهند.

Akhosroshahi@IAUT.ac.ir
Akhosroshahi@Gmail.com

باتشکر

علی چوداری خسروشاهی

قسمت اول: الگوریتم و فلوچارت

۲	۱- الگوریتم
۲	۱-۱- خصوصیات الگوریتم.....
۲	۲-۱- شرط و پرش در الگوریتم.....
۴	۳-۱- حلقه ها در الگوریتم.....
۶	۲- فلوچارت
۶	۱-۲- نمادهای فلوچارت.....
۶	۱-۱-۲- نماد شروع و پایان.....
۶	۲-۱-۲- نماد جایگزینی و انتساب.....
۷	۳-۱-۲- نماد ورودی.....
۷	۴-۱-۲- نماد شرطی.....
۷	۵-۱-۲- نماد چاپ (خروجی).....
۸	۶-۱-۲- نماد ادامه.....
۸	۲-۲- حلقه ها در فلوچارت.....
۱۴	تمرینات
۱۵	۳- آرایه ها (لیست ها)
۱۵	۱-۳- آرایه یک بعدی.....
۱۵	۲-۳- آرایه دوبعدی.....
۱۵	۳-۳- آرایه سه بعدی.....

قسمت دوم: برنامه نویسی به زبان C

۲۱	1- مقدمات زبان C
۲۱	۱-۱- کلمات کلیدی.....
۲۱	۲-۱- Comment یا توضیحات.....
۲۱	۳-۱- انواع داده‌ای.....
۲۱	۴-۱- خصوصیات دستورالعمل در زبان C.....
۲۲	۵-۱- متغیرها.....
۲۲	۶-۱- تعریف ثابت.....
۲۳	۷-۱- عملگرها.....
۲۳	۱-۷-۱- عملگرهای محاسباتی.....
۲۴	۲-۷-۱- عملگرهای رابطه‌ای.....
۲۴	۳-۷-۱- عملگرهای منطقی.....
۲۵	۴-۷-۱- عملگرهای ترکیبی.....
۲۵	۵-۷-۱- عملگرهای بیتی.....
۲۶	۶-۷-۱- تقدم عملگرها در حالت کلی.....
۲۶	۷-۷-۱- عملگرهای * و &.....

۲۶	۸-۷-۱- عملگر ؟
۲۶	۹-۷-۱- عملگر sizeof
۲۷	۸-۱- تبدیل انواع داده‌ای
۲۸	۲- ساختار برنامه زبان C و ورودی/خروجی
۲۸	۱-۲- دستورات ورودی و خروجی
۲۸	۱-۱-۲- تابع () printf
۳۰	۲-۱-۲- تابع () scanf
۳۰	۳-۱-۲- توابع () getch و () getche
۳۱	۴-۱-۲- تابع () getchar
۳۱	۵-۱-۲- نوشتن کاراکتر با توابع () putchar و () putch
۳۲	۳- حلقه های تکرار و ساختارهای تصمیم
۳۲	۱-۳- حلقه for
۳۳	۲-۳- انواع خطاها
۳۴	۳-۳- حلقه while
۳۴	۴-۳- حلقه do...while
۳۵	۵-۳- ساختار تصمیم if
۳۶	۶-۳- دستور break
۳۶	۷-۳- دستور continue
۳۶	۸-۳- ساختار تصمیم switch
۳۸	تمرینات
۳۹	۴- توابع و کلاسهای حافظه
۳۹	۱-۴- توابع
۴۱	۱-۱-۴- روش های ارسال پارامتر ها به توابع
۴۳	۲-۱-۴- متغیرهای محلی و عمومی
۴۳	۳-۱-۴- روش بازگشتی
۴۵	۲-۴- کلاس های حافظه
۴۶	۱-۲-۴- کلاس حافظه اتوماتیک
۴۶	۲-۲-۴- کلاس حافظه ثبات
۴۶	۳-۲-۴- کلاس حافظه استاتیک
۴۷	۴-۲-۴- کلاس حافظه خارجی
۴۹	۵- آرایه ها و رشته ها
۴۹	۱-۵- آرایه های تک بعدی
۵۰	۱-۱-۵- ارسال آرایه های تک بعدی به عنوان پارامتر به توابع
۵۱	۲-۵- مرتب سازی آرایه ها
۵۱	۱-۲-۵- روش مرتب سازی حبابی
۵۱	۲-۲-۵- مرتب سازی انتخابی
۵۲	۳-۵- جستجو در آرایه

۵۲	۵-۳-۱- جستجوی ترتیبی.....
۵۲	۵-۳-۲- جستجوی دودویی.....
۵۳	۵-4- آرایه دو بعدی.....
۵۳	۵-4-1- ارسال آرایه های دو بعدی به عنوان آرگومان به توابع.....
۵۳	۵-5- مقداردهی اولیه آرایه ها.....
۵۳	۵-۶- آرایه های n بعدی.....
۵۴	۵-۷- رشته ها.....
۵۴	۵-۷-۱- مقدار دهی اولیه رشته ها.....
۵۴	۵-۷-۲- ورودی و خروجی رشته ها.....
۵۴	۵-۷-۳- ارسال رشته ها به عنوان پارامتر به توابع.....
۵۶	۵-۷-۴- انتساب رشته ها.....
۵۶	۵-۷-۵- مقایسه رشته ها.....
۵۶	۵-۷-۶- الحاق رشته ها.....
۵۶	۵-۷-۷- آرایه ای از رشته ها.....

۶- اشاره گر ها..... ۵۷

۵۷	۶-۱- متغیرهای اشاره گر.....
۵۷	۶-۲- اعمال روی اشاره گر ها.....
۵۸	۶-۲-۱- عمل انتساب اشاره گر ها به یکدیگر.....
۵۸	۶-۲-۲- اعمال محاسباتی جمع و تفریق.....
۵۸	۶-۳- تخصیص حافظه ی پویا.....
۵۹	۶-۴- اشاره گر ها و توابع.....
۵۹	۶-۵- اشاره گر ها و آرایه ها.....
۶۰	۶-۶- آرایه پویا.....
۶۲	۶-۷- ارزش دهی اولیه به اشاره گر ها.....
۶۲	۶-۸- اشاره گر به اشاره گر.....
۶۳	۶-۹- آرگومان های تابع main().....

۷- ساختمان ها..... ۶۴

۶۴	۷-۱- متغیرهای ساختمان.....
۶۵	۷-۲- مقداردهی اولیه ساختمان.....
۶۵	۷-۳- آرایه ای از ساختمان.....
۶۶	۷-۴- ساختمانهای لانه ای.....
۶۶	۷-۵- اشاره گر ساختمان.....
۶۶	۷-۶- اختصاص حافظه به صورت پویا برای اشاره گر ساختمان.....
۶۷	۷-۷- ساختمانهای بیتی.....

بخش اول

الگوریتم

و

فلوچارت

مراجع

کتاب الگوریتم و فلوچارت

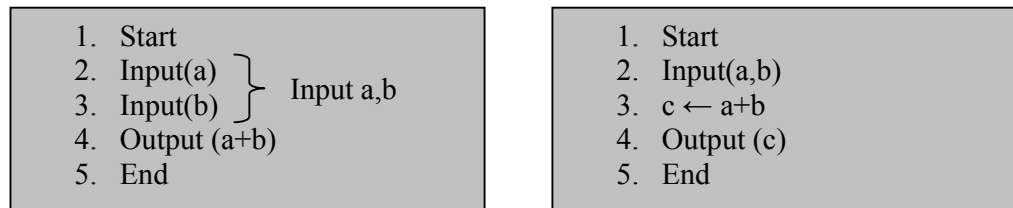
مجمع فنی تهران

قسمت اول: الگوریتم و فلوچارت

۱- الگوریتم

سلسله مراتبی از اعمال خاص می باشد که با دنبال کردن آنها کار خاصی انجام می شود. برای اینکه بتوانیم اعمالی را که انجام می دهیم به ساختاری تبدیل کنیم که امکان نوشتن برنامه ی آن وجود داشته باشد باید از الگوریتم ها استفاده کنیم.

مثال ۱: «می خواهیم الگوریتمی بنویسیم که دو عدد را مثل a,b از کاربر گرفته آنها را با هم دیگر جمع کرده و نتیجه را نمایش دهد.»



شکل ۱: دو جواب متفاوت برای مثال ۱.

۱-۱- خصوصیات الگوریتم

- هر الگوریتمی باید یک سطر start داشته باشد.
- دستورات الگوریتم باید قابل اجرا باشند.
- دستورات الگوریتم نباید مبهم باشند
- هر الگوریتم می تواند شامل یک یا چند ورودی باشد (اجباری نیست).
- هر الگوریتم باید شامل حداقل یک خروجی باشد.
- الگوریتم باید در یک سطر خاتمه (end) پیدا کند.

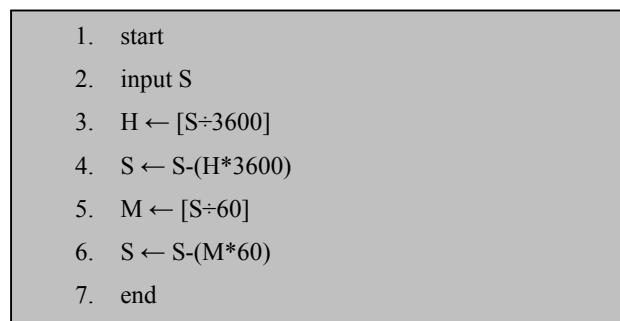
۱-۲- شرط و پرش در الگوریتم

شرط ها در الگوریتم با عبارت if مشخص می شود و می توان به یکی از دو شکل زیر از آن استفاده کرد

- If ...
- If ... Else ...

در الگوریتمها برای پرش از یک سطر به سطر بعدی می توان از Goto استفاده کرد مقابل این عبارت باید شماره سطری را که باید به آن پرش شود را مشخص کنیم که این سطر می تواند قبل یا بعد از سطر جاری باشد. معمولا در الگوریتمهایی که Goto استفاده شود If نیز استفاده می شود.

مثال ۲: الگوریتمی بنویسید که زمان را بر اساس ثانیه گرفته و مشخص کند که این زمان چند ساعت، دقیقه و ثانیه می باشد.



شکل ۲: جواب مثال ۲

مثال ۳: الگوریتمی بنویسید که معادله ی درجه ی ۲، $aX^2 + bX + c = 0$ را حل کرده و نتیجه را در خروجی نمایش دهد. راهنمایی: اگر $\Delta = b^2 - 4ac$ باشد آنگاه برای Δ می تواند سه حالت زیر وجود داشته باشد.

$$(1) \text{ اگر } \Delta > 0 \text{ باشد آنگاه } x_1 = \frac{-b + \sqrt{\Delta}}{2a}, x_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

$$(2) \text{ اگر } \Delta = 0 \text{ باشد آنگاه } x_1, x_2 = \frac{-b}{2a}$$

$$(3) \text{ اگر } \Delta < 0 \text{ باشد آنگاه معادله ریشه حقیقی ندارد.}$$

```

1. start
2. input a,b,c
3. if a=0 then goto 2
4.  $\Delta \leftarrow b^2 - 4ac$ 
5. if  $\Delta > 0$  then  $x_1 \leftarrow \frac{-b + \sqrt{\Delta}}{2a}$ ,  $x_2 \leftarrow \frac{-b - \sqrt{\Delta}}{2a}$ , output( $x_1, x_2$ )
6. if  $\Delta < 0$  then output ('not result')
7. if  $\Delta = 0$  then  $x_1, x_2 \leftarrow \frac{-b}{2a}$ , output( $x_1, x_2$ )
8. end

```

شکل ۳: الگوریتم حل معادله درجه ۲

```

1. start
2. input a,b,c
3. if a=0 then goto 2
4.  $\Delta \leftarrow b^2 - 4ac$ 
5. if  $\Delta < 0$  then goto 10
6.  $x_1 \leftarrow \frac{-b + \sqrt{\Delta}}{2a}$ 
7.  $x_2 \leftarrow \frac{-b - \sqrt{\Delta}}{2a}$ 
8. output( $x_1, x_2$ )
9. goto 11
10. output ('not result')
11. end

```

شکل ۴: الگوریتم حل معادله درجه ۲

```

1. start
2. input a,b,c
3. if a=0 then goto 2
4.  $\Delta \leftarrow b^2 - 4ac$ 
5. if  $\Delta < 0$  then output ('not result')
6. if  $\Delta \geq 0$  then output ( $x_1 \leftarrow \frac{-b + \sqrt{\Delta}}{2a}, x_2 \leftarrow \frac{-b - \sqrt{\Delta}}{2a}$ )
7. end

```

شکل ۵: الگوریتم حل معادله درجه ۲

در تمام این الگوریتمها فرض شده است که a یک عدد غیر صفر است .

۳-۱- حلقه ها در الگوریتم

حلقه ها دو نوعند:

۱. حلقه های شرطی

۲. حلقه های شمارشی

حلقه های شرطی تعداد تکرار مشخصی را ندارند و بر اساس شرایط برای تکرار و یا خاتمه تصمیم می گیرند.

حلقه های شرطی تعداد تکرار مشخصی را دارند و برای اینگونه حلقه ها یک متغیر به عنوان شمارنده حلقه در نظر گرفته می شود.

حلقه های شمارشی خود دو نوع هستند **شمارشی افزایشی** و **شمارشی کاهشی** . در حلقه های شمارشی افزایشی شمارنده حلقه از یک مقدار کوچک شروع کرده و تا یک مقدار بزرگ شمارش می کنند و در هر گام حلقه شمارنده حلقه افزایش پیدا می کند. برای حلقه های شمارشی کاهشی این روند برعکس می باشد. سه ساختار زیر می تواند برای حلقه ها شمارشی افزایشی می تواند استفاده شود.

<pre> ... m-1) I←1 m) n-1) I←I+1 n) if I≤K then Goto m ...</pre>	<pre> ... m-1) I←1 m) n) if I<K then I←I+1,Goto m ...</pre>	<pre> ... m-1) I←1 m) if I>K then Goto m . . . n-1) I←I+1 n) Goto m ...</pre>
--	--	--

شکل ۶: ساختار حلقه های شمارشی افزایشی

در این ساختارها I شمارنده حلقه، مقدار اولیه I برابر با ۱ گامهای حلقه افزایشی یک واحدی و K تعداد تکرار حلقه می باشد. رسم

این ساختارها برای حلقه های شمارشی کاهشی بر عهده دانشجو می باشد.

مثال ۴: الگوریتمی بنویسید که N عدد را از کاربر گرفته و میانگین آنها را محاسبه کند.

1. Start
2. Input N
3. $Sum \leftarrow 0$, $I \leftarrow 1$
4. Input T
5. $Sum \leftarrow Sum + T$, $I \leftarrow I + 1$
6. If $I \leq N$ then Goto 4
7. $avg \leftarrow Sum \div N$
8. Output avg
9. end

شکل ۷: جواب مثال ۴

مثال ۵ : الگوریتمی بنویسید که a^b را با استفاده از ضربهای متوالی محاسبه کند .

```

1. start
2. input (a,b)
3. if b<0 goto 2
4. c←1 , i← 1
5. if i>b then goto 9
6. c ← c × a
7. i ← i+1
8. goto 5
9. output ( c )
10. end

```

شکل ۸: الگوریتم محاسبه a^b

```

1. start
2. input a,b
3. if b<0 goto 2
4. c ←1
5. for i=1 to b do
6. c ← a × c
7. end for
8. output
9. end

```

شکل ۹: الگوریتم محاسبه a^b با استفاده از حلقه for (برای آشنایی)

مثال ۶: الگوریتمی بنویسید که $a \times b$ با جمعهای متوالی محاسبه کند.
توضیح: برای این کار دو راه حل وجود دارد می توان b تا a جمع کرد یا a تا b را با همدیگر جمع کرد.

```

1. start
2. input a,b
3. if b<0 and a<0 goto 2
4. m ← 0
5. if a=0 then Goto 9
6. m ← m+b
7. a ← a-1
8. Goto 5
9. output m
10. end

```

شکل ۱۰: الگوریتم محاسبه $a \times b$ با جمعهای متوالی

مثال ۷: الگوریتمی برای محاسبه $S = 1!+2!+3!+\dots + N!$ بنویسید.
توضیح: برای این کار دو راه حل وجود دارد راه حل اول ایجاد حلقه ای برای محاسبه هر فاکتوریل و دوم استفاده از فاکتوریل مرحله قبل برای محاسبه فاکتوریل مرحله فعلی.

$$N! = 1 \times 2 \times 3 \times \dots \times N$$

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

```

1. start
2. input N
3. if N<1 goto 2
4. S ← 0 , I ← 1
5. if I≤N then Goto 13
6. F ← 1 , J ← 1
7. F ← F * J
8. J ← J+1
9. If J≤I then Goto 7
10. S ← S+F
11. I ← I+1
12. Goto 5
13. output S
14. end

```

```

1. start
2. input N
3. if N<1 goto 2
4. S ← 0 , I ← 1 , F ← 1
5. if I≤N then Goto 10
6. F ← F * I
7. S ← S + F
8. I ← I+1
9. Goto 5
10. output S
11. end

```

شکل ۱۱: الگوریتم برای مثال ۷ با دو حالت مختلف

۲- فلوچارت

با استفاده از فلوچارت می توانیم روند کاری (الگوریتم) را با استفاده از نمادهای خاص نمایش دهیم که استفاده از این نمادها برای فهم الگوریتم کمک خواهد کرد.

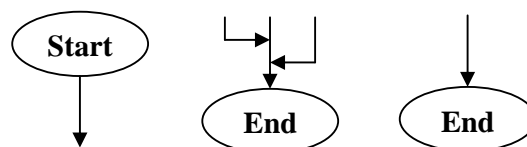
توجه: در این جزوه منظور از فلش همان جریان می باشد.

۱-۲- نمادهای فلوچارت

در فلوچارتهای به ترتیب نمادهای زیر وجود خواهد داشت:

۱-۱-۲- نماد شروع و پایان

از نماد بیضی برای شروع و پایان می توان استفاده کرد اگر نماد از نوع شروع باشد فقط یک فلش از آن خارج می شود و هیچ فلش دیگری به آن وارد نمی شود برای نماد end نیز یک فلش وارد شده و هیچ فلش دیگری از آن خارج نمی شود.

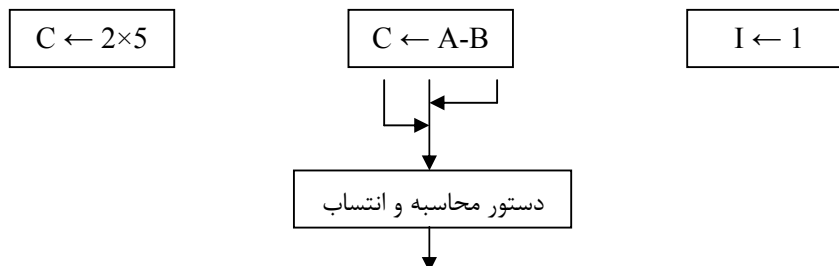


شکل ۱۲: نمادهای شروع و پایان

۲-۱-۲- نماد جایگزینی و انتساب

از نماد مستطیل برای جایگزینی و انتساب استفاده می شود این نماد میتواند چندین فلش ورودی و یک فلش خروجی را خواهد داشت. داخل این نماد می توان اعمال محاسباتی یا جایگزینی مقدار را انجام داد.

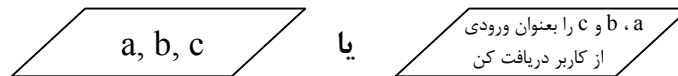
مانند:



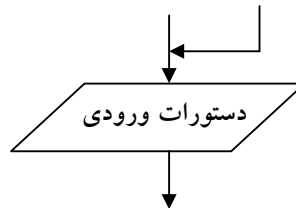
شکل ۱۳: نماد جایگزینی و انتساب

۲-۱-۳- نماد ورودی

از نماد متوازی الاضلاع برای دستور ورودی استفاده می شود برای گرفتن ورودی از کاربر می توانیم از این نماد استفاده کنیم. مثال زیر به این مفهوم است که سه عدد را از کاربر بگیرد و داخل متغیرهای a, b, c قرار دهد.



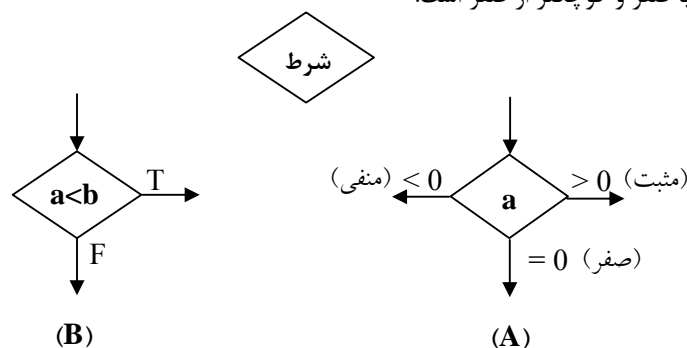
این نماد نیز میتواند چندین فلش ورودی و یک فلش خروجی داشته باشد.



شکل ۱۴: نماد ورودی

۲-۱-۴- نماد شرطی

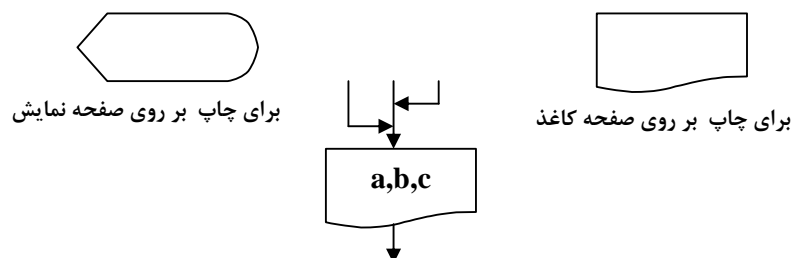
با استفاده از این نماد می توانیم شرط ها را کنترل کنیم و این بسته به شرط (درست یا نادرست) مسیر T (درست) یا F (نادرست) دنبال میشود. شکل 10-A شرط $a < b$ را کنترل می کند این شرط می تواند دو حالت داشته باشد درست یا نادرست و در هر حالت مسیری را که ادامه می دهیم با استفاده از f, t نمایش داده شده است. در 10-B مقدار متغیر a برای سه حالت کنترل می شود. حالت بزرگتر از صفر، مساوی با صفر و کوچکتر از صفر است.



شکل ۱۵: نماد شرط در دو حالت (A) با سه جریان خروجی (B) با دو جریان خروجی

۲-۱-۵- نماد چاپ (خروجی)

از این نمادها می توان برای نمایش مقادیری در روی خروجی و یا چاپ کردن استفاده کرد. به عنوان مثال شکل ۱۱ این عمل را به شکل زیر انجام می دهد. به این صورت است که محتوای سه متغیر a, b, c را در خروجی چاپ می کند.

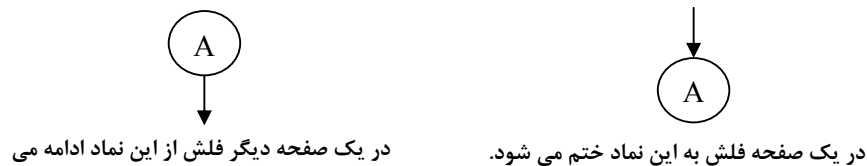


شکل ۱۶: چاپ متغیرهای a, b, c

این نماد نیز می تواند مانند اکثر نمادها می تواند چندین فلش ورودی و فقط یک فلش خروجی داشته باشد.

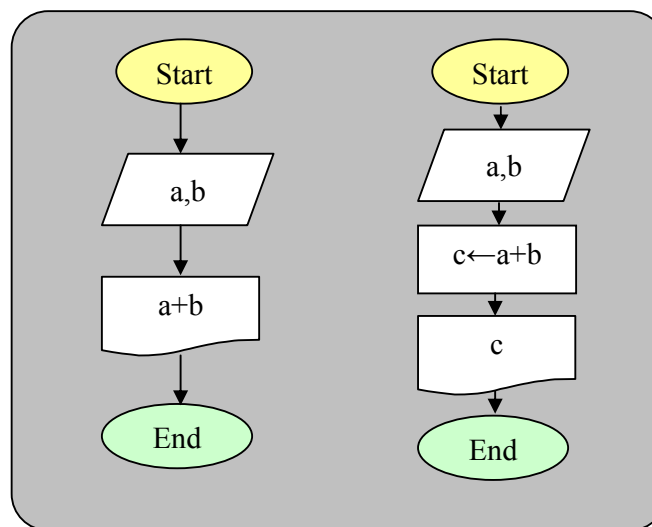
۲-۱-۶- نماد ادامه

موقعی که رسم کل مندرجات در روی یک صفحه امکان پذیر نباشد میتوانیم از این نماد برای ادامه ی مندرجات در صفحات بعدی استفاده کنیم داخل این نمادها می توانیم از هر حرف انگلیسی استفاده کنیم.



شکل ۱۷: نماد ادامه

مثال ۸: فلوچارت مثال ۱ را رسم کنید ؟



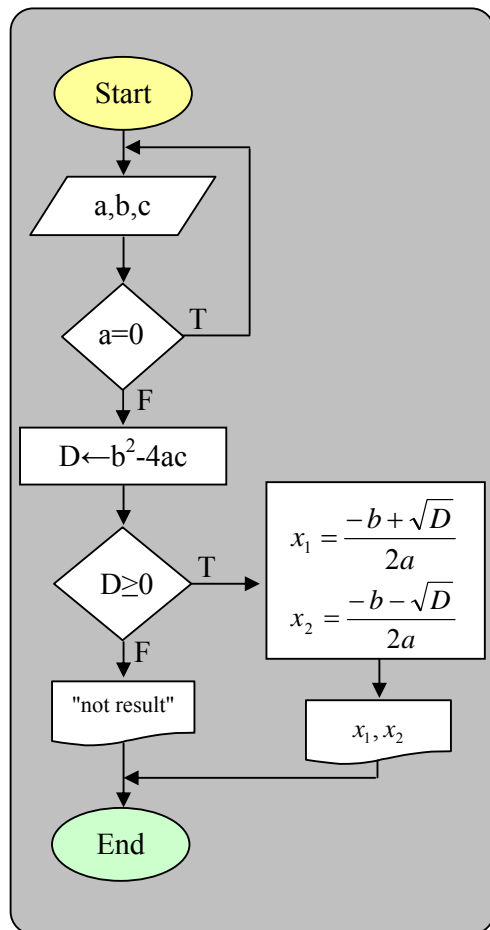
شکل ۱۸: فلوچارتهای مربوط به الگوریتمهای شکل ۱

مثال ۹: فلوچارت الگوریتم حل ax^2+bx+c را رسم کنید.

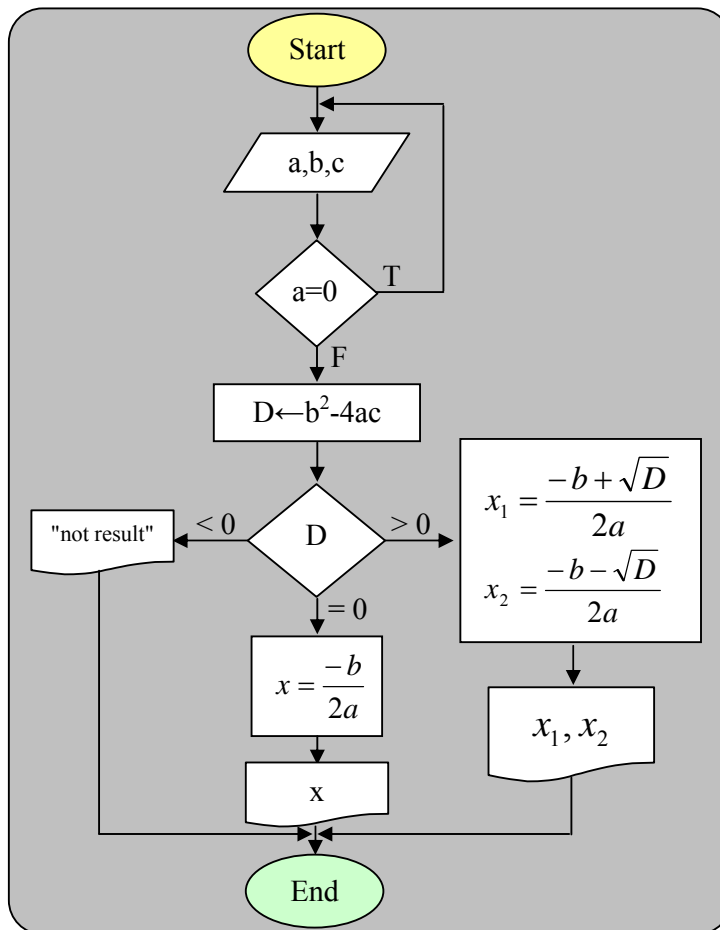
مثال ۱۰: فلوچارتی رسم کنید که سه عدد a, b, c را از کاربر گرفته سپس این اعداد را به ترتیب نزولی مرتب کند.

۲-۲- حلقه ها در فلوچارت

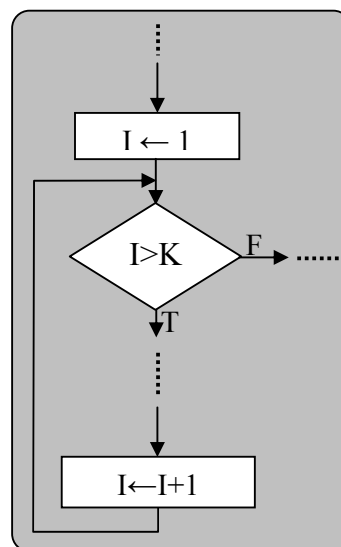
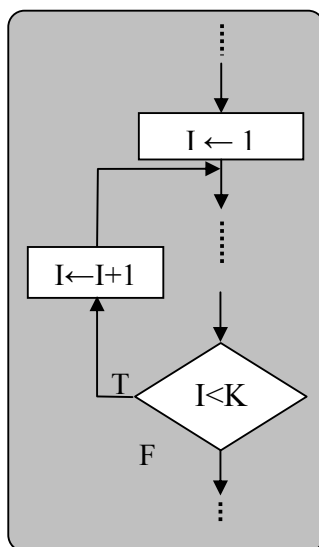
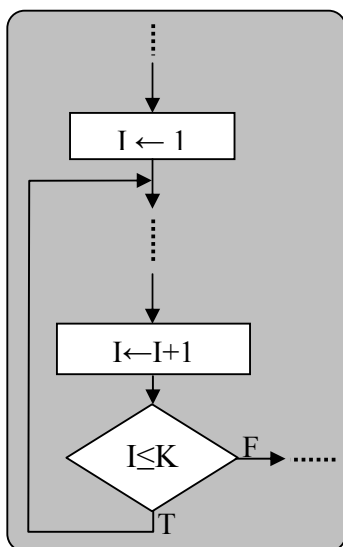
با حلقه ها در مبحث الگوریتم آشنا شدید. در این بخش قصد داریم فلوچارت مربوط به ساختارهای حلقه مطرح شده در مبحث الگوریتمها را رسم کنیم. این سه ساختار در شکل ۲۱ نشان داده شده اند. رسم بقیه ساختارها بر عهده خود دانشجو می باشد.



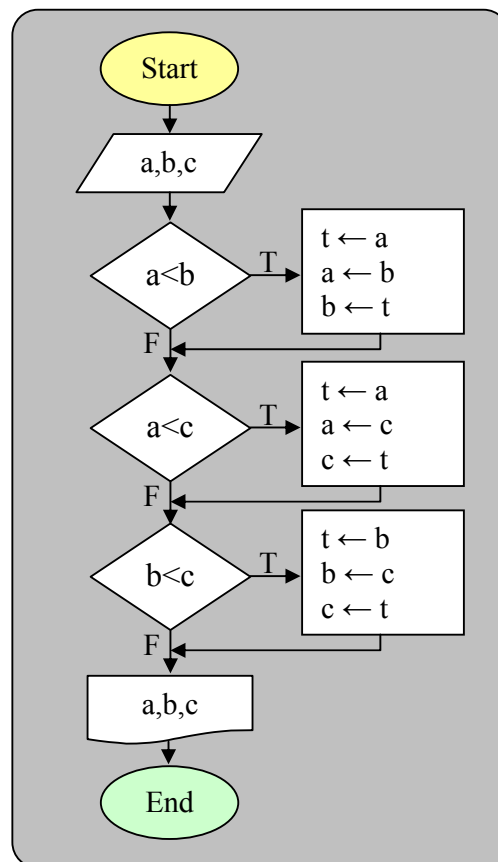
شکل ۲۰: فلوجارتهای مربوط به الگوریتم معادله درجه ۲ (شکل ۴)



شکل ۱۹: فلوجارتهای مربوط به الگوریتم معادله درجه ۲ (شکل ۲)

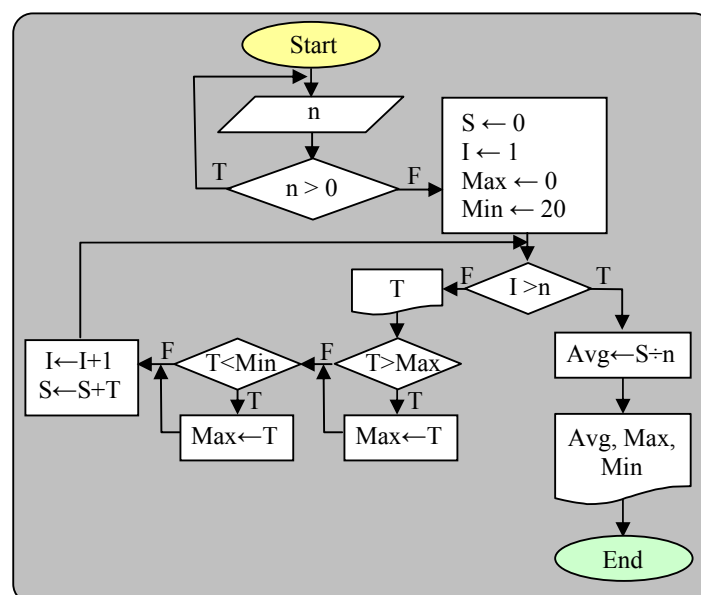


شکل ۲۱: فلوجارت مربوط به سه ساختار مختلف حلقه



شکل ۲۲: فلوچارت مثال ۶

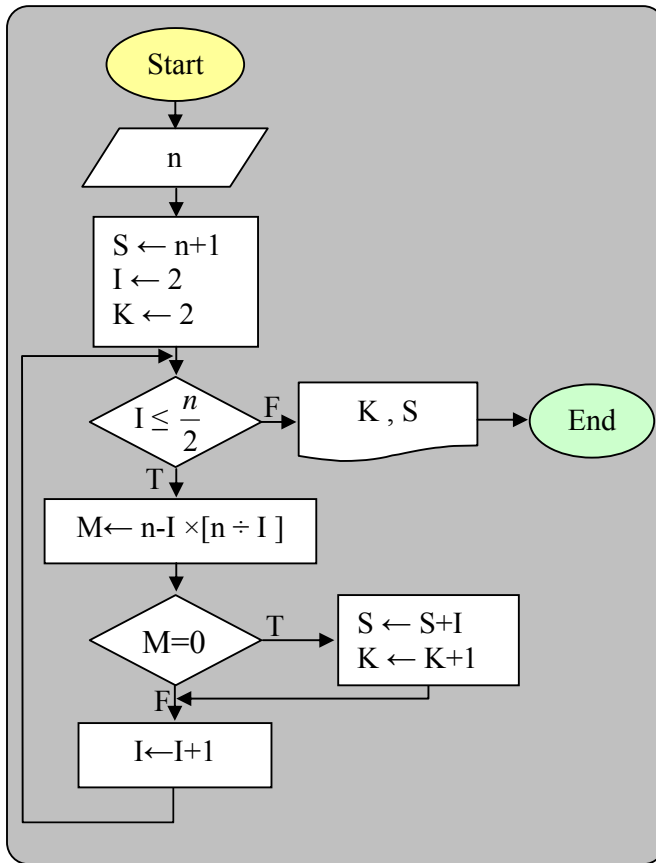
مثال ۱۱: فلوچارتی رسم کنید که نمرات دانشجویان کلاس n نفره را گرفته سپس بزرگترین، کوچکترین نمره و میانگین نمرات را محاسبه کرده و در خروجی نمایش دهد.



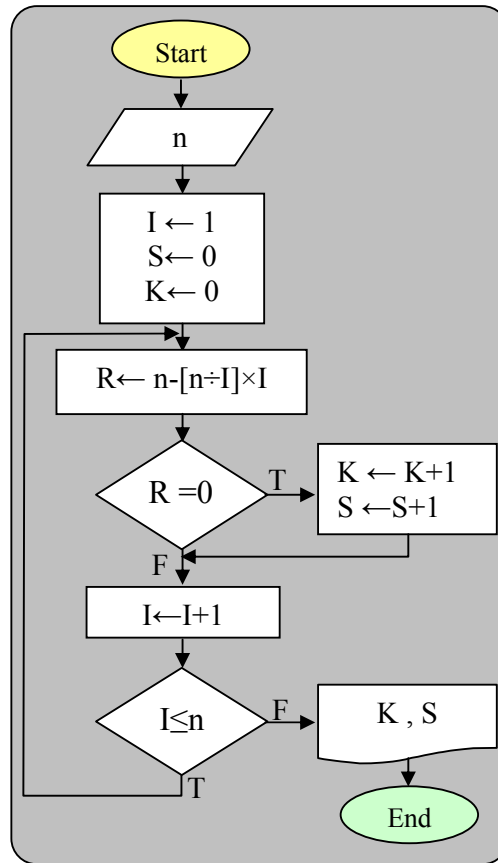
شکل ۲۳: فلوچارت مثال ۱۱

مثال ۱۲: فلوچارتی رسم کنید که تعداد و مجموع مقسوم علیه های عدد n را محاسبه کرده و خروجی را چاپ کند.

سه فلوچات مختلف با کارایی های متفاوت به ترتیب در شکل های ۲۴، ۲۵ و ۲۶ ارائه شده است.



شکل ۲۵: فلوچارت مثال ۱۲



شکل ۲۴: فلوچارت مثال ۱۲

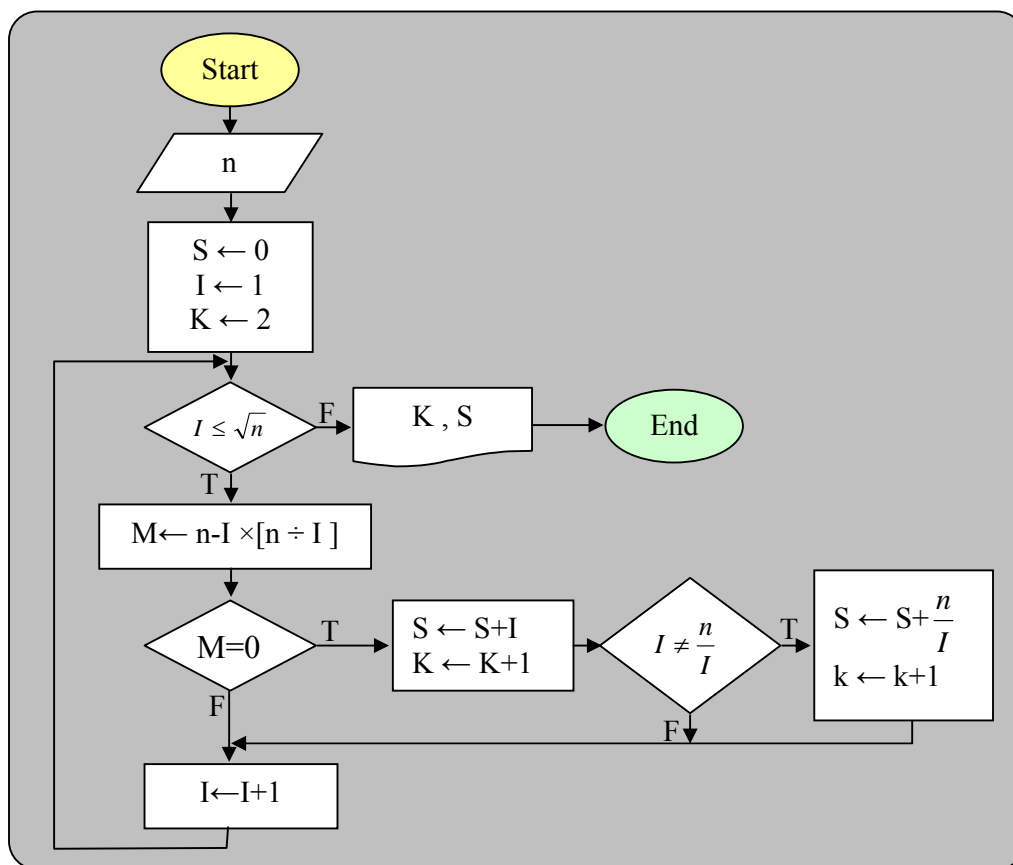
مثال ۱۳: فلوچارتی برای محاسبه جواب سری مقابل را رسم کنید.

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \dots + \frac{1}{(n-1)!} + \frac{1}{n!}$$

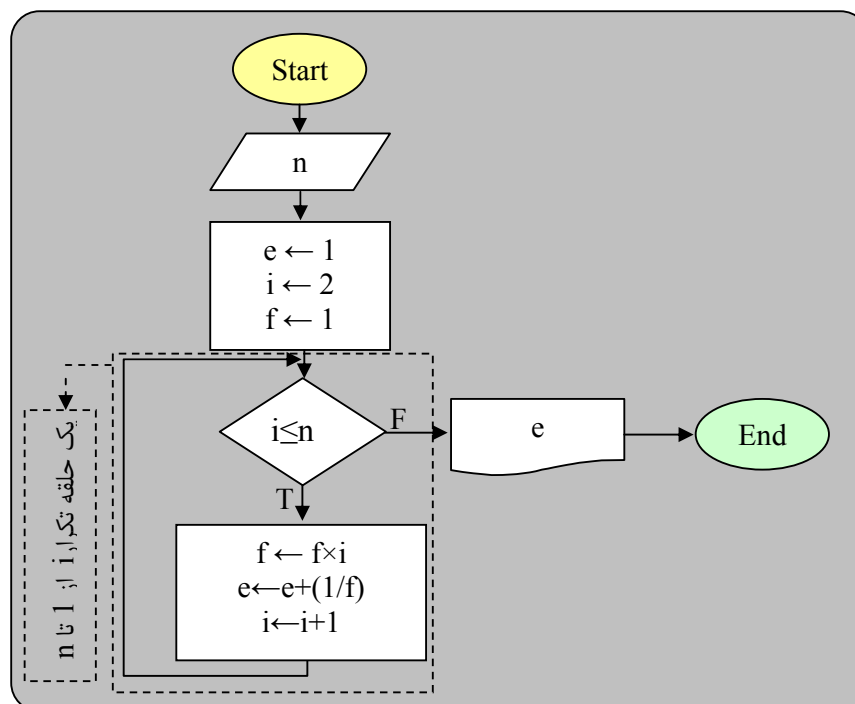
راهنمایی: برای محاسبه $n!$ می توان به یکی از دو روش زیر عمل کرد

1. $n! = 1 \times 2 \times 3 \times \dots \times n$
2. $n! = (n-1)! \times n$

شکل ۲۷ نمونه فلوچارتی را با استفاده از روش دوم نشان می دهد برای مشاهده نمونه فلوچارتهایی از روش اول می توانید به کتاب مراجعه نمایید



شکل ۲۶: فلوچارت مثال ۱۲



شکل ۲۷: فلوچارت مثال ۱۳

مثال ۱۴: فلوچارتی برای محاسبه جواب سری مقابل را رسم کنید.

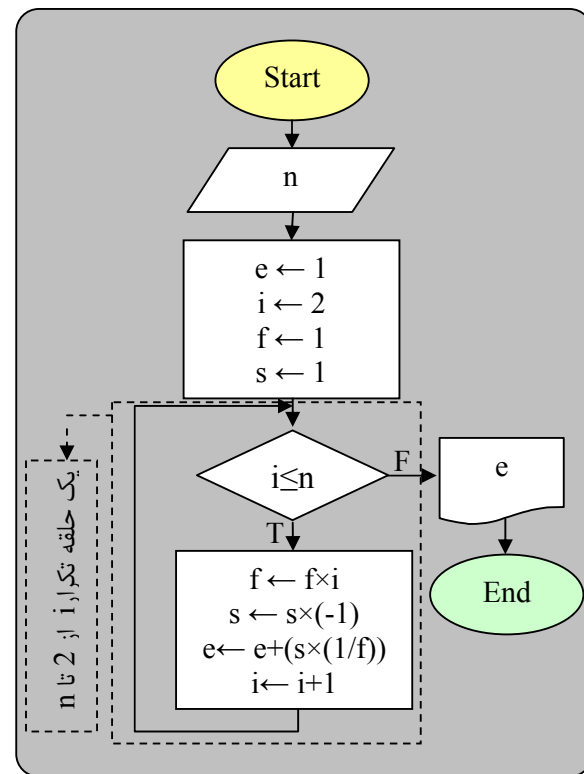
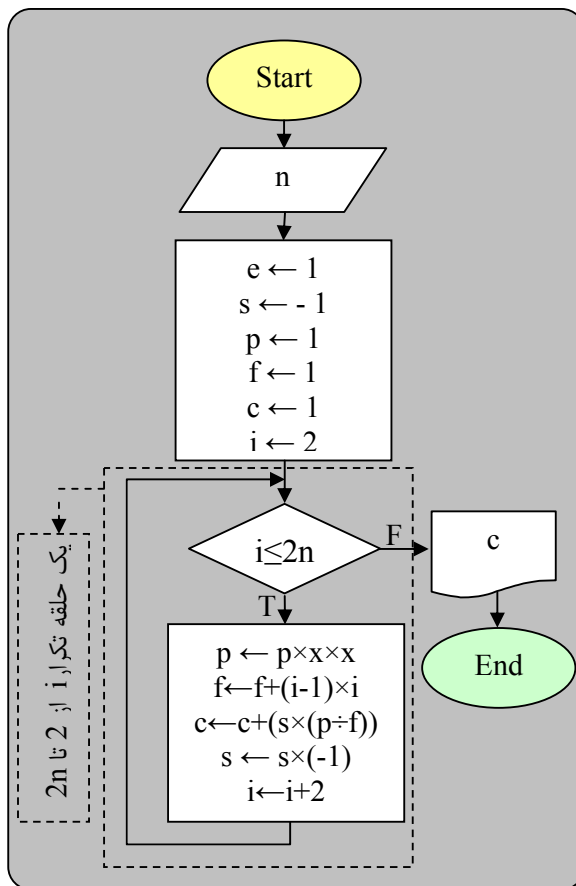
$$e = 1 - \frac{1}{2!} + \frac{1}{3!} + \dots + (-1)^{n-1} \times \frac{1}{ni}$$

مثال ۱۵: فلوچارتی برای محاسبه جواب سری مقابل را رسم کنید.

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + (-1)^n \times \frac{x^{2n}}{(2n)!}$$

راهنمایی: برای محاسبه x^n و $n!$ می توان به روش زیر عمل کرد .

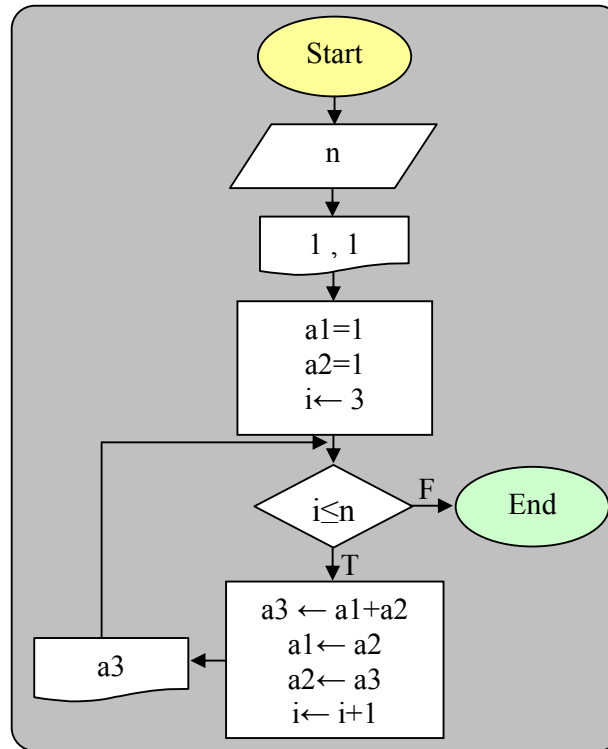
$$n! = (n-2)! \times (n-1) \times n \quad \text{و} \quad x^n = (x^{n-2}) \times x \times x$$



شکل ۲۸: فلوچارت مثال ۱۴

مثال ۱۶: فلوچارت رسم کنید که n جمله از سری فیبوناچی را تولید کند.

1, 1, 2, 3, 5, 8, 13, ...



شکل ۳۰: فلوچارت مثال ۱۶

تمرینات

۱. فلوچارتی رسم کنید که مجموع سری $S = 1! + 2! + 3! + \dots + N!$ را محاسبه کرده و در خروجی نمایش دهد. این فلوچارت را با شش ساختار مختلف حلقه پیاده سازی کنید.
۲. الگوریتم و فلوچارتی برای محاسبه مجموع ارقام عدد صحیح n طراحی کنید.
۳. الگوریتم و فلوچارتی برای محاسبه مجموع ارقام عدد حقیقی n تا چهار رقم اعشار طراحی کنید.
۴. الگوریتم و فلوچارتی برای محاسبه $a \times b$ با جمعهای متوالی طراحی کنید. برای حالت علامت دار و بدون علامت.
۵. الگوریتم و فلوچارتی برای محاسبه $a \div b$ با تفریقهای متوالی طراحی کنید. برای حالت علامت دار و بدون علامت.
۶. فلوچارتی رسم کنید که نمرات شاگردان یک کلاس n نفره را گرفته و نمره شاگرد دوم را پیدا کند.
۷. فلوچارتی رسم کنید که عددی را از کاربر گرفته و مجموع ارقام آنرا محاسبه کرده و در خروجی چاپ کند.
۸. فلوچارتی رسم کنید که عددی را از کاربر گرفته و بررسی کند که آیا با معکوس خود برابر است یا نه.

۳- آرایه ها (لیست ها)

آرایه سلولهای حافظه پشت سرهمی هستند که با یک نام شناخته می شوند ، برای دسترسی به هر یک از خانه های این آرایه می توان از اندیس آن سلول استفاده کرد برای مشخص کردن اندیس معمولاً از اعداد استفاده می شود شروع این اعداد می تواند از 0 یا 1 باشد که ما بخاطر سازگاری با زبان C شروع آنرا از 0 در نظر می گیریم . آرایه ها در بعدهای مختلفی می تواند وجود داشته باشد که بترتیب اشاره خواهد شد .

۳-۱- آرایه یک بعدی

شکل زیر نمونه ای از آرایه یک بعدی با هشت سلول و با نام A را نشان می دهد.

A نام آرایه			5					0
شماره اندیس	0	1	2	3	4	5	6	7

با اجرای دستوراتی بعنوان مثال $A(2) \leftarrow 5$ عدد 5 در اندیس 2 (خانه سوم) آرایه A قرار می گیرد همچنین با اجرای $A(7) \leftarrow 0$ عدد 0 در اندیس 7 (خانه هشتم) از آرایه A قرار می گیرد.

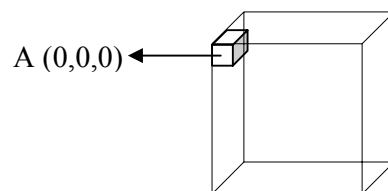
۳-۲- آرایه دوبعدی

در آرایه های دو بعدی برای دسترسی به یک خانه هم باید شماره سطر و هم ستون آنرا مشخص کرد. بعنوان مثال زمانی که می نویسیم $A(4,1)$ یعنی اینکه می خواهیم به سلولی در سطر 4 و ستون 1 دسترسی داشته باشیم

	A	0	1	2	3	4	← شماره ستون
0							
1							
2			●				→ A(2,2)
3							
4			●				
							↑ شماره سطر
							A(4,1)

۳-۳- آرایه سه بعدی

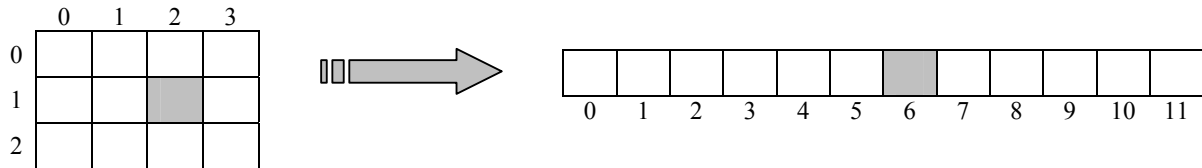
آرایه سه بعدی را می توان بصورت یک مکعب با سلولهای حافظه در نظر گرفت برای دسترسی به هر یک از خانه های آن باید به ترتیب شماره های عمق ، سطر و ستون را مشخص کنیم.



تا حال متغیرهایی که استفاده می کردیم فقط یک سلول حافظه داشتند و اگر داخل برنامه به سلول های حافظه بیشتری نیاز داشتیم مجبور بودیم برای این متغیرها سلول های حافظه ی مجزا با نام های متفاوت در نظر بگیریم، استفاده از آرایه ها این مشکل را برای ما حل کرده است.

هر نوع آرایه باید نهایتاً به یک آرایه یک بعدی نگاشت شود. نحوه نگاشت یک آرایه دو بعدی به یک آرایه تک بعدی بصورت زیر می باشد.

بعنوان مثال فرض کنید یک آرایه 3×4 بصورت شکل زیر وجود دارد برای این آرایه دو بعدی باید یک آرایه متناظر تک بعدی برای نگاشت ایجاد کنیم، در این مثال تعداد خانه های آرایه باید برابر 12 خانه ($3 \times 4 = 12$) باشد.



اگر بخواهیم بعنوان مثال خانه نگاشت شده $A(m,n)$ را در آرایه تک بعدی معادل بدست آوریم می توانیم از رابطه زیر استفاده کنیم.

$$I = (Y \times m) + n$$

در این رابطه

I = اندیس معادل در آرایه تک بعدی

Y = تعداد ستونهای آرایه دو بعدی

m و n = سطر و ستون خانه مورد نظر

بعنوان مثال اگر بخواهیم موقعیت $A(1,2)$ در شکل بالا را در آرایه تک بعدی بدست بصورت زیر عمل می شود:

$$I = (4 \times 1) + 2 \Rightarrow I = 6$$

مثال ۱۷: عدد طبیعی n مفروض است فلوچارتی رسم کنید که معادل این عدد را در مبنای ۲ بنویسد.

مثال ۱۸: فلوچارتی رسم کنید که دو بردار n بعدی (n را سؤال نماید) a, b را در نظر گرفته و مقادیر مختلفی را سؤال نموده و در خانه های a, b ذخیره نماید و سپس حاصل جمع و حاصل ضرب دو بردار a, b را به ترتیب در دو بردار c, d ذخیره نماید.

$A: A_0, A_1, A_2, \dots, A_{n-1}$

$B: B_0, B_1, B_2, \dots, B_{n-1}$

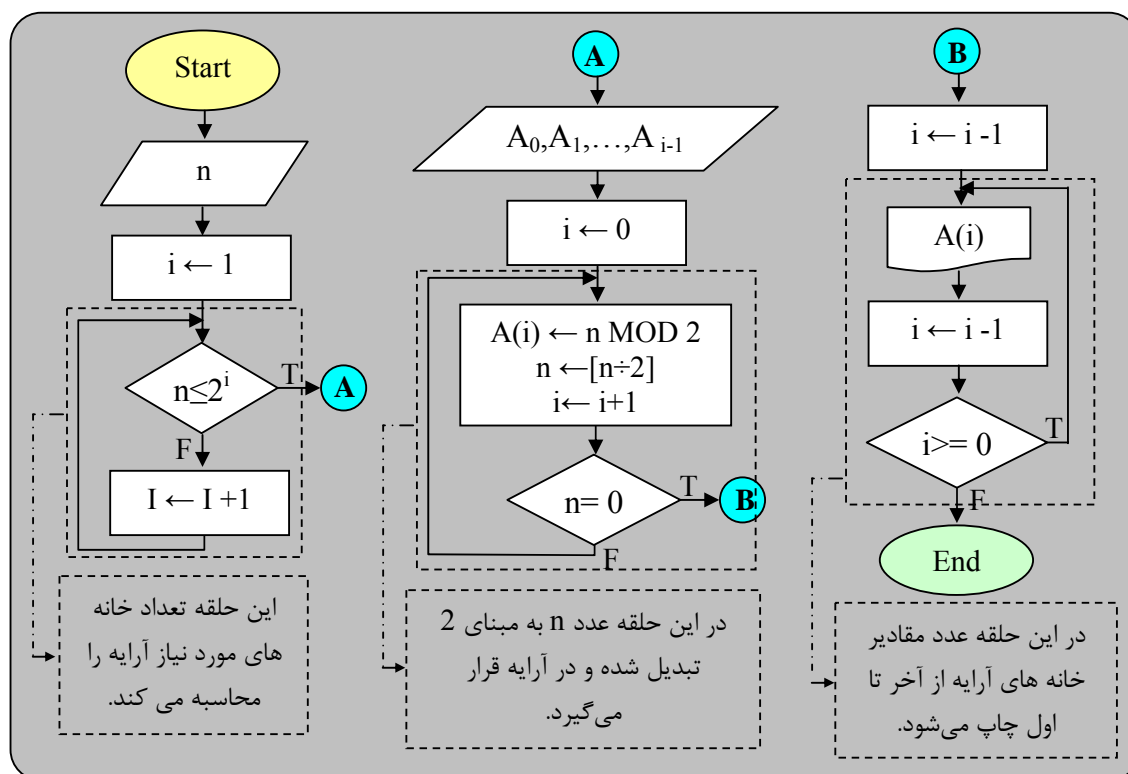
$A + B = (A_1 + B_1, A_2 + B_2, \dots, A_n + B_n)$

$A - B = (A_1 - B_1, A_2 - B_2, \dots, A_n - B_n)$

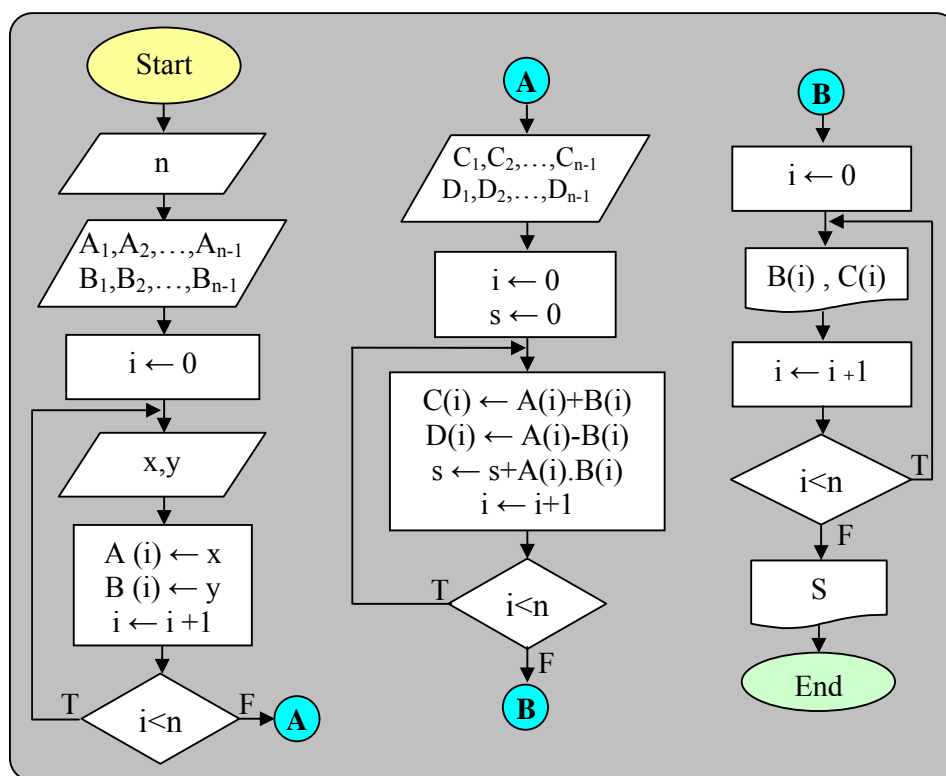
$A \cdot B = (A_1 \cdot B_1 + A_2 \cdot B_2 + \dots + A_n \cdot B_n)$

مثال ۱۹: فلوچارتی رسم کنید که n جمله از سری فیبوناچی را تولید کرده در داخل آرایه قرار داده سپس آنها را در خروجی چاپ کند.

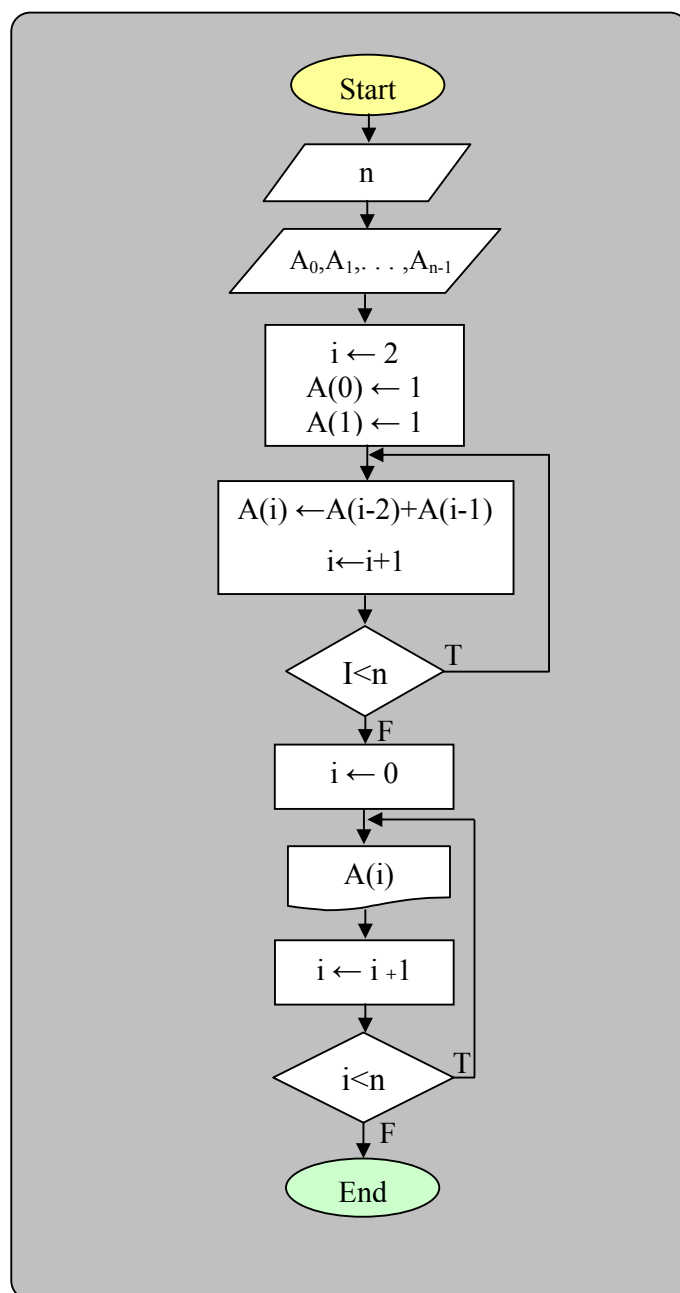
مثال ۲۰: فلوچارتی رسم کنید که عدد طبیعی N را سؤال و N خانه در نظر گرفته و N عدد دلخواه را یکی یکی سؤال نموده و داخل آنها قرار دهد و ابتدا میانگین آنها را محاسبه کند سپس مشخص کند چند عدد از میانگین بزرگتر و چند عدد کوچکتر و چند عدد با میانگین برابر است.



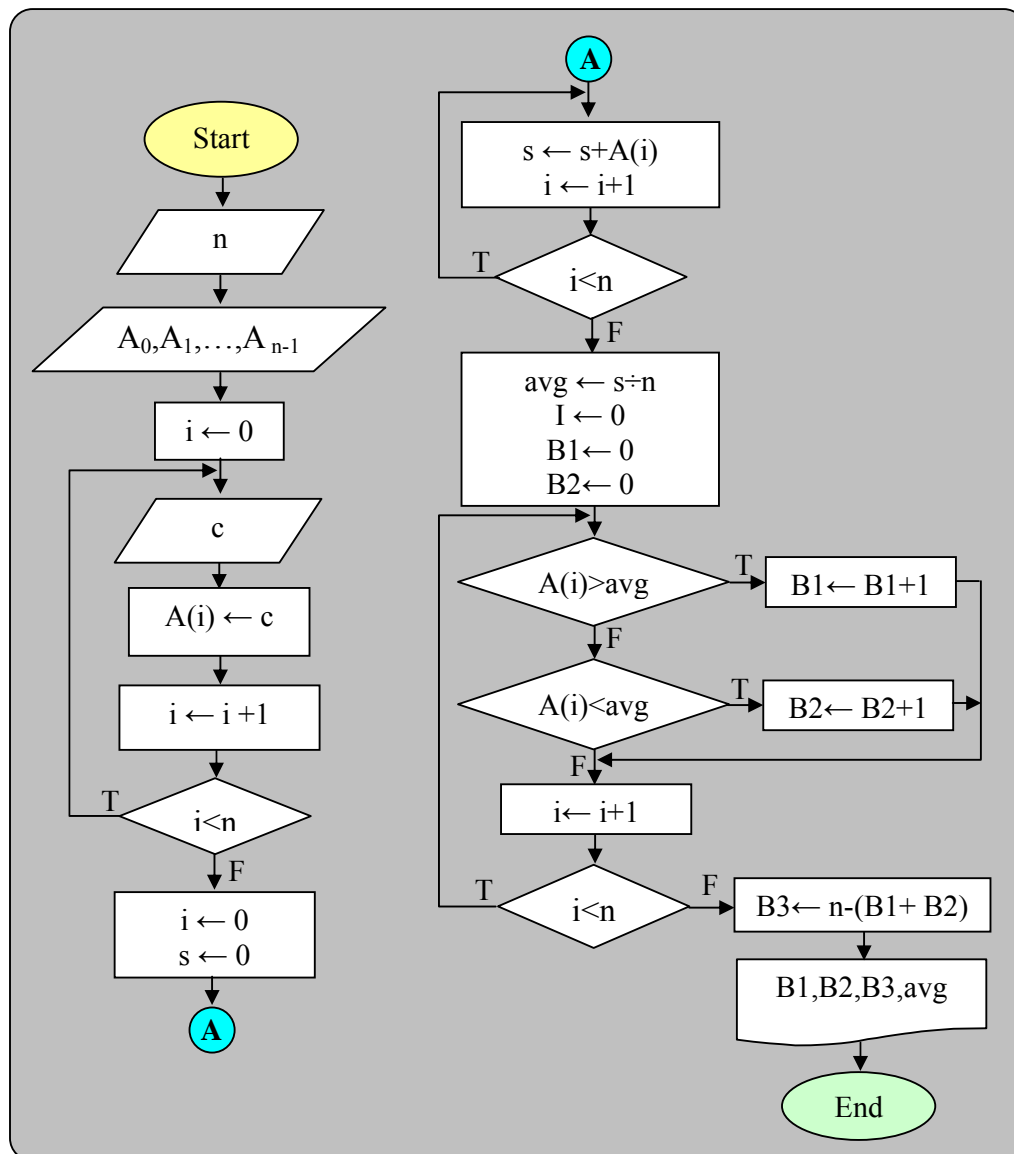
شکل ۳۱: فلوچارت مثال ۱۷



شکل ۳۲: فلوچارت مثال ۱۸



شکل ۳۳: فلوچارت مثال ۱۹



شکل ۳۴: فلوچارت مثال ۲۰

بخش دوم

برنامه نویسی به زبان

C

مرجع کتاب

برنامه نویسی به زبان C

تالیف عین اله جعفر نژاد قمی

قسمت دوم: برنامه نویسی به زبان C

۱- مقدمات زبان C

۱-۱- کلمات کلیدی

کلماتی هستند که برای کامپایلر (compiler) زبان C مفهوم خاصی دارند و از آنها نمی توان برای اهداف دیگر مانند نام متغیر استفاده کرد. برخی از این کلمات عبارتند از: int ، for ، else ، switch و ...
توجه: کل این کلمات در جدول ۱-۳ و ۱-۴ کتاب لیست شده است

۱-۲- Comment یا توضیحات

در داخل برنامه ها اگر بخواهیم توضیحاتی راجع به بخش هایی از برنامه بنویسیم می توانیم از Comment استفاده کنیم.
Comment در زبان C با دو حالت زیر مشخص می شود.

1. /* This is a sample comment */
2. //This is a sample comment

در حالت اول هر عبارتی که مابین /* و */ قرار گیرد به عنوان Comment در نظر گرفته شده و هیچ تأثیری در اجرای برنامه نخواهد داشت.

در حالت دوم هر عبارتی که بعد از نماد // قرار گیرد در همان سطر به عنوان Comment در نظر گرفته می شود در حالت اول توضیحات می تواند روی یک و یا بیش از یک سطر نوشته شود اما در حالت دوم توضیحات فقط باید روی یک سطر باشد.

۱-۳- انواع داده ای

در زبان C پنج نوع داده ای متفاوت char , int , float, double, void وجود دارد که char برای مشخص کردن نوع داده ی کاراکتری ، int برای مشخص کردن نوع داده ی صحیح ، float برای نوع داده ی اعشاری، double برای مشخص کردن عدد اعشاری با دقت بیشتر استفاده می شود void کاربرد خاصی دارد بعداً به آن اشاره خواهیم کرد void نوع داده ی بی ارزش است.
همچنین با استفاده از این انواع داده های و توصیف کننده های signed (با علامت)، unsigned (بدون علامت)، long و short می توان انواع داده ای جدیدی تعریف کرد. که همراه با char توصیف کننده های signed و unsigned استفاده می شود همراه با int هر چهار توصیف کننده و همراه با double فقط long استفاده می شود. جدول ۱-۵ کتاب انواع داده ای مختلف را همراه با مقادیر قابل قبول آن نشان می دهد.

بسته به نوع داده ای انتخاب شده فضای تعریف شده اختصاص داده می شود. بعنوان مثال برای نوع داده ای char

char 1 byte \rightarrow 8 bit $\rightarrow 2^8 = 256$ -128 الی 127

'A'=65 کداسکی

میزان فضای اختصاصی برای هر متغیر و همچنین کل لنوع داده ای زبان C در جدول ۱-۵ کتاب لیست شده است.

۱-۴- خصوصیات دستورالعمل در زبان C

۱. هر دستور زبان C باید به ; ختم شود.
۲. حداکثر طول یک دستور ۲۵۵ کاراکتر می باشد.
۳. هر دستور می تواند روی یک سطر و یا بیش از یک سطر ادامه داشته باشد.
۴. در هر سطر می توان بیش از یک دستور را تایپ کرد.

۵-۱- متغیرها

برای تعریف متغیرها در زبان C ما می توانیم از انواع داده ی استاندارد زبان C استفاده کنیم هم چنین برای هر متغیری باید یک شناسه نیز در نظر گرفته شود که برای تعریف این شناسه می توانیم از قوانین تعریف شناسه زبان C استفاده کنیم.

قانون ۱: برای تعریف شناسه ها می توانیم از ترکیب حروف و یا اعداد و هم چنین کاراکتر underline استفاده کنیم.

اعداد: '0' '9' حروف کوچک: 'a' 'z'

حروف بزرگ: 'A' 'Z' _ : UNDERLINE

قانون ۲: شروع نام شناسه حتما باید با یک حرف باشد. مانند A1

قانون ۳: نام یک شناسه نمی تواند برابر یکی از کلمات رزرو شده باشد.

شکل کلی تعریف متغیر در زبان C بصورت زیر می باشد.

نام متغیر نوع دادهای

برای تعریف متغیرها ابتدا نوع متغیر مشخص می شود سپس نام متغیر به عنوان مثال :

```
int a;
```

هنگام تعریف نام متغیر می توانیم بیش از یک نام را مشخص کنیم در این حالت باید مابین هر یک از متغیرها از کاراکتر کاما « , » استفاده کرد.

```
int A,B,C;
```

در زیر روش دیگری برای تعریف سه متغیر A,B,C از نوع INT (int) نشان می دهد.

```
int A;
```

```
int B;
```

```
int C;
```

برای مقدار دهی متغیرها می توان به روشهای زیر عمل کرد:

۱. مقداردهی متغیر هنگام تعریف آن : بعنوان مثال `int A=5;` عدد ۵ را به متغیر A هنگام تعریف آن اختصاص می دهد.

۲. با استفاده از عملگر انتساب : بعنوان مثال `A=10` عدد 10 را به متغیر A انتساب می دهد.

۳. با استفاده از دستورات ورودی : یکی از دستورات ورودی `scanf` می باشد. که می توانیم با استفاده از این دستورات

متغیرها را مقداردهی کنیم. بعنوان مثال `scanf("%d",&a);` از صفحه کلید مقداری را گرفته بر روی متغیر a آنرا

ذخیره می کند.

توجه: در کامپایلر زبان C مابین حروف کوچک و حروف بزرگ تفاوت وجود دارد پس سه عبارت زیر با همدیگر برابر نیستند.

Int int INT

۶-۱- تعریف ثابت

برای تعریف ثابت در زبان C دو راه وجود دارد.

۱. استفاده از دستور `#define`

۲. استفاده از دستور `const`

مثال زیر نمونه ای از تعریف ثابت با استفاده از دستور `#define` را نمایش می دهد. ساختار کلی دستور `#define` از سه بخش

تشکیل شده است.

```
#define PI 3.14
```

بخش اول عبارت `#define` را مشخص می کند. بخش دوم نام ثابت و بخش سوم مقدار را مشخص می کند.

تذکر: در انتهای دستور `#define` نباید از کاراکتر `;` استفاده کرد.

شکل کلی دستور `const` برای تعریف ثابت بصورت زیر می باشد:

`const` <مقدار> = <نام ثابت> <نوع داده>

```
const float PI=3.14;
```

۷-۱- عملگرها

در زبان C عملگرها به چهار دسته عملگرهای محاسباتی، عملگرهای رابطه ای، عملگرهای منطقی و عملگرهای بیتی تقسیم بندی می شود.

۷-۱-۱- عملگرهای محاسباتی

در زبان C عملگرهای محاسباتی وجود دارد که به ترتیب عبارتند از

عملگر	مثال	توضیحات
+	a+b	عملگر پلاس (+) عمل جمع محاسباتی را انجام می دهد.
-	a-b یا -a	عملگر ماینس (-) عمل تفریق محاسباتی را انجام می دهد.
*	a*b	عملگر استار (*) عمل ضرب محاسباتی را انجام می دهد.
/	a/b	عملگر اسلش (/) عمل تقسیم محاسباتی را انجام می دهد.
%	a%b	عملگر درصد (%) عمل باقیمانده تقسیم را انجام می دهد.
++	a++ یا ++a	عملگر پلاس پلاس (++) عمل افزایش یک واحدی را انجام می دهد و در دو حالت استفاده می شود
--	a-- یا --a	عملگر ماینس ماینس (--) عمل کاهش یک واحدی را انجام می دهد و در دو حالت استفاده می شود

توضیح: عملگر ++ و -- می تواند به دو حالت برای متغیرها اعمال شود قبل از نام متغیر (سمت چپ) قرار بگیرد یا بعد از نام متغیر (سمت راست). اگر این عملگر قبل از نام متغیر قرار بگیرد ابتدا افزایش یک واحدی (یا کاهش یک واحدی) انجام شده سپس محاسبه انجام می گیرد اما اگر بعد از نام متغیر قرار بگیرد ابتدا محاسبه انجام شده سپس افزایش یا کاهش یک واحدی صورت می گیرد. در سه مثال زیر در مثال اول هیچ تفاوتی مابین دو حالت وجود ندارد اما مابین مثال ۲ و مثال ۳ تفاوت وجود دارد بین حالتی که این عملگرها قبل از نام متغیر قرار بگیرند یا بعد از نام متغیر قرار بگیرند.

مثال ۱:

<code>int a=5;</code>	<code>int a=5;</code>
<code>++a;</code>	<code>a++;</code>
(الف)	(ب)

در این مثال هیچ تفاوتی مابین حالت الف و ب وجود ندارد و در هر دو حالت روی متغیر a عدد ۶ قرار خواهد گرفت.

مثال ۲:

<code>int a=5,b;</code>	<code>int a=5,b;</code>
<code>b=a++;</code>	<code>b=++a;</code>
(الف)	(ب)

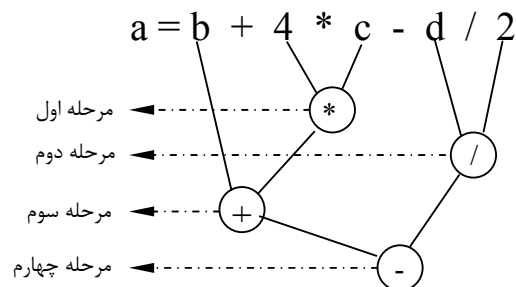
ولی در این مثال مابین حالت الف و ب تفاوت وجود دارد. در حالت الف پس از اجرای سطر دوم $b=5$ ، $a=6$ خواهد بود ، چون اول مقدار a داخل b قرار گرفته سپس a یک واحد افزایش می یابد. ولی در حالت ب پس از اجرای سطر دوم $b=6$ ، $a=6$ خواهد بود ، چون اول a یک واحد افزایش می یابد سپس مقدار a داخل b قرار می گیرد.

تقدم این عملگرها به صورت زیر مشخص می شود.

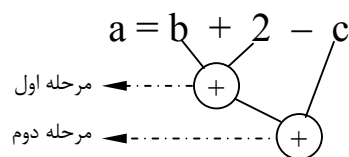
++ , --	بالاترین تقدم
- (منهای یکانی)	
*, / , %	
+, -	پایین ترین تقدم

مثال: باتوجه به تقدم عملگرهای محاسباتی مراحل محاسبه در فرمول زیر را نشان دهید؟

$$A = b + 4 * c - d / 2$$



توجه: اگر یک عبارت عملگرهایی با اولویت یکسان داشته باشد در این حالت اولویت اجرا از سمت چپ به سمت راست مشخص خواهد شد.



۱-۷-۲- عملگرهای رابطه ای

مثال	نام	عملگر
$a > b$	بزرگتر	$>$
$a \geq b$	بزرگتر یا مساوی	\geq
$a < b$	کوچکتر	$<$
$a \leq b$	کوچکتر یا مساوی	\leq
$a == b$	متساوی	$==$
$a != b$	نامساوی	$!=$

۱-۷-۳- عملگرهای منطقی

مثال	نام	عملگر
$!a$	نقیض (not)	$!$
$a > b \ \&\& \ c < d$	و (and)	$\&\&$
$a > b \ \ c < d$	یا (or)	$ $

تقدم سه عملگر محاسباتی، رابطه ای و منطقی:

!	بالاترین تقدم
>, <, >=, <=	
=, !=	
&&	
	پایین ترین تقدم

۱-۷-۴- عملگرهای ترکیبی

عملگر	نام	مثال	معادل
+=	انتساب جمع	a+=b	a=a+b یا a=b+a
-=	انتساب تفریق	a-=b	a=a-b
=	انتساب ضرب	a=b	a=a*b یا a=b*a
/=	انتساب تقسیم	a/=b	a=a/b
%=	انتساب باقیمانده تقسیم	a%=b	a=a%b

۱-۷-۵- عملگرهای بیتی

نام	عملگر
and	&
or	
xor	^
not	~
right shift	>>
left shift	<<

عملکرد هر یک از عملگرهای بیتی بصورت زیر می باشد.

a	b	a & b	a b	a ^ b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

مثال: نمونه ای از عملگر بیتی بصورت زیر می باشد.

01100101	00010111	01101111	01100101
00100110	01010010	01111001	
&		^	~
00100100	01010111	00010110	10011010

جدول ۱-۱۵ کتاب نمونه مثالی را از عملگرهای بیتی shift left و shift right را نشان می دهد. شکل کلی این عملگرها بصورت

زیر می باشد.

تعداد شیفت >> متغیر

تعداد شیفت << متغیر

این عملگرها کل بیت های عدد به تعداد مشخص شده به سمت راست یا به سمت چپ شیفت داده خواهند داد.

این عملگرها بر روی تک تک بیت های عدد عمل خواهند کرد. در زبان C ۶ نوع عملگر بیتی وجود دارد که در بالا نوشته شد.

توجه: شیفت ۱ واحد به سمت راست عمل تقسیم بر ۲ را انجام خواهد داد و شیفت ۱ واحد به سمت چپ همان عمل ضرب در ۲ را انجام خواهد داد.

۱-۷-۶- تقدم عملگرها در حالت کلی

جدول زیر تقدم عملگرها را در حالت کلی نشان می دهد.

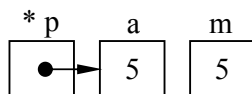
()	بالاترین تقدم
! ~ ++ -- sizeof	
* / %	
+ -	
<< >>	
< <= >= >	
== !=	
&	
^	
&&	
?	
= += -= *= /= %=	پایین ترین تقدم

۱-۷-۷- عملگرهای * و &

این عملگرها بیشتر در بخش مربوط به اشاره گرها توضیح داده خواهند شد. عملگر & برای واکشی (استخراج) آدرس یک متغیر و عملگر * برای دسترسی به یک آدرس استفاده می شود.

مثال زیر نمونه ای از کاربرد این عملگرها را نشان می دهد. در سطر دوم این مثال آدرس متغیر a با استفاده از عملگر & داخل اشاره گر p قرار می گیرد و در سطر سوم داخل خانه ای که p به آن اشاره دارد (با استفاده از عملگر *) عدد ۵ قرار خواهد گرفت و در سطر ۴ محتوای خانه ای که p به آن اشاره دارد داخل متغیر m قرار خواهد گرفت.

```
1. int a,*p,m;
2. p=&a;
3. *p=5;
4. m=*p;
```



۱-۷-۸- عملگر ؟

شکل کلی عملگر ؟ از سه بخش بصورت زیر تشکیل شده است.

<عبارت ۳> : <عبارت ۲> ؟ <عبارت ۱> = متغیر

بخش اول یک شرط را کنترل می کند در صورت درست بودن شرط مقدار عبارت دوم داخل متغیر قرار می گیرد در غیر این صورت مقدار عبارت سوم داخل متغیر قرار خواهد گرفت در مثال زیر شرط $a > b$ بررسی می شود در صورتی که شرط برقرار باشد مقدار متغیر b داخل متغیر a قرار می گیرد در غیر این صورت مقدار $a - b$ داخل متغیر a قرار خواهد گرفت.

A = A > B ? B : A - B;

۱-۷-۹- عملگر sizeof

این عملگر میزان فضای اختصاصی داده شده به یک متغیر را با یک نوع داده ای براساس بایت را مشخص می کند در مثال زیر اگر نوع داده ی متغیر a از نوع کاراکتری باشد عدد ۱ داخل s و اگر از نوع int باشد عدد ۲ داخل s و ... قرار خواهد گرفت. دستور sizeof دوم نیز میزان فضای حافظه مورد نیاز برای نوع داده ی float را مشخص کرده و داخل متغیر s قرار می گیرد. (عدد ۴)
مثال ؟ :


```
s=sizeof a;          s=sizeof(float);
```

۸-۱- تبدیل انواع داده‌ای

```
char ch;
int a;
float b;
double c;
b = ( ch/a )+( b+c )+( b+a );
```

در این مثال همچنان که مشاهده می‌کنید متغیرهای زیادی با انواع داده‌ای متفاوت شرکت دارند برای اینکه این محاسبه انجام شود باید انواع داده‌ای به همدیگر تبدیل شوند تا امکان انجام محاسبه وجود داشته باشد.

در زبان C تبدیلات نوع در دو حالت صورت می‌گیرد:

۱. تبدیلات نوع در محاسبات

تبدیل نوع در محاسبات از قانون زیر پی‌روی می‌کند:

"نوع داده‌ای یک محاسبه بر اساس بزرگترین نوع داده‌ای مشخص می‌شود"

اندازه انواع داده‌ای به ترتیب از کوچک به بزرگ عبارتند از char, int, float و double. یعنی اگر در یک محاسبه حتی یک double وجود داشته باشد نتیجه از نوع double خواهد بود.

۲. تبدیلات نوع در انتساب

جدول ۱۷-۱ کتاب تبدیل انواع در احکام انتساب را نشان می‌دهد.

به عنوان مثال اگر بخواهیم unsighned char را به یک unsighned int تبدیل کنیم روش تبدیل به این صورت خواهد بود که یک بایت متغیر کاراکتری داخل بایت کم ارزش متغیر intiger قرار گرفته و به جای بایت پردازش عدد صفر قرار داده شود اگر در تبدیلات نوع میزان حافظه مبدأ کوچکتر مساوی میزان حافظه مقصد باشد هیچ داده‌ای از بین نخواهد رفت اما در غیر این صورت ممکن است در تبدیلات نوع برخی از داده‌ها از بین بروند.

برای تبدیلات نوع قوانینی وجود دارد که در کتاب بصورت کامل اشاره شده است. اما بصورت خلاصه می‌توان گفت همیشه داده با میزان حافظه کوچک به نوع داده‌ای با حافظه بزرگ تبدیل می‌شود (حق تقدم تبدیل با نوع داده‌ای حافظه بزرگ می‌باشد).

۲- ساختار برنامه زبان C و ورودی/خروجی

مثال زیر نمونه ای از برنامه ی زبان C را نمایش می دهد در سطر اول این برنامه با استفاده از دستور #include فایل سرآیند(هدر فایل) مورد استفاده قرار گرفته (مانند stdio.h) مشخص می شود نام هدر فایل مابین دو نماد < > قرار می گیرد که نباید هیچ فاصله ای مابین این نمادها و نام فایل وجود داشته باشد.

در سطر دوم تابع () main معرفی شده تابع () main اصلی ترین تابع هر برنامه ی زبان C می باشد که معمولاً اجرای برنامه ها از آن تابع شروع می شود در سطر ۳ () { آغاز بدنه تابع و سطر ۷ () } پایان بدنه تابع را مشخص می کند مابین این سطرها مشخص کننده ی دستورات اجزایی تابع می باشد یکی از اجزای تابع بخش تعریف متغیرهای آن می باشد که در این مثال سطر ۴ این عمل را انجام می دهد بخش دیگر توابع دستورات اجرایی می باشد که در این مثال سطر ۵ برای این منظور داده شده است در تابع main دستور return 0 نیز وجود دارد که در این مثال سطر ۶ این عمل را انجام می دهد.

1. #include <stdio.h>
2. int main (void)
3. {
4. تعریف متغیر
5. دستورات اجرایی
6. return 0;
7. }

۲-۱- دستورات ورودی و خروجی

دستورات ورودی و خروجی روی ابزارهای ورودی استاندارد (صفحه کلید) و خروجی استاندارد (صفحه نمایش) عمل خواهند کرد. در زبان C دستورات ورودی و خروجی متنوعی وجود دارد که ما به آن اشاره خواهیم کرد.

۲-۱-۱- تابع () printf

تابع () printf که در فایل stdio.h قرار دارد برای چاپ در خروجی استاندارد استفاده می شود. بدنه کلی دستور () printf از دو بخش تشکیل شده است. شکل کلی تابع printf بصورت زیر میباشد.

printf("<عبارت ۱>,<عبارت ۲>");

عبارت ۱: عبارت چاپی را مشخص میکند.

عبارت ۲: داخل عبارت ۱ برخی کاراکترهای فرمت وجود دارد در این عبارت متغیرهایی را مشخص میکنیم که قرار است داده ها و آنها بجای کاراکترهای فرمت چاپ شود.

کاراکتر	نوع اطلاعاتی که باید به خروجی برود
%d	برای چاپ اعداد صحیح
%f	برای چاپ اعداد اعشاری
%s	برای چاپ رشته ای از کاراکترها
%c	برای چاپ یک کاراکتر
%./	برای چاپ کاراکتر درصد

جدول ۲-۱ کتاب بصورت کامل کل این کاراکترها را لیست کرده است.

داخل عبارت ۱ سه نوع کاراکتر می تواند قرار بگیرد.

۱. کاراکترهای معمولی: مانند حروف انگلیسی، ارقام، #، @ و

۲. **کاراکترهای کنترلی:** این کاراکترها شکل خروجی اطلاعات را مشخص می کنند. اینکه آیا اطلاعات در یک سطر باشند یا در چند سطر چاپ شوند، آیا اطلاعات با فاصله خاصی از یکدیگر چاپ شوند یا خیر و مواردی از این قبیل، توسط کاراکترهای کنترلی مشخص می شود. کاراکترهای کنترلی با (\) آغاز می شوند. به عنوان مثال \n موجب می شود تا سطر جاری (سطری که فعلا در حال نوشتن در آن سطر هستیم)، رد شود و چاپ اطلاعات از سطر بعدی آغاز گردد. \f باعث انتقال کنترل به صفحه جدید می شود. \t مکان نما را به ابتدای همان سطر انتقال می دهد، \b مکان نما را به ستون قبلی انتقال می دهد. جدول ۲-۲ کتاب بصورت کامل این کاراکترهای کنترلی را نشان می دهد.
۳. **کاراکترهای فرمت:** این کاراکترها نوع اطلاعاتی را که در عبارت ۲ قرار دارند و باید بجای این کاراکتر چاپ شوند، مشخص می کند. کاراکترهای فرمت با علامت % شروع می شوند. کاربردترین کاراکترهای فرمت در جدول ۲-۱ نشان داده شده اند.

```
printf("%d*d%"+\"A\"-\"string\" ",A,B);
```

برای **پاکسازی صفحه تصویر** می توانیم از تابع (clrscr) که در فایل conio.h قرار دارد استفاده کنیم. همچنین برای **انتقال مکان نما** در صفحه تصویر می توانیم از دستور gotoxy(int x,int y) که در فایل conio.h قرار دارد استفاده کنیم. این تابع شامل دو پارامتر ورودی می باشد. پارامتر اول شماره ستون و پارامتر دوم شماره سطر را مشخص می کند.

چاپ اعداد نوع short و long: برای چاپ اطلاعات عددی از نوع short می توانیم از کاراکتر فرمت %hd و برای چاپ اطلاعات عددی از نوع long می توانیم از کاراکتر %ld استفاده کنیم.

تعیین طول میدان در تابع printf()

با استفاده از امکاناتی که در تابع (printf) وجود دارد می توان مشخص کرد که هر کدام از اطلاعاتی که به خروجی می روند، چند کاراکتر از فضای خروجی را اشغال کنند. طول میدان خروجی، معمولا برای اعداد تعیین می شود و در ایجاد نظم در خروجی و شکل دهی جدول وار اطلاعات خروجی بسیار مفید است.

طول میدان مقادیر صحیح به صورت %wd بیان می شود که w طول میدان را مشخص می کند. اگر طول میدان با تعداد ارقام عدد صحیح برابر باشد عدد در طول میدان مشخص شده چاپ می شود. اگر طول میدان از تعداد ارقام عدد صحیح بیشتر باشد، عدد در سمت راست میدان قرار می گیرد و سمت چپ خالی می ماند. اما اگر طول میدان کمتر از تعداد ارقام عدد باشد، طول میدان نادیده گرفته شده تمام ارقام عدد در خروجی چاپ می شود.

طول میدان مقادیر اعشاری به صورت %wd بیان می شود که در آن، w کل طول میدان و d تعداد ارقام اعشاری است. اگر طول میدان قسمت صحیح بیشتر از بخش صحیح عدد باشد، سمت چپ قسمت صحیح خالی باقی مانده ولی اگر طول میدان قسمت صحیح، کمتر از بخش صحیح عدد باشد، کل بخش صحیح در خروجی چاپ و طول میدان نادیده گرفته می شود. اگر طول میدان قسمت اعشار، از تعداد ارقام اعشار بیشتر باشد، ارقام اعشاری در سمت چپ میدان قرار گرفته سمت راست میدان خالی می ماند. اما اگر طول میدان قسمت اعشاری، از تعداد ارقام اعشاری کمتر باشد، قسمت اعشاری عدد، گرد می شود.

هنگام گرد کردن، چنانچه رقم حذف شده، بزرگتر یا مساوی w باشد، یک واحد به رقم سمت چپ آن اضافه می شود. بعنوان مثال اگر A=123 باشد در هر یک از حالات %5d ، %3d و %2d خروجی بصورت زیر خواهد بود.

```
%5d  [ ][ ][1][2][3]
%3d  [1][2][3]
%2d  [1][2][3]
```

همچنین اگر B=-15.25431 باشد خروجی در هریک از حالات %7.2f ، %6.2f و %4.2f بصورت زیر خواهد بود

%7.2f		-	1	5	.	2	5
-------	--	---	---	---	---	---	---

%6.2f	-	1	5	.	2	5
-------	---	---	---	---	---	---

%4.2f	-	1	5	.	2	5
-------	---	---	---	---	---	---

۲-۱-۲- تابع (scanf)

در زبان C برای خواندن ورودی از صفحه کلید می توان از تابع (scanf) که در فایل stdio.h قرار دارد استفاده کرد. این تابع در صورت بروز خطا مقدار EOF را برگشت خواهد داد. ساختار تابع (scanf) همانند تابع (printf) از دو بخش تشکیل شده است، کешکل کلی آن در زیر نشان داده شده است.

scanf("<عبارت ۲>",<عبارت ۱>);

بخش اول شامل مقادیر ورودی و چگونگی خواندن آن ها می باشد و بخش دوم محل ذخیره سازی مقادیر گرفته شده را مشخص می کند. در بخش دوم باید قبل از نام متغیر علامت & را قرار دهیم به غیر از متغیرهای نوع اشاره گر.

داخل بخش اول سه نوع کاراکتر می تواند قرار بگیرد:

(۱) **کاراکترهای فرمت:** این کاراکترها مشخص کننده داده هایی است که باید از ورودی خوانده شوند.

(۲) **کاراکترهای فضای خالی:** وجود این کاراکترها باعث می شود تا (scanf) از فضای خالی موجود در ابتدای اطلاعات صرف نظر کند.

(۳) **کاراکترهای غیر فضای خالی و فرمت:** وجود این نوع کاراکترها باعث می شود تا چنان چه آن کاراکتر از ورودی خوانده شود از آن صرف نظر گردد.

```
scanf ("%d",&A);
scanf ("char?%c",&ch);
```

۲-۱-۳- توابع (getch) و (getche)

این توابع که در فایل conio.h قرار دارند برای خواندن یک کاراکتر از صفحه کلید استفاده می شوند. اگر اجرای برنامه به یکی از این توابع برسد اجرای آن متوقف شده و منتظر فشار دادن کلیدی از کاربر می ماند. به محض فشار دادن کلید کد آن برگشت داده می شود و اجرای برنامه ادامه می یابد. بعنوان مثال اگر از صفحه کلید کلید A فشار داده شده باشد توسط این تابع عدد ۶۵ برگشت داده می شود.

انواع کلیدها:

در روی صفحه کلید دو نوع کلید وجود دارد: **کلیدهای تک کده، کلیدهای دو کده**

کلیدهای تک کده: کلیدهایی هستند که با فشار دادن آن ها فقط یک کد ایجاد می شود و بسته به کلید فشار داده شده متفاوت می باشد.

کلیدهای دو کده: کلیدهایی هستند که با فشار دادن آن ها دو کد تولید می شود که کد اول همیشه برابر صفر و کد دوم بسته به کلید فشار داده شده متغیر می باشد.

از جمله کلیدهای تک کده می توان به کلیدهای کاراکتری، علائم، اعداد، ... اشاره کرد. و از جمله کلیدهای دو کده می توان به کلید F1 تا F12، کلیدهای جهت (←→↑↓)، کلیدهای ترکیبی و ... اشاره کرد.

برای خواندن کلیدهای دو کده نیاز به استفاده از دو تابع (getch) یا (getche) وجود دارد. اما برای خواندن کلیدهای تک کده یکی از آن ها کافی می باشد.

۲-۱-۴- تابع () getchar

مانند تابع () getche برای خواندن یک کاراکتر از صفحه کلید استفاده می شود. این تابع در فایل stdio.h قرار دارد. در این تابع پس از وارد کردن کاراکتر باید کلید Enter را فشار دهیم.

۲-۱-۵- نوشتن کاراکتر با توابع () putchar و () putch

این توابع می توانند یک کاراکتر یا یک متغیر کاراکتری را در صفحه نمایش چاپ کنند. تابع () putch در فایل conio.h و تابع () putchar در فایل stdio.h قرار دارد.

مثال ۱: برنامه ای بنویسید که عددی را بعنوان ثانیه از کاربر گرفته و مشخص کند که چند ساعت، دقیقه و ثانیه می باشد.

```
#include <stdio.h>
#include <conio.h>
int main( )
{
    unsigned int S,H,M;
    printf("\n Enter secound:");
    scanf("%u",&S);
    H=S/3600;
    S=S%3600;
    M=S/60;
    S=S%60;
    printf("\nH=%u, M=%u, S=%u",H,M,S);
    return 0;
}
```

برنامه ۱: تبدیل ثانیه به ساعت، دقیقه و ثانیه

۳- حلقه های تکرار و ساختارهای تصمیم

۳-۱- حلقه for

ساختار حلقه for در زبان C از سه بخش تشکیل می شود که مابین هر یک از بخش ها باید کاراکتر ; قرار گیرد.

(گام حلقه ; شرط حلقه ; مقدار اولیه اندیس حلقه) **for**

```
{
    دستور ۱
    دستور ۲
    ...
    دستور n
}
```

در بخش اول متغیر حلقه مقدار دهی اولیه می شود. این متغیر را شمارنده یا اندیس حلقه تکرار می نامیم. اندیس حلقه دارای یک مقدار اولیه است و در هر بار اجرای دستورات حلقه مقداری به آن اضافه یا کم می شود. بخش دوم شرط پایان حلقه را مشخص می کند و بخش سوم گام های حلقه را مشخص می کند.

بدنه حلقه for می تواند به یکی از دو حالت زیر باشد:

(۱) فقط از یک دستور تشکیل شده باشد

```
for(i=0; i<100;i++)
    printf("i=%d\n",i);
```

(۲) از بیش از یک دستور تشکیل شده باشد

در این حالت باید از نماد ابتدا ({) و نماد انتها (}) استفاده کنیم تا ابتدا و انتهای حلقه for مشخص شود.

```
for(i=0;i<100;i++)
{
    scanf("%d",&A);
    printf("A=%d\n",A);
}
```

در حلقه for این امکان وجود دارد که در بخش اول بیش از یک متغیر را مقدار دهی اولیه کنیم یا اینکه اصلاً مقدار دهی اولیه انجام ندهیم. اگر بیش از یک متغیر را مقدار دهی اولیه انجام دهیم باید بین متغیرها از کاراکتر ", استفاده کنیم. و اگر نخواهیم مقدار دهی اولیه انجام دهیم بخش اول را خالی می گذاریم. مثالهای زیر سه حالت مختلف را برای بخش اول نشان می دهند.

```
for (i=0; i<100;i++)
for (i=0 , j= 0 ;i<100;i++)
for ( ; i<100 ; i++)
```

در حلقه for می توان شرط پایان حلقه را حذف کرد که در این حالت حلقه به صورت بی نهایت به کار خود ادامه خواهد داد در این حالت برای پایان کار حلقه از دستورات شکننده حلقه استفاده می کنیم.

```
for (i=0 ; ; i++)
```

در حلقه های زبان C این امکان وجود دارد که در گام های حلقه بیش از یک متغیر به روزرسانی شود در این حالت باید مابین هر یک از بخش ها از کاراکتر کاما ", استفاده کنیم. همچنین میتوان این بخش را از حلقه for حذف کرد در این حالت باید عمل به روزرسانی در داخل بدنه حلقه for انجام دهیم. در زیر مثالهایی از این دو حالت آمده است.

```
for (i=0,j=0;i<100;i++,j+=2)
for (i=0;i<100; )
```

نکته: متغیر حلقه در داخل بدنه حلقه قابل تغییر می باشد پس تغییر آن داخل بدنه for می تواند تعداد تکرارهای این حلقه را

تغییر دهد.

مثال ۲: برنامه ای بنویسید که از کاربر ۲۰ عدد را گرفته میانگین اعداد را محاسبه کرده و در خروجی نمایش دهد.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int i,sum=0,num;
    float avg;
    clrscr();
    for(i=0;i<20;i++)
    {
        printf("\n Enter a number %d:",i+1);
        scanf("%d",&num);
        sum+=num;
    }
    avg=(float)sum/20;    // or  avg=sum/20.0;
    printf("the average is:%6.2f ",avg);
    getch();
    return 0;
}
```

برنامه ۲: میانگین ۲۰ عدد

۳-۲- انواع خطاها

۱- **خطاهای نحوی:** اینگونه خطاها زمانی ایجاد می شوند که برنامه نوشته شده از قوانین نحوی زبان C تبعیت نکند. به عنوان مثال متغیری تعریف شده نباشد. در این حالت کامپایلر پیام خطایی را نمایش خواهد داد.

۲- **خطاهای منطقی:** اینگونه خطاها مربوط به منطق برنامه نوشته شده می باشد و برای آن ها هیچ پیام خطایی نمایش داده نمی شود و فقط برنامه نتیجه درستی را نمی دهد.

۳- **خطاهای زمان اجرا:** اینگونه خطاها زمان اجرای برنامه رخ داده و برای کاربر پیام خطایی نمایش داده می شود. مانند خطای تقسیم بر صفر.

مثال ۳: برنامه ای بنویسید که یک جدول ضرب 10×10 را در خروجی نمایش دهد.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,j;
    clrscr();
    for(i=1;i<=10;i++)
    {
        for(j=1;j<=10;j++)
            printf("%4d",i*j);
        printf("\n");
    }
    getch();
    return 0;
}
```

برنامه ۳: جدول ضرب 10×10

۳-۳- حلقه while

ساختار تکرار while شرط را کنترل کرده و تا زمانی که این شرط برقرار باشد دستورات بدنه حلقه اجرا می شود و به محض نقض شدن شرط، اجرای این حلقه خاتمه پیدا می کند. در حلقه های while شرط حلقه در ابتدای حلقه کنترل می شود. دوحالت برای استفاده از حلقه while وجود دارد.

while (شرط)

دستور

while (شرط)

```
{
    دستور ۱
    دستور ۲
    ...
    دستور n
}
```

مثال ۴: برنامه ای بنویسید که تا زمانی که کاربر بخواهد اعدادی را از کاربر گرفته و میانگین این اعداد را محاسبه و چاپ نماید.

مثال ۵: برنامه ای بنویسید که مجموع ارقام یک عدد صحیح را محاسبه کرده و در خروجی چاپ کند.

```
#include<stdio.h>
#include<conio.h>
int main( )
{
    int i=0;
    long int s=0;
    char ch='y';
    while(ch=='y')
    {
        printf("\n Enter a number:");
        scanf ("%d",&i);
        s+=i;
        printf("Do you want to continue?");
        ch=getch();
        i++;
    }
    printf("Avg=%f", (float)s/i);
    return 0;
}
```

برنامه ۴: میانگین تعدادی عدد

```
#include <stdio.h>

int main( )
{
    int n,sum=0;
    printf("\n Enter a number:");
    scanf ("%d",&n);
    while(n)
    {
        sum+=(n%10);
        n/=10;
    }
    printf("\nsum=%d",sum);
    return 0;
}
```

برنامه ۵: مجموع ارقام یک عدد صحیح

۳-۴- حلقه do...while

ساختار تکرار do...while مانند ساختار تکرار while است، با این تفاوت که در ساختار while، شرط حلقه در ابتدای حلقه تست می شود، در حالی که در do...while شرط حلقه در انتهای حلقه تست می شود. بنابراین دستورات موجود در حلقه do...while در هر حال حداقل یک بار اجرا می شود.


```
i=0;
do
{
    printf("%d",i);
    i++;
}while(i<100);
```

```
i=0;
do
    printf("%d",i++);
while(i<100);
```

۳-۵- ساختار تصمیم if

برای تصمیم گیری می توانیم از دستور if استفاده کنیم. مثال زیر دو نمونه از دستور if را نمایش می دهد. در حالت اول اگر شرط برقرار باشد دستور یا دستورات خاصی اجرا خواهد شد در غیر این صورت هیچ دستوری اجرا نمی شود. اما در حالت دوم، اگر شرط برقرار باشد دستور یا دستورات بخش if اجرا می شود و اگر شرط برقرار نباشد دستور یا دستورات بخش else اجرا می شود.

حالت اول: `if (A<B) A=B-1;`

حالت دوم: `if (A<B) A=B;`
`else A=B+1;`

توجه: در بخش شرط دستورات if, while, do... while, for, لزومی ندارد که فقط از عملگرهای رابطه ای یا منطقی استفاده کنیم. می توان از هر عملگر دیگر نیز استفاده کرد اگر نتیجه محاسبه عدد صفر باشد به مفهوم `0=false`، هر عدد غیر صفر به مفهوم `0~0=True` می باشد برای مثال:

`if(!A)` اگر `A=0` باشد

`if(A)` اگر `A≠0` باشد

مثال ۶: برنامه ای بنویسید که عدد صحیح و مثبت `N` و رقم `D` را از کاربر گرفته مجموع ارقام و تعداد تکرارهای رقم `D` را در داخل آن محاسبه کرد و در خروجی چاپ کند.

```
#include<stdio.h>
int main( )
{
    unsigned int N,D,i=0,T,S=0;
    printf("Enter N ?");
    scanf("%d",&N);
    do
    {
        printf("\n Enter a digit (0..9)?");
        scanf("%d",&D);
    }while(D>=9);
    do
    {
        T=N%10;
        N/=10;
        if(T==D)i++;
        S+=T;
    }while(N!=0);
    printf("\nS=%d , i=%d",S,i);
    return 0;
}
```

برنامه ۶: برنامه مثال ۶

۳-۶- دستور break

اجرای این دستور در داخل حلقه های تکرار باعث خواهد شد که داخلی ترین حلقه شکسته شود یکی دیگر از موارد استفاده دستور break در ساختار switch می باشد.

۳-۷- دستور continue

اجرای این دستور باعث می شود در حلقه تکرار اجرای برنامه به ابتدای حلقه منتقل شود اگر در داخل حلقه ها بعد از دستور break یا continue دستورات دیگری قرار گیرد اجرا نخواهد شد.

۳-۸- ساختار تصمیم switch

ساختار کلی این دستور بصورت زیر می باشد:

```
switch (عبارت)
{
    case <مقدار ۱>:
        <دستورات ۱>
        break;
    case <مقدار ۲>:
        <دستورات ۲>
        break;
    .
    .
    .
    default:
        <دستورات n>
}
```

شاید برخی مواقع نیاز باشد که بخواهیم مقدار یک متغیر را برای حالت های مختلف بررسی کنیم در این حالت اگر از دستور if استفاده کنیم نیاز به استفاده از چندین if else پشت سرهم می باشد در چنین حالتی می توانیم از ساختار تصمیم switch استفاده کنیم. در این ساختار اگر مقدار متغیر مورد نیاز با یکی از case ها برابر باشد دستورات مربوط آن case تا رسیدن به دستور break اجرا خواهد شد اگر در انتهای دستورات یک case دستور break را قرار ندهیم باعث می شود با مقدار case بعدی or شود مثال زیر مقدار متغیر c را برای حالت های R,G,Y تست می کند و بسته به حالت مورد نظر پیام مناسب را چاپ خواهد کرد. این مثال با استفاده از دستور if, switch به صورت زیر خواهد بود:

```
switch(c)
{
    case 'R': printf("Red");
              break;
    case 'G': printf("Green");
              break;
    case 'Y': printf("Yellow");
              break;
    default:
              printf ("error");
}
```

مثال بالا با استفاده از switch

```
if (c= 'R')
    printf('Red');
else if (c= 'G')
    printf("Green");
else if (c= 'Y')
    printf("Yellow");
else printf("error");
```

مثال بالا با استفاده از if...else

دستور switch راحتتر و انعطاف پذیرتر از if...else می باشد. بعنوان مثال اگر در مثال بالا اگر بخواهیم مابین حروف کوچک یا بزرگ تفاوت وجود نداشته باشد می توان دستور switch بالا را بصورت زیر تغییر داد :

```
switch(c)
{
    case 'r':
    case 'R': printf("Red");
              break;
    case 'g':
    case 'G': printf("Green");
              break;
    case 'y':
    case 'Y': printf("Yellow");
              break;
    default:
              printf ("error");
}
```

اگر متغیر بخش switch با هیچ یک از case ها برابر نباشد دستورات بخش default اجرا خواهد شد. نوشتن بخش default اختیاری می باشد. مقادیر موجود در case های switch نمی توانند با همدیگر برابر باشند. در بخش case نمی توان از عبارت منطقی با رابطه ای استفاده کرد فقط حالت مساوی بودن بررسی می شود و همچنین دستورات switch را می توان به صورت تو در تو استفاده کرد.

مثال ۷: برنامه ای بنویسید که دو عدد و یک عملگر را از کاربر گرفته و بسته به عملگر وارد شده نتیجه محاسبه بر روی اعداد را در خروجی نمایش دهد.

```
#include<stdio.h>
#include<conio.h>

int main ( )
{
    int n1,n2;
    char op;
    printf ("\n Enter two numbers:");
    scanf ("%d%d",&n1,&n2);
    op=getch( );
    switch(op){
        case '+': printf ("sum=%d",n1+n2);
                  break;
        case '-': printf ("\n minus=%d",n1-n2);
                  break;
        case '/': printf ("\n division=%f",(float)n1/n2);
                  break;
        case '*': printf ("\n multiply=%d",n1*n2);
                  break;
        default: printf ("error in operator");
    }
    return 0;
}
```

برنامه ۷: جواب مثال ۷

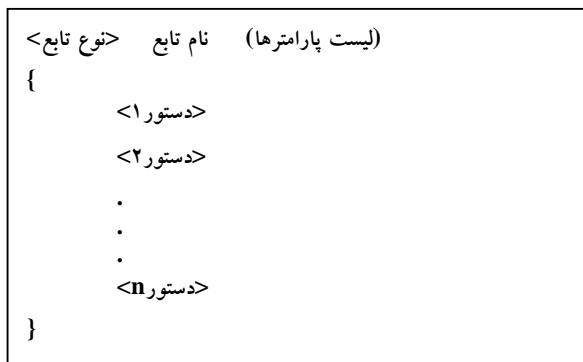
تمرینات

۱. برنامه ای برای محاسبه ریشه های معادله درجه دو $ax^2 + bx + c = 0$ تحت شرایطی که a مخالف 0 باشد بنویسید.
راهنمایی: برای محاسبه جذر می توان از تابع `sqrt()` که در فایل `math.h` قرار دارد استفاده کرد.
۲. برنامه ای برای محاسبه $a \times b$ با جمع های متوالی بنویسید. برای حالت علامتدار.
۳. برنامه ای برای محاسبه $a \div b$ با تفریق های متوالی بنویسید. برای حالت علامتدار.
۴. برنامه ای برای محاسبه سری $S = 1! + 2! + 3! + \dots + N!$ بنویسید.

۴- توابع و کلاسهای حافظه

۴-۱- توابع

در پایین ساختار توابع در زبان C آورده شده است.



این ساختار از بخش های زیر تشکیل شده است :

نوع تابع : این بخش مشخص کننده نوع مقداری است که تابع برگشت می دهد و می تواند یکی از انواع داده ای استاندارد زبان C باشد نوشتن این بخش اختیاری می باشد.

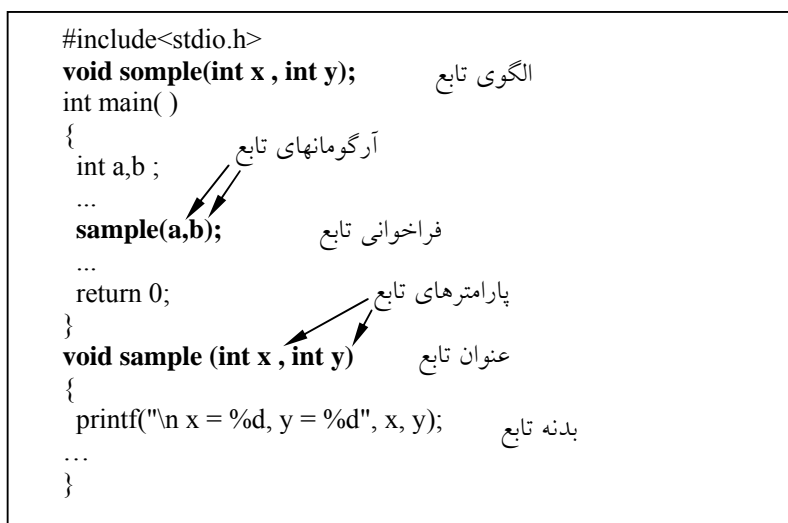
نام تابع : این بخش مشخص کننده نام تابع میباشد و باید از قوانین نام گذاری شناسه های زبان C تبعیت کند.

لیست پارامترها : بعد از تعریف نام تابع داخل () لیست پارامتر های ورودی تابع قرار می گیرد در این لیست می توانیم ۰ تا N پارامتر را قرار دهیم و باید مابین پارامترها از عملگر " , " استفاده کنیم.

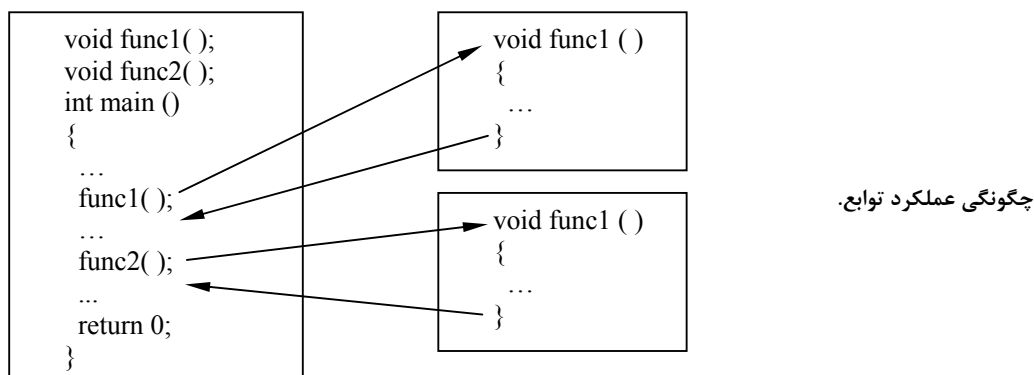
بدنه توابع : این بخش مشخص کننده بدنه توابع بوده و شامل دستورات اجرایی زبان C می باشد.

باید داخل برنامه های زبان C قبل از تابع () main الگوی تابع را تعریف کنیم. در فصل ۴ کتاب ۱۰ مورد مهم برای تعریف و استفاده از توابع آورده شده است که باید به آنها دقت داشته باشید.

شکل زیر نمونه مثالی از تعریف توابع در زبان C می باشد.



شیوه به کارگیری توابع در برنامه



شکل بالا چگونگی عملکرد توابع را نشان می‌دهد، هنگام فراخوانی تابع اجرای برنامه به بدنه تابع منتقل شده و از اولین دستور تا آخرین دستور اجرا می‌شود پس از اتمام کار تابع اجرای برنامه به تابع فراخوانی کننده و به دستور بعد از فراخوانی تابع منتقل می‌شود.

مثال ۸: تابعی بنویسید که سه عدد را به عنوان آرگومان دریافت کرده سپس از بین این اعداد بزرگترین عدد را پیدا کرده و در خروجی چاپ کند.

```
#include<stdio.h>

void max(int a,int b,int c );

int main()
{
    int a1,a2,a3;
    printf("\n enter three number ?");
    scanf ("%d%d%d",&a1,&a2,&a3);
    max(a1,a2,a3);
    return 0;
}
/*****
void max (int a,int b,int c)
{
    int m=a;
    if(b>m) m=b;
    if (c>m) m=c;
    printf("max=%d",m);
}
```

برنامه ۸: برنامه مربوط به مثال ۸

نوع تابع مشخص کننده مقداری است که توسط تابع برگشت داده می‌شود، برای برگرداندن مقدار از توابع می‌توان از دستور `return` استفاده کرد مقابل دستور `return` مقداری را که قرار است برگشت داده شود قرار خواهیم داد. پس از اجرای دستور `return` مقدار خواسته شده برگشت داده شد و کار تابع خاتمه پیدا می‌کند پس دستوراتی که داخل تابع بعد از `return` قرار بگیرد اجرا نخواهد شد. اگر تابع از نوع `void` باشد هیچ لزومی به استفاده از دستور `return` وجود ندارد می‌توان از `return;` استفاده کرد اما اگر نوع تابع غیر `void` باشد حتما باید برای برگرداندن مقدار از `return` استفاده کنیم.

مثال ۹: برنامه ای بنویسید که مجموع ارقام یک عدد اعشاری را محاسبه کند؟

```
#include <stdio.h>
#include <conio.h>
int Dsum(float f);
int main()
{
    float p;
    printf("\n Enter a float number?");
    scanf("%f",&p);
    printf("\n Digit sum of %8.4f = %d",p,Dsum(p));
    return 0;
}
//*****
int Dsum(float f)
{
    float t;
    int s=0,i,p;
    if(f<0)
        f*=-1;
    i=(int)f;
    t=f-i;
    do
    {
        p=i%10;
        i=i/10;
        s+=p;
    }while(i);
    i=4;
    do
    {
        t*=10;
        p=(int)t;
        t-=p;
        s+=p;
        i--;
    }while( t && i );
    return s;
}
```

برنامه ۹: برنامه مثال ۹ برای محاسبه مجموع ارقام عدد اعشاری تا ۴ رقم اعشار.

۴-۱-۱- روش های ارسال پارامتر ها به توابع

به دو طریق می توان پارامتر ها را ارسال کرد

۱. فراخوانی با مقدار (call by value)
۲. فراخوانی با ارجاع (call by reference)

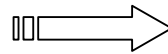
در روش فراخوانی با مقدار هنگام فراخوانی مقادیر آرگومان ها در پارامترها متناظر کپی می شود پس هر گونه تغییر روی پارامتر ها در روی آرگومان ها هیچ تاثیری ندارد، اما در روش فراخوانی با ارجاع آدرس آرگومان ها به پارامترها منتقل می شود پس در این حالت هر گونه تغییر روی پارامترها باعث خواهد شد آرگومان متناظر تغییر پیدا کند.

نکته : در حالت call by value امکان برگشت دادن مقدار با استفاده از آرگومان ها وجود ندارد اما در روش فراخوانی با ارجاع می توان مقادیر را با استفاده از آرگومان ها برگشت داد.

در زیر نمونه مثالی از دو روش ارسال پارامتر آورده شده است تابع `f1()` بروش فراخوانی با مقدار فراخوانی می شود بهمین دلیل همچنان که خروجی را مشاهده می کنید با تغییر پارامتر `a` داخل تابع `i` هیچ تغییری نمی کند (همچنان 0 باقی می ماند) اما تابع `f2()` بروش فراخوانی با ارجاع فراخوانی می شود بهمین دلیل همچنان که خروجی را مشاهده می کنید با تغییر پارامتر `*a` داخل تابع `i` تغییر می کند (برابر 5 می شود).

```
void f1(int a);
void f2(int *a);
int main()
{
    int i=0;
    f1(i);
    printf("\n in call by value i = %d",i);
    f2 (&i);
    printf("\n in call by reference i = %d",i);
    return 0;
}
void f1(int a)
{
    a=5;
}
void f2(int *a)
{
    *a=5
}
```

خروجی حاصل از اجرای برنامه



in call by value i = 0
in call by reference i =5

```
#include <stdio.h>
#include <conio.h>
int Dsum(float f);
int main()
{
    float p;
    printf("\n Enter a float number?");
    scanf("%f",&p);
    printf("\n Digit sum of %8.4f = %d",p,Dsum(p));
    return 0;
}
//*****
int Dsum(float f)
{
    int s=0,p;
    long int i;
    if(f<0)
        f*=-1;
    while(f != (long int)f) f*=10;
    i=(long int)f;
    do
    {
        p=i%10;
        i=i/10;
        s+=p;
    }while(i);
    return s;
}
```

برنامه ۱۰: برنامه مثال ۹ برای محاسبه مجموع ارقام عدد اعشاری.

۴-۱-۲- متغیرهای محلی و عمومی

متغیر ها را میتوان داخل برنامه های C در دو محل تعریف کرد

۱. داخل توابع

۲. خارج از توابع

متغیر هایی که داخل توابع تعریف شوند به عنوان **متغیرهای محلی (local variables)** شناخته می شوند و متغیر هایی که خارج از توابع تعریف شوند به عنوان **متغیرهای عمومی (global variables)** شناخته می شوند. متغیرهای محلی فقط برای همان تابع شناخته شده و خارج از تابع داخل توابع دیگر قابل دسترسی و تغییر نمی باشند اما متغیر های سراسری در هر بخش از برنامه قابل دسترسی و تغییر می باشد در مثال زیر متغیرهای t و p متغیر محلی بوده و فقط داخل main شناخته شده می باشد (داخل تابع f1 تعریف شده نبود و قابل دسترسی نمی باشد) و همچنین متغیر های a و b متغیر سراسری بوده و داخل کلیه توابع (در این مثال f1 و main) قابل دسترسی و تغییر می باشد.

```
#include <stdio.h>
int a,b;
void f1();
int main()
{
    int p,t;
    ...
    return 0;
}
void f1()
{
    float x;
    ...
}
```

اگر داخل توابع متغیر محلی همانام با متغیر سراسری تعریف شود داخل آن تابع متغیر سراسری همانام با متغیر محلی دیگر قابل دسترسی و استفاده نخواهد بود بلکه از متغیر محلی استفاده می شود.

۴-۱-۳- روش بازگشتی

اگر تابع مستقیماً خودش را فراخوانی کند و یا با واسطه خودش را فراخوانی کند مثلاً تابع f1 و f2 را فراخوانی کرده و f2 نیز f1 را فراخوانی کند به این روش نوشتن توابع روش بازگشتی گفته می شود.

```
int f1 (
)
{
    ...
    f1 ( );
    ...
}

int f1 (
)
{
    ...
    f2 ( )
    ...
}
int f2 ( )
{
    ...
    f1 ( )
    ...
}
```

مثال ۱۰: تابع فاکتوریل را به دو روش بازگشتی و غیر بازگشتی بنویسید.

راهنمایی: برای حالت غیر بازگشتی می‌توانید از رابطه $n! = 1 \times 2 \times \dots \times (n-1) \times n$ استفاده کنید و برای حالت بازگشتی می‌توانید از رابطه $n! = (n-1)! \times n$ استفاده کنید.

```
long int fact (int n)
{
    long int t=1;
    int i;
    for (i=1;i<=n;i++)
        t*=i;
    return t;
}
```

روش غیر بازگشتی

```
long int fact (int n)
{
    if (n==1) return 1;
    else
        return fact(n-1)*n;
}
```

روش بازگشتی

برنامه ۱۱: جواب مثال ۱۰

توجه: در توابع بازگشتی حتما باید شرطی را برای پایان فراخوانی های پشت سر هم قرار دهیم به عنوان مثال در تابع فاکتوریل به روش بازگشتی شرط $n==1$ این عمل را انجام میدهد در غیر این صورت برنامه داخل حلقه بی نهایت گیر خواهد کرد.

مثال ۱۱: تابعی بنویسید که حاصل ضرب دو عدد را با استفاده از جمع های متوالی و به صورت بازگشتی و غیر بازگشتی محاسبه کند؟
راهنمایی: برای این محاسبه $A \times B$ می‌توان A ها را به تعداد B بار با هم جمع کرد $(A \times B = A+A+A+\dots+A)$ یا B ها را به تعداد A بار با هم جمع کرد $(A \times B = B+B+B+\dots+B)$.

راهنمایی: برای حالت بازگشتی می‌توانید از رابطه $A \times B = (A \times (B-1)) + A$ استفاده کرد.

```
int mul(unsigned int A, unsigned int B)
{
    unsigned int R=0 ,i;
    for(i=1 ; i<=B ; i++)
        R += A;
    return R;
}
```

روش غیر بازگشتی

```
int mul(unsigned int A, unsigned int B)
{
    if(B==0)
        return 0;
    else
        return mul(A,B-1)+A;
}
```

روش بازگشتی

برنامه ۱۲: جواب مثال ۱۱

مثال ۱۲: تابعی بنویسید که حاصل تقسیم دو عدد صحیح مثبت A, B را با استفاده از تفریق های متوالی این کار را به دو روش بازگشتی و غیر بازگشتی انجام دهید؟

راهنمایی: برای این محاسبه $A \div B$ می توان از A هر بار B را کم کرد تا زمانی که $A \geq B$ است تعداد این کم کردنها حاصل تقسیم را مشخص می کند.

راهنمایی: برای حالت بازگشتی می توانید از رابطه $A \div B = ((A - B) \div B) + 1$ استفاده کرد.

```
int div(unsigned int A, unsigned int B)
{
    unsigned int R=0;
    while(A>=B)
    {
        A-=B;
        R++;
    }
    return R;
}
```

روش غیر بازگشتی

```
int div(unsigned int A, unsigned int B)
{
    if(A<B)
        return 0;
    else
        return div(A-B,B)+1;
}
```

روش بازگشتی

برنامه ۱۲: جواب مثال ۱۲

مثال ۱۳: برنامه ای بنویسید که جمله n ام از سری فیبوناچی را به روش بازگشتی تولید کند ؟

```
int fib(int n)
{
    if (n==1 || n==2) return 1;
    else return fib(n-1)+fib(n-2);
}
```

برنامه ۱۴: جواب مثال ۱۳

۴-۲- کلاس های حافظه

برای کلاس های حافظه در زبان C دو مورد باید مشخص شود :

۱. scope یا حوزه متغیر

۲. life time یا طول عمر

منظور از حوزه متغیر محل هایی از برنامه است که متغیر قابل دسترسی می باشد و منظور از طول عمر مدت زمانی است که متغیر در حافظه باقی می ماند (زمانی که متغیر در حافظه ایجاد شده و زمانی که از حافظه حذف می شود).

در زبان C چهار کلاس حافظه زیر وجود دارد

۱. کلاس حافظه اتوماتیک (automatic)

۲. کلاس حافظه ثبات (register)

۳. کلاس حافظه استاتیک (static)

۴. کلاس حافظه خارجی (extern)

برای تعیین کلاس حافظه برای متغیرها، بصورت زیر عمل می‌شود:

نام متغیر <نوع متغیر> <کلاس حافظه>

مانند:

```
static int x;
register char y;
```

۴-۲-۱- کلاس حافظه اتوماتیک

متغیرهایی که داخل توابع تعریف می‌شوند یا همان متغیرهای محلی به عنوان متغیرها با کلاس حافظه اتوماتیک می‌باشند. حوزه برای این نوع کلاس حافظه داخل تابع می‌باشد و طول عمر آن مدت زمان اجرای تابع (با فراخوانی تابع ایجاد شده و با تمام کار تابع از حافظه حذف می‌شود)

در زبان C کلاس حافظه اتوماتیک را با auto مشخص می‌شود اگر کلاس حافظه برای متغیری مشخص نشود خود کامپایلر بصورت پیش فرض کلاس حافظه آنرا را از نوع اتوماتیک در نظر خواهد گرفت.

مثال: auto int a;

۴-۲-۲- کلاس حافظه ثبات

کلاس حافظه ثبات به کامپایلر پیشنهاد می‌کند که متغیر اتوماتیک را در ثبات پردازنده قرار دهد بنابر این حوزه و طول عمر این متغیرها همانند کلاس حافظه اتوماتیک می‌باشد. این عمل فقط جنبه پیشنهاد به کامپایلر را دارد اگر امکان اختصاص ثبات وجود داشته باشد این عمل انجام خواهد شد و در غیر این صورت کامپایلر با این متغیر همانند کلاس حافظه اتوماتیک عمل خواهد کرد و آنرا روی RAM ایجاد می‌کند تعداد ثبات حافظه محدود بوده پس باید تعداد محدود متغیر از این نوع کلاس حافظه تعریف شود. معمولاً متغیرهایی که بیشتر مورد دسترسی قرار می‌گیرند از نوع کلاس حافظه ثبات تعریف می‌شود مانند متغیرهای حلقه. استفاده از کلاس حافظه ثبات محدودیت‌هایی دارد که عبارتند از:

- فقط برای متغیرهای محلی قابل استفاده است.
- انواع کاراکتری صحیح و اشاره‌گر را می‌توان با کلاس حافظه ثبات تعریف کرد.
- به دلیل محدودیت تعداد ثبات‌های پردازنده فقط می‌توان تعداد محدودی متغیر از این نوع تعریف کرد.
- آدرس متغیرهای با کلاس حافظه ثبات معنی ندارد پس نمی‌توان برای آنها اشاره‌گری را تعریف کرد.

۴-۲-۳- کلاس حافظه استاتیک

برای کلاس حافظه استاتیک دو حالت در نظر گرفته می‌شود:

۱. استاتیک محلی.

۲. استاتیک سراسری.

متغیر/استاتیک محلی: فقط داخل همان تابع قابل دسترسی می‌باشد و هنگام فراخوانی تابع در حافظه ایجاد شده و پس از اتمام کار تابع آخرین مقدار خود را حفظ خواهد کرد. روی این گونه متغیرها فقط یک بار عمل مقدار دهی اولیه انجام می‌شود. مثال زیر نمونه‌ای از کاربرد این گونه کلاس حافظه می‌باشد:

```

void f1( );
int main()
{
    int I;
    for(I=0;I<5;I++)
        f1();
    return 0;
}
void f1( )
{
    int a=0;
    static int b=0;
    printf("\n a= %d , b=%d", a++ ,b++);
}

```

```

a=0 , b=0
a=0 , b=1
a=0 , b=2
a=0 , b=3
a=0 , b=4

```

خروجی حاصل از اجرای برنامه مقابل

همچنانکه خروجی برنامه را مشاهده می کنید مقادیر تولید شده برای b متفاوت از مقادیر a می باشد، به این دلیل که کلاس حافظه b از نوع استاتیک محلی می باشد.

برای متغیرهای استاتیک سراسری حوزه متغیر داخل توابعی است که بعد از تعریف متغیر قرار گرفته اند و طول عمر آن مدت اجرای برنامه می باشد در مثال زیر متغیرهای a,b از نوع متغیر استاتیک سراسری می باشد و برای توابع f1, f2 قابل دسترسی بوده اما برای تابع main قابل دسترسی نمی باشد.

```

void f1();
void f2();
vnt main()
{
    ...
}
static int a,b ;
void f1()
{
    ...
}
void f2()
{
    ...
}

```

۴-۲-۴- کلاس حافظه خارجی

این گونه متغیرها با شروع اجرای برنامه ایجاد شده و تا پایان اجرای برنامه در حافظه باقی می ماند و همچنین از سراسر برنامه قابل دسترسی می باشند.

دو مثال زیر نمونه از کاربرد این گونه متغیرها می باشد در مثال شماره ۱ برنامه به دو فایل f1.c و f2.c تقسیم شده است که متغیر a, b static فقط داخل f1 شناخته شده و قابل استفاده می باشد اما متغیر m,n داخل هر دو فایل قابل استفاده می باشد، ذکر عبارت extern int m,n داخل تابع f2.c برای این منظور می باشد که توابع داخل این فایل تشخیص دهند متغیر سراسری با نام m,n از نوع int داخل فایل دیگر بصورت سراسری تعریف شده است.

اما در مثال دوم متغیر k به صورت سراسری تعریف شده نوشتن عبارت extern int k داخل تابع f1 به این منظور می باشد که تابع f1 تشخیص دهد خارج از این تابع متغیر k از نوع int به صورت سراسری تعریف شده است نوشتن این عبارت ضروری نمی باشد چون خود کامپایلر به صورت اتوماتیک این حالت را تشخیص می دهد .

فایل f1.c

```
static int a,b
int m ,n ;
void f1 ( );
void f2 ( );
void f3 ( );
vnt main ()
{
...
}
void f1( )
{
...
}
```

فایل f2.c

```
extern int m,n;
void f2 ( )
{
...
}
void f3( )
{
...
}
```

```
int k;
void f1( );
int main ()
{
...
}
static int a , b ;
void f1( );
{
extern int k;
...
}
```

کاربرد extern جهت اعلان متغیرهای عمومی

کاربرد متغیرهای عمومی و استاتیک عمومی

۵- آرایه ها و رشته ها

تا حال که متغیر ها تعریف می کردیم فقط یک سلول حافظه داشتند و اگر به عنوان مثال بخواهیم ۱۰ سلول حافظه داشته باشیم باید ۱۰ متغیر با نام های متفاوت تعریف کنیم استفاده از آرایه ها باعث می شود بتوانیم چندین سلول حافظه با یک نام را تعریف کنیم آرایه با ابعاد مختلف وجود دارد

- آرایه های تک بعدی
- آرایه های دو بعدی
- و
- و آرایه های n بعدی

۵-۱- آرایه های تک بعدی

مثال زیر نمونه ای از تعریف یک آرایه تک بعدی به طول ۵ سلول حافظه از نوع int می باشد برای دسترسی به هر یک از سلول های حافظه می توانیم از نام آرایه و اندیس آن استفاده کنیم در زبان C شماره اندیس های آرایه از عدد صفر شروع می شود.

```
int A[5];
```

```
A[2]=5;
```

آرایه A			5		
	A[0]	A[1]	A[2]	A[3]	A[4]

مثال ۱۴: برنامه ای بنویسید که نمرات ۲۰ دانشجوی کلاسی را در درس برنامه نویسی گرفته و از بین نمرات بزرگترین نمره و کوچکترین نمره و همچنین میانگین نمرات کلاس را محاسبه کرده و چاپ کند؟

```
#include<stdio.h>

#define N 20

int main()
{
    float a[N],max =0,min=20,sum=0;
    int i;
    for( i=0; i<N; i++)
    {
        printf ("\n enter number %d :", i+1);
        scanf ("%f",&a[i]);
    }
    for (i=0; i<N; i++)
    {
        sum += a[i];
        if(a[i]<min) min=a[i];
        else if(a[i]>max) max=a[i];
    }
    printf ("\n max = %5.2f , min = %5.2f , avg = %5.2f",max, min, sum/N);
    return 0;
}
```

برنامه ۱۵: جواب مثال ۱۴

تذکر: اگر بخواهیم بزرگترین و کوچکترین عدد را در بین اعداد ورودی بدست آوریم دو حالت وجود دارد :

۱. این عدد بازه مشخصی دارد

۲. این عدد بازه نامشخص دارد

اگر عدد بازه مشخصی داشته باشد به عنوان مثال نمره (مابین صفر تا بیست) می توان به صورت پیش فرض طبق مثال قبل کوچکترین عدد بازه را روی max و بزرگترین عدد بازه را روی min قرار داد سپس مقایسات را روی اعداد ورودی انجام می دهیم اما اگر بازه اعداد مشخصی نباشد مانند عدد صحیح (از منفی بینهایت تا مثبت بینهایت) در این حالت معمولاً اولین عدد گرفته شده هم به عنوان max و هم به عنوان min در نظر گرفته می شود سپس مقایسات انجام می شود طبق مثال زیر :

مثال ۱۵ : برنامه ای بنویسید که ۱۰ عدد را از کاربر گرفته و در آرایه ای قرار داده سپس بزرگترین، کوچکترین و میانگین اعداد را محاسبه کرده و در خروجی چاپ کند ؟

```
#include<stdio.h>
#define N 10

int main()
{
    int a[N],max , min , sum=0;
    int i;
    for (i=0; i<N; i++)
    {
        printf ("\n enter number %d :",i+1);
        scanf ("%d",&a[i]);
    }
    min=max=a[0];
    for(i=0; i<N; i++)
    {
        sum += a[i];
        if(a[i]<min) min = a[i];
        else if(a[i]>max)max = a[i];
    }
    printf ("\nmax =%d , min = %d , avg = %f",max, min, (float)sum/N);
    return 0;
}
```

برنامه ۱۶: جواب مثال ۱۵

۵-۱-۱- ارسال آرایه های تک بعدی به عنوان پارامتر به توابع

برای ارسال آرایه ها به عنوان پارامتر به توابع سه راه حل وجود دارد:

۱. ارسال آرایه با طول مشخص به توابع؛ در این حالت داخل تابع می توانیم طول آرایه را تشخیص دهیم.

void f1(int arr[15])

۲. ارسال آرایه با طول نامشخص به توابع؛ در این حالت باید پارامتر دیگری داشته باشیم تا از طریق آن داخل تابع طول آرایه را تشخیص دهیم.

void f2(int arr[],int len)

۳. استفاده از اشاره گر ها (این راه حل بعداً در مبحث اشاره گر ها توضیح داده خواهد شد).

۵-۲- مرتب سازی آرایه ها

۵-۲-۱- روش مرتب سازی حبابی

در روش مرتب سازی حبابی هر عدد با عنصر بعدی خود مقایسه می شود و بسته به شرایط (صعودی و نزولی) جای آنها با همدیگر عوض خواهد شد. در این روش چون ترتیب حرکت اعداد همانند حباب می باشد روش مرتب سازی حبابی گفته می شود.

(در این مثال مرتب سازی به صورت صعودی می باشد)

```
void bubble(int a[] , int len )
{
    int i, j, temp;
    for(i=len-1; 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 ;
            } //end of if
}
```

برنامه ۱۷: مرتب سازی حبابی

۵-۲-۲- مرتب سازی انتخابی

در این روش مرتب سازی هر بار از ابتدای آرایه تا انتهای آن کل اعداد مقایسه می شوند و هر بار کوچکترین (یا بزرگترین) عدد از بین این اعداد انتخاب می شود، سپس این عدد در محل مناسب خود قرار گرفته و دوباره همین عمل بر روی اعداد باقی مانده انجام می شود تا اینکه کل این اعداد داخل مجموعه بصورت مرتب قرار گیرند. این روش مرتب سازی به روش مرتب سازی انتخابی موسوم است.

تابع به صورت نزولی :

```
void selection(int a[] , int len)
{
    int i, j, temp, min;
    for (i=0 ; i<len ; i++)
    {
        min = i;
        for (j=i ; j< len ; j++)
            if (a[min] > a[j] )min = j;
        temp = a[i];
        a[i] = a[min];
        a[min] = temp
    }
}
```

برنامه ۱۸: مرتب سازی انتخابی

۵-۳- جستجو در آرایه

برای جستجو در آرایه اول باید مشخص کنیم که آرایه مورد نظر مرتب است یا نه برای این دو حالت برنامه های متفاوتی را می توان نوشت در صورتی که آرایه نامرتب باشد می توان از روش جستجوی ترتیبی استفاده کرد و در صورتی که آرایه مرتب باشد یکی از الگوریتمهایی که وجود دارد جستجوی دودویی می باشد.

۵-۳-۱- جستجوی ترتیبی

در این الگوریتم با شروع از اولین عنصر تا آخرین عنصر بترتیب تمام عناصر با عنصر مورد نظر مقایسه می شود تا اینکه عنصر مورد نظر پیدا شده یا کل عناصر جستجو شده و عنصر پیدا نشود.

```
int search(int A[] , int len , int no )
{
    int i;
    for(i=0;i<len;i++)
        if(A[i]==no) return i;
    return -1 ;
}
```

برنامه ۱۹: مرتب سازی ترتیبی

۵-۳-۲- جستجوی دودویی

در این روش جستجو که بر روی مجموعه اعداد مرتب اعمال می شود در بازه اعداد جستجو عنصر وسطی پیدا شده و عدد مورد نظر با آن مقایسه می شود ، سه حالت امکان پذیر است. در صورتی که عدد مورد نظر از عنصر وسطی کوچکتر باشد باز عمل جستجو روی بازه اعداد سمت چپ عنصر وسطی انجام می شود.

در صورتی که عدد مورد نظر از عنصر وسطی بزرگتر باشد ، عمل جستجو روی بازه اعداد سمت راست عنصر وسطی انجام می شود. در صورتی که عدد مورد نظر با عنصر وسطی برابر باشد در این صورت عدد مورد نظر پیدا شده است، و مراحل جستجو متوقف می شود .

در صورتی که این نصف کردن ها تا جایی ادامه پیدا کرد که دیگر بازه اعداد قابل نصف کردن نباشد (مجموعه تک عنصری داشته باشیم که با عدد مورد نظر برابر نباشد) در این صورت جستجو خاتمه پیدا کرده و عدد مورد نظر پیدا نشده است.

```
int bsearch(int a[] , int len , int no )
{
    int l=0 , r=len-1,m;
    while(l<=r)
    {
        m=(l+r)/2;
        if(no < a[m])
            r = m-1;
        else if (no > a[m])
            l=m+1;
        else return m;
    }
    return -1 ;
}
```

برنامه ۲۰: مرتب سازی دودویی

۵-۴- آرایه دو بعدی

آرایه های دو بعدی قبلاً در مبحث فلوچارتها بصورت کامل بحث شده اند. برای تعریف آرایه های دو بعدی در زبان C می توان بصورت زیر عمل کرد.

نوع آرایه نام آرایه [بعد۱] بعد۲];

مثال: `int A[4][3];`

مثال بالا نمونه ای از تعریف آرایه دو بعدی میباشد در این تعریف عدد اول (۴) تعداد سطرها و عدد دوم (۳) تعداد ستونها را مشخص می کند، پس مجموعاً برای این آرایه ۱۲ خانه حافظه ($4 \times 3 = 12$) از نوع `int` اختصاص داده می شود.

بدلیل خصوصیات حافظه، سلولهای حافظه پشت سر هم بصورت خطی قرار گرفته اند پس امکان پیاده سازی آرایه دو بعدی روی حافظه وجود ندارد، پس برای حل این مشکل باید این آرایه دو بعدی به یک آرایه تک بعدی نگاشت (تبدیل) کنیم برای این منظور از راه حلی که قبلاً در مبحث فلوچارتها به آن اشاره شد استفاده می شود. برای مشخص کردن تعداد خانه های آرایه تک بعدی معدل می توان از رابطه زیر استفاده کرد :

تعداد ستونها \times تعداد سطرها = تعداد خانه های حافظه مورد نیاز

همچنین برای پیدا کردن موقعیت خانه `A[x][y]` این آرایه دو بعدی روی آرایه تک بعدی معدل می توان از رابطه زیر استفاده کرد.

$y + (x \times \text{تعداد ستونها}) = \text{اندیس آرایه تک بعدی}$

از این رابطه می توان نتیجه گرفت که هنگام تعریف آرایه دو بعدی حتماً باید تعداد ستونها را مشخص کنیم.

۵-۴-۱- ارسال آرایه های دو بعدی به عنوان آرگومان به توابع

برای ارسال آرایه های دو بعدی به عنوان آرگومان به توابع سه روش متفاوت وجود دارد

- آرایه دو بعدی با تعداد سطرها و ستون های مشخص

مثال: `void f1(int A[5][4]);`

- آرایه دو بعدی با سطر نامشخص اما با ستون مشخص (حتماً باید تعداد ستون را مشخص کنیم بنابه دلایلی که قبلاً توضیح داده ایم) در این حالت حتماً باید پارامتر دیگری برای مشخص کردن تعداد سطرها داشته باشیم.

مثال: `void f1(int A[][4],int row);`

- استفاده از اشاره گرها (این حالت بعداً در بخش اشاره گرها توضیح داده خواهد شد).

۵-۵- مقداردهی اولیه آرایه ها

برای مقداردهی اولیه آرایه های تک بعدی و دو بعدی راه حلهای متفاوتی وجود دارد که نمونه مثالهایی از آنها را در زیر مشاهده می کنید.

```
int a1[4]={ 1,2,4,0 }
int a2[]={ 3,2,4,5 };
int a[2][3]={ {1,0,3},{3,10,2} };
int a[2][3]={ 1,0,3,5,10,2 };
int a[][3]={ 1,0,3,5,10,2 };
```

۵-۶- آرایه های n بعدی

در زبان C می توان آرایه های سه بعدی و بیشتر را تعریف کرد شکل تعریف آرایه ها با بعدها بیشتر بصورت زیر می باشد.

نوع آرایه نام آرایه [بعد۱] بعد۲]...[بعدn];

برای پیاده سازی آرایه n بعدی این آرایه به یک آرایه n-1 بعدی، آرایه n-1 بعدی به یک آرایه n-2 بعدی تبدیل می شود و به این ترتیب انجام می شود تا نهایتاً به یک آرایه تک بعدی برسیم.

۵-۷- رشته‌ها

در زبان C برای ذخیره سازی رشته ها از آرایه های کاراکتری استفاده می‌شود. در مثال زیر برای ذخیره سازی رشته "Ali" آرایه کاراکتری s با ۷ سلول حافظه تعریف شده است بترتیب تک تک عناصر این رشته در خانه های مورد نظر روی آرایه قرار گرفته است و نهایتاً برای مشخص کردن انتهای رشته کاراکتر '\0' (NULL) بعد از آخرین کاراکتر افزوده می‌شود.

```
char s[7];
```

رشته "Ali" روی آرایه کاراکتری s بصورت زیر قرار خواهد گرفت.

A	l	i	\0	?	?	?
s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]

سوال: در زبان C چه تفاوتی ما بین 'A' با "A" وجود دارد؟
عبارت اول به مفهوم کاراکتر 'A' می‌باشد و برای ذخیره سازی آن فقط یک سلول حافظه مورد نیاز است اما عبارت دوم به مفهوم رشته می‌باشد و برای ذخیره سازی آن حداقل دو سلول حافظه مورد نیاز می‌باشد (یکی برای کاراکتر A و دیگری برای کاراکتر '\0')

۵-۷-۱- مقدار دهی اولیه رشته ها

```
char str1[15] = "Computer";
char str2[] = "Computer";
char str3[] = {'C','o','m','p','u','t','e','r','\0'};
```

۵-۷-۲- ورودی و خروجی رشته ها

برای گرفتن رشته از ورودی می‌توانیم از تابع scanf و یا gets استفاده کنیم هنگام استفاده از تابع scanf باید از کاراکتر format و "%s" استفاده کنیم در این حالت به یاد داشته باشید که نباید قبل از نام رشته از علامت & استفاده کنیم تابع scanf ورودی را تا رسیدن به کاراکتر فاصله و یا tab یا enter داخل رشته قرار میدهد پس در رشته خوانده شده نمی‌تواند فاصله و یا tab وجود داشته باشد به عنوان مثال اگر کاربر عبارت ali reza را تایپ کند داخل رشته فقط عبارت "ali" قرار خواهد گرفت برای رفع این مشکل می‌توان از تابع gets استفاده کرد. تابع gets در فایل stdio.h قرار دارد. این تابع کاراکترهای وارد شده را تا رسیدن به Enter داخل رشته قرار می‌دهد.

```
scanf("%s", str );
gets(str );
```

برای چاپ رشته می‌توان از تابع printf و یا puts استفاده کرد هر دو این توابع داخل فایل سرایند stdio.h قرار دارند

```
printf("\n string = %s" , str );
puts(str);
```

۵-۷-۳- ارسال رشته ها به عنوان پارامتر به توابع

این حالت همانند آرایه ها می‌باشد با این تفاوت که هنگام ارسال آرایه با طول نامشخص دیگر نیازی به آرگومان دیگر برای مشخص کردن طول وجود ندارد

```
void f1(char str[15])
void f2(char str[ ])
```

مثال ۱۶: تابعی بنویسید که رشته ای را دریافت کرده و داخل آن کلیه حروف کوچک را به حروف بزرگ تبدیل کند؟
توجه: اختلاف مابین حروف بزرگ و حروف کوچک در جدول کدهای اسکی ۳۲ واحد می‌باشد.

```
void upper(char s[])
{
    int i;
    for(i=0 ; s[i] ; i++)
        if ((s[i]>='a') && (s[i]<='z')) s[i]-=32;
}
```

اگر بعنوان مثال ورودی این تابع رشته "Computer" باشد خروجی آن رشته "COMPUTER" خواهد بود.
 مثال ۱۷: برنامه ای بنویسد که رشته را از کاربر دریافت کرده سپس آن رشته را به ترتیب معکوس در خروجی چاپ کند ؟

```
#include<stdio.h>
void strinvert(char s[]);
int main()
{
    char str[50];
    printf("\n enter a string :");
    gets(str);
    strinvert(str);
    puts (str) ;
    return 0;
}
//*****
void strinvert(char s[])
{
    int i,length;
    char t;
    for(length=0 ; s[length] ; length++);
    length--;
    for(i=0 ; i<(length/2) ; i++)
    {
        t = s[i];
        s[i] = s[length-i];
        s[length-i] = t;
    }
}
```

برنامه ۲۲: معکوس کردن رشته

مثال ۱۸: برنامه ای بنویسید که رشته ای را از کاربر دریافت کرده سپس کلیه کاراکترهای فاصله داخل آن رشته را حذف کند ؟

```
#include<stdio.h>
void delspace(char s[]);
int main()
{
    char str[50];
    printf("\n enter a string :");
    gets(str);
    delspace(str);
    puts(str);
    return 0;
}
//*****
void delspace(char s[])
{
    int i=0,j;
    while(s[i])
        if(s[i] == ' ')
            for (j=i ; s[j] ; j++)
                s[j] = s[j+1];
            else i++;
}
```

برنامه ۲۳: حذف کاراکتر فاصله از رشته

۵-۷-۴- انتساب رشته ها

در زبان C امکان انتساب رشته ها به یکدیگر با استفاده از عملگر = وجود ندارد برای این منظور strcpy استفاده می شود که در فایل string.h قرار دارد. شکل کلی این تابع بصورت زیر می باشد.

strcpy(s1,s2);

دقت داشته باشید که باید طول s1 بزرگتر مساوی طول s2 باشد در غیر این صورت سلولهای حافظه بعد از s1 تخریب خواهند شد.

۵-۷-۵- مقایسه رشته ها

مقایسه رشته ها در زبان C نمیتوان به سادگی با استفاده از عملگرهای مقایسه ای انجام شود، برای این منظور تابع strcmp در فایل string.h قرار داده شده است.

strcmp(s1,s2);

این تابع رشته s1 را با s2 مقایسه کرده و عددی را برگشت می دهد، اگر این دو رشته برابر باشند عدد صفر برگشت داده می شود اگر s1 کوچکتر از s2 باشد یک عدد منفی و اگر s1 بزرگتر از s2 باشد یک عدد مثبت برگشت داده خواهد شد.

۵-۷-۶- الحاق رشته ها

برای الحاق رشته ها در زبان C تابع strcat قرار داده شده است، این تابع در فایل string.h قرار دارد.

strcat(s1,s2);

این تابع رشته s2 را به انتهای رشته s1 اضافه می کند باید دقت داشته باشیم که رشته s1 باید به تعداد مورد نیاز حافظه خالی در اختیار داشته باشد در غیر این صورت موجب تخریب برخی از سلولهای حافظه بعد از s1 خواهد شد.

۵-۷-۷- آرایه ای از رشته ها

برای تعریف آرایه ای از رشته ها می توانیم از آرایه دو بعدی استفاده کنیم در مثال زیر یک آرایه دو بعدی با نام name تعریف شده است این آرایه می تواند ۵ رشته که هر رشته می تواند حداکثر ۱۲ کاراکتر (۱۲=۱۳) داشته باشد ذخیره سازی کند.

char name[5][13];

۶- اشاره گرها

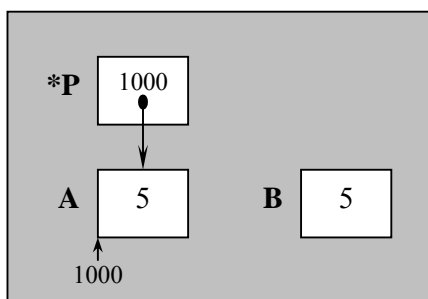
۶-۱- متغیرهای اشاره گر

برای تعریف اشاره گر می توانیم از عملگر * استفاده کنیم. هر متغیر اشاره گر حاوی آدرس محلی از حافظه می باشد که می توان با استفاده از این آدرس به آن محل اشاره کرد و به محتویات آن دسترسی پیدا کرد.

```
int *p;
```

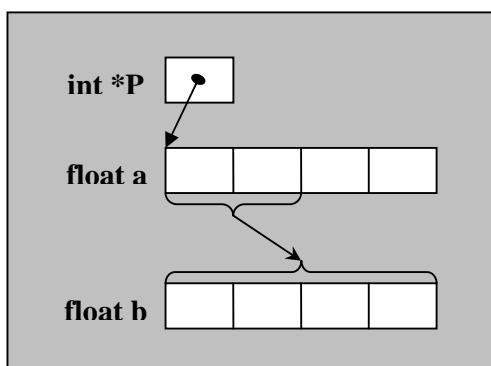
اشاره گری با نام p از نوع int تعریف می کند. مثال زیر نمونه ای از کاربرد اشاره گر را نشان می دهد

```
int *P,A,B;
p=&A;
*p=5;
B=*P;
```



تذکر: تعریف نوع اشاره گر مهم است به این دلیل که نشان دهنده نوع سلول حافظه ای است که به آن اشاره دارد. در مثال زیر طبق برنامه ای که نوشته شده انتظار می رود عدد 5.95 داخل متغیر b قرار گیرد. در حالی که این اتفاق نخواهد افتاد و عددی غیر از 5.95 داخل b قرار می گیرد. به این دلیل که نوع اشاره گر p (int) با نوع سلول حافظه ی اشاره شده (float) برابر نمی باشد.

```
float a,b;
int *P;
a=5.95;
P=&a;
b=*P;
```



همانطور که در شکل هم دیده می شود دلیل این اتفاق این است که فقط دو بایت از متغیر a داخل b (به اندازه ۴ بایت) قرار می گیرد که باعث می شود که عمل انتصاب بدرستی انجام نشود.

۶-۲- اعمال روی اشاره گرها

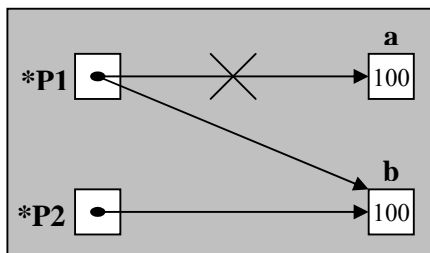
روی اشاره گرها می توان اعمال مختلفی را مانند متغیرهای معمولی انجام داد که عبارتند از :

۱. عمل انتساب اشاره گرها به یکدیگر
۲. اعمال محاسباتی جمع و تفریق
۳. عمل مقایسه رشته ها

۶-۲-۱- عمل انتساب اشاره گرها به یکدیگر

نمونه مثالی از این عمل در زیر آورده شده است.

```
1. int *P1,*P2,a=50,b=100;
2. P1=&a;
3. P2=&b;
4. *P1=*P2;
5. P1=P2;
```

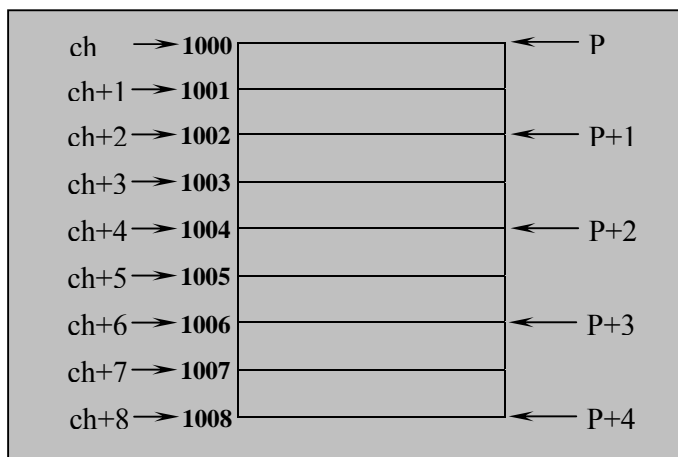


تذکر: در مثال بالا پس از اجرای سطر ۴ محتوای خانه ای که P2 به آن اشاره دارد (متغیر b) داخل سلول حافظه‌ای قرار می‌گیرد که P1 به آن اشاره دارد (متغیر a). پس از اجرای سطر ۵ مقدار اشاره گر P2 داخل اشاره گر P1 قرار خواهد گرفت. پس، بعد از اجرای این دستور، P2 به هر جا که اشاره دارد P1 نیز به همان جا اشاره خواهد کرد.

۶-۲-۲- اعمال محاسباتی جمع و تفریق

روی اشاره گرها می‌توان اعمال جمع و تفریق را انجام داد. اگر P یک اشاره گر باشد P+x به x امین سلول بعد از P (بسته به نوع P) اشاره خواهد کرد و P-x به x امین سلول قبل از P (بسته به نوع P) اشاره خواهد کرد. اگر P از نوع char باشد P+1 به یک بایت بعدی، P+2 به دو بایت بعدی و ... اشاره خواهد کرد. اما اگر P از نوع int باشد P+1 به دو بایت بعدی، P+2 به چهار بایت بعدی و ... اشاره خواهد کرد.

```
int *P;
char *ch;
```



۶-۳- تخصیص حافظه ی پویا

در زبان C برای تخصیص حافظه به صورت پویا می‌توان از تابع malloc() که فایل Stdlib.h قرار دارد استفاده کرد. شکل کلی آن بصورت زیر می‌باشد.

```
malloc( size ) (نوع) = اشاره گر;
```


این تابع به اندازه خواسته شده سلول حافظه را جدا کرده و آدرس آن را برگشت می‌دهد. این آدرس برگشتی از نوع void می‌باشد، یعنی اینکه هیچ نوعی برای آن تعریف نشده است، پس باید این آدرس را به نوع مورد نظر تبدیل کنیم در مثال زیر 10 سلول حافظه از نوع int توسط تابع malloc() اختصاص یافته سپس این آدرس با استفاده از (int *) به آدرسی از نوع int تبدیل می‌شود و نهایتاً این آدرس داخل اشاره گر P قرار می‌گیرد.

```
int *P;
P = (int *)malloc( sizeof(int) );
```

اگر بنا بر هر دلیلی عمل تخصیص به درستی انجام نشود مقدار NULL توسط تابع malloc() برگشت داده می‌شود. برای برگرداندن حافظه ی اختصاص داده شده به سیستم می‌توان از تابع free() استفاده کرد.

```
free (p);
```

۴-۶- اشاره گر ها و توابع

مثال ۱۹: تابعی بنویسید که دو متغیر به عنوان پارامتر دریافت کرده و محتوای آنها را با هم عوض کند.

```
#include<stdio.h>
void swap(int *a,int *b);
int main()
{
    int i,j;
    printf ("\n Enter two numbers?");
    scanf ("%d%d",&i,&j);
    swap(&i,&j);
    printf ("\n i=%d , j=%d",i,j);
    return 0;
}
void swap(int *a,int *b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}
```

برنامه ۲۴: جواب مثال ۱۹

در مثال بالا برای تابع swap() دو پرا متر با نام a,b از نوع اشاره گر تعریف شده است در هنگام فراخوانی این تابع اگر بخواهیم محتوای متغیرهای i,j را تغییر دهیم باید آدرس این متغیرها را به تابع swap() ارسال کنیم (swap (&i,&j)) با این کار امکان تغییر محتوای متغیر در i,j داخل تابع swap() ایجاد می‌شود. به این روش فرا خوانی، فراخوانی با ارجاع (call by reference) گفته می‌شود. نمونه ای از این حالت تابع scanf می‌باشد.

۵-۶- اشاره گر ها و آرایه ها

در زبان C نام آرایه اشاره گری به اولین عنصر آرایه می‌باشد. پس با آرایه ها می‌توان همانند اشاره گر ها و با اشاره گر ها می‌توان همانند آرایه ها برخورد کرد، به عنوان مثال اگر اشاره گری با نام p را داشته باشیم:

P[0] معادل *p ، P[1] معادل *(p+1) ، P[2] معادل *(p+2) ، ... و P[i] معادل *(p+i) می‌باشد حالت عکس نیز برقرار می‌باشد.

```
int Arr[5],*P;
P=Arr;
*(P+3)=4;    \ \ OR  p[3]=4;
*(P+2)=3;    \ \ OR  p[2]=3;
```

در زبان C می‌توان عباراتی بر حسب char *S="Computer"; را نوشت، در این حالت *S معادل تعریف آرایه با بعد نا مشخص می‌باشد. (char *S معادل S[])

مثال ۲۰: برنامه ای بنویسید که آدرس های تولید شده برای اشاره گرهایی از نوع char و float و int و double را در اعمال محاسباتی جمع و تفریق نمایش دهد.

```
int main()
{
    int i,*P1,a;
    char *P2,b;
    float *P3,c;
    double *P4,d;
    for(i=0;i<5;i++)
        printf("\n P1=%P, P2=%P, P3=%P, P4=%P in P+%d", P1+i,P2+i,P3+i,P4+i,i);
    for(i=0;i<5;i++)
        printf("\n P1=%P, P2=%P, P3=%P, P4=%P in P-%d", P1-i,P2-i,P3-i,P4-i,i);
    return 0;
}
```

برنامه ۲۵: جواب مثال ۲۰

مثال ۲۱: برنامه ای بنویسید که با استفاده از تخصیص حافظه پویا دو عدد را از کاربر گرفته و مجموعات مربعات آنها را محاسبه کرده و در خروجی نمایش دهد.

```
int main( )
{
    int *a,*b,c;
    a = (int*)malloc(sizeof (int));
    if(!a)
    {
        printf("\nError in allocate memory.");
        exit(1);
    } // end if
    b = (int *)malloc(sizeof(int));
    if(!b)
    {
        printf("Error in allocation memory.");
        exit(1);
    } //end if
    printf("\nEnter two number:");
    scanf("%d%d",a,b);
    c = *a**a+*b**b;
    printf("\nresult = %d ", c);
    free(a);
    free(b);
    return 0;
} //end main
```

برنامه ۲۶: جواب مثال ۲۱

۶-۶- آرایه پویا

برای ایجاد آرایه به صورت پویا می توان از تابع malloc() استفاده کرد بطور مثال اگر بخواهیم یک آرایه پویا به تعداد ۱۰ خانه را ایجاد کنیم می توانیم از تابع malloc() بصورت زیر استفاده کنیم:

```
int *P;
P=(int *)malloc(sizeof(int)*10);
```

برای دسترسی به خانه های این آرایه می توانیم از حالت اشاره گریا آرایه ای استفاده کنیم ، به عنوان مثال اگر بخواهیم به خانه ی ششم (اندیس 5) دسترسی داشته باشیم حالت اشاره گر برابر $*(P+5)$ و حالت آرایه ای برابر $P[5]$ خواهد بود ، نهایتاً با استفاده از تابع $free()$ می توان حافظه ی اختصاص داده شده را به سیستم برگرداند.

مثال ۲۲: برنامه ای بنویسید که شبیه تابع $strcpy$ عمل کرده رشته ی دوم را داخل پارامتر اول $copy$ کند با این تفاوت که تخصیص حافظه باید به صورت پویا انجام شود (به تعداد مورد نیاز حافظه اختصاص داده شود).

```
#include<stdio.h>
#include<stdlib.h>
void strcpy(char *, char *);
int main()
{
    char str1[50] , *str2,l;
    printf("\n Enter a string?");
    gets(str1);
    for( l=0 ; str1[l] ; l++)
        l++;
    str2 = (char *)malloc(sizeof(char)*l+1);
    strcpy(str2 , str1);
    printf("\n Str2 = %s",str2);
    free(str2);
    return 0;
}
/*****
void strcpy(char *s1,char *s2)
{
    int i;
    for(i=0 ; s2[i] ; i++)
        s1[i] = s2[i];
    s1[i] = '\0';
}
```

برنامه ۲۲: جواب مثال ۲۲

مثال ۲۳: برنامه ای بنویسید که دو رشته را دریافت کرده به صرت پویا حافظه ای برای الحاق دو رشته ایجاد کرده و نهایتاً دو رشته را به هم ملحق کرده و آدرس این آرایه را برگشت دهد.

```
char *stringcat( char *s1 , char *s2)
{
    char *t;
    int l1,l2,l;
    for (l1=0 ; s1[l1] ; l1++);
    for (l2=0 ; s2[l2] ; l2++);
    l = l1 + l2 + 1;
    t = (char *) malloc ( sizeof(char) * l );
    if(!t)
    {
        printf("\n Error in memory allocation.");
        exit (1);
    }
    strcpy (t , s1);
    strcat (t, s2);
    return t;
}
```

برنامه ۲۸: جواب مثال ۲۳

۶-۷- ارزش دهی اولیه به اشاره گرها

برای ارزش دهی اولیه به اشاره گرها می توانیم به صورت زیر عمل کنیم در این مثال ها S آرایه ای با بعد نامشخص برابر همان اشاره گر می باشد.

```
int A[] = {1,4,3,2,0};      OR      int *A = {1,4,3,2,0};
char S[] = "computer";      OR      char *S = "computer";
void f1(int A[],int len)     OR      void f1(int *A,int len)
```

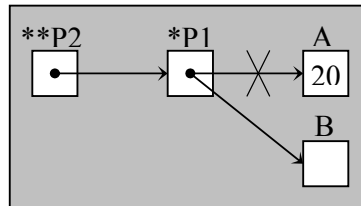
در مثال سوم (ارسال آرایه ها با بعد نامشخص به توابع) باز می توان بجای آرایه ی با بعد نامشخص از اشاره گر استفاده کرد.

۶-۸- اشاره گر به اشاره گر

زمانی که یک متغیر معمولی بعنوان مثال A را تعریف می کنیم کامپایلر فضای حافظه ی مورد نیاز را اختصاص داده و نام آن را برابر حرف A قرار می دهد. و زمانی هم که بخواهیم اشاره گری به این خانه تعریف کنیم از * استفاده می کردیم. بعنوان مثال int *P1. این اشاره گر P1 به خانه ی حافظه ای از نوع int اشاره خواهد کرد. اما اگر بخواهیم اشاره گری به خود این اشاره گر تعریف کنیم نمی توانیم به روش قبل عمل کنیم، این بار باید در مقابل نام اشاره گر بجای یک * از دو * استفاده می کنیم، به عنوان مثال int **P2

مثال زیر نمونه ای از تعریف این نوع اشاره گرها را نشان می دهد. در این مثال اگر بخواهیم از طریق P2 به متغیر A برسیم باید از دو علامت * استفاده کنیم و اگر بخواهیم از طریق P1 برسیم باید از یک علامت * استفاده کنیم.

```
int A,*P1,**P2,B;
P1=&A;
P2=&P1;
**P2=20;
P2=&B;
```



مثال ۲۴: مساله ی شبیه سازی تابع strcpy را به صورتی تغییر دهید که تخصص حافظه داخل خود تابع انجام شود.

```
#include<stdio.h>
#include<stdlib.h>
void strcpy(char **,char *);
int main()
{
    char str1[50],*str2;
    printf("\n Enter a string :");
    gets(str1);
    strcpy(&str2,str1);
    printf("\n str2 = %s",str2);
    free(str2);
    return 0;
}
//*****
void strcpy( char **s1, char *s2)
{
    int l;
    for(l=0 ; s2[l] ; l++);
    l++;
    *s1=(char *)malloc(sizeof(char)*l);
    for( l=0 ; s2[l] ; l++)
        *(*s1+l) = *(s2+l);
    *(*s1+l) = '\0';
}
```

برنامه ۲۹: جواب مثال ۲۴

هنگام استفاده از اشاره گرها باید دقت داشته باشیم که به محل هایی دسترسی پیدا کنیم که اجازه ی دسترسی را داریم ، مثال زیر از لحاظ نحوی درست می باشد اما از لحاظ منطقی درست نیست چون هنوز مشخص نیست اشاره گر P به کدام محل از حافظه اشاره می کند پس هنگام قرار دادن عدد ۵ داخل *P ممکن است محلی از حافظه که اجازه دسترسی به آن را نداریم تخریب شود.

```
int *P;
*P=5;
```

۹-۶- آرگومان های تابع main()

تا حالا توابع main() هایی را که تعریف می کردیم بدون آرگومان ورودی بودند ، برای تابع main() می توان آرگومان هایی را قرار داد.

```
int main(int Argc, char *Argv[])
```

Argc مشخص کننده ی تعداد پارامترهای وارد شده در خط فرمان می باشد و Argv مقادیر این پارامترهای خط فرمان را نشان می دهد. به عنوان مثال فرض کنید فایلی با نام test.exe در مسیر E:\TC قرار دارد.

فرض کنید هنگام اجرای این برنامه در خط فرمان عبارت

```
D:\TC>test test1 test2
```

تایپ شده است،

در این حالت Argc مقدار 3 را خواهد داشت و Argv اندیس 0 برابر test ، اندیس 1 برابر test1 و اندیس 2 برابر test2 خواهد بود.

توجه : حد اقل مقدار برای Argc برابر 1 خواهد بود و همیشه در اندیس 0 ام Argv نام برنامه قرار می گیرد.

مثال ۲۵: برنامه ای بنویسید که مقابل خط فرمان تعدادی نام را دریافت کرده و برای هر یک از آنها پیام Hello را چاپ کند.

```
int main(int argc , char *argv[] )
{
    int i;
    if(argc==1)
        printf("\n Don't Enter name ");
    else
        for(i=1;i<argc;i++)
            printf("\n hello %s " , argv[i] );
    return 0 ;
}
```

برنامه ۳۰: جواب مثال ۲۵

۷- ساختمان ها

۷-۱- متغیرهای ساختمان

تا حالا انواع داده ای که تعریف شده نوع های داده ای استاندارد زبان C بودند و کاربر مجبور به استفاده از فقط این نوع های داده ای بود اما ساختمانها (struct) به کاربر این امکان را می دهند که بتواند انواع داده ای جدیدی بصورت دلخواه تعریف کند ، به عنوان مثال تاریخ، کاربر می تواند نوع داده ای جدیدی با استفاده از ساختمان ها برای ذخیره سازی تاریخ تعریف کند. شکل کلی تعریف ساختمانها در زبان C بصورت زیر می باشد.

```
Struct <ساختمان> {
```

```
    بدنه ساختمان
```

```
};
```

هنگام تعریف ساختمان بعد از عبارت struct نام ساختمان نوشته می شود ، داخل بدنه ی ساختمان می توانیم اجزاء ساختمان را معرفی کنیم این اجزاء می تواند از هر نوع داده ای باشد و نهایتاً بعد از بدنه ی ساختمان حتماً باید عبارت (;) قرار داده شود. مثال زیر نمونه ای از تعریف ساختمان برای ذخیره سازی اطلاعات پرسنلی را نشان می دهد :

```
struct personel
{
    char name[21];
    char fname[31];
    int P_ID;
};
```

این حالت فقط جنبه تعریف دارد و هیچ تغییری از آن ایجاد نمی شود (یعنی هیچ حافظه ای اختصاص داده نمی شود). پس از تعریف ساختار کلی ساختمان می توانیم متغیرهایی از نوع ساختمان تعریف شده را ایجاد کنیم. به عنوان مثال

```
struct personel P1,P2;
```

دو متغیر با نام های P1,P2 از نوع struct personel تعریف می شود، یعنی برای هر یک از آنها بصورت مجزا حافظه تخصیص داده می شود. توجه داشته باشید که برای تعریف متغیرهایی از نوع یک ساختمان خاص می توانیم پس از اتمام تعریف بدنه ی ساختمان و قبل از قرار دادن (;) این کار را انجام دهیم :

```
struct personel
{
    char name[21];
    char fname[31];
    int P_ID;
}P1,P2;
```

برای دسترسی به اجزای این متغیرها می توانیم از عملگر "." استفاده کنیم ، ابتدا نام متغیر نوشته شده سپس عملگر "." و نهایتاً نام عنصر مورد نظر نوشته می شود ، به عنوان مثال اگر بخواهیم به اجزای P1 دسترسی داشته باشیم می توانیم بصورت زیر عمل کنیم :

```
struct personel P1;
```

```
P1.P_ID
P1.name
P1.fname
```

همچنین بسته به عنصر مورد نظر می توانیم عملیات مربوط به آنرا انجام دهیم ، به عنوان مثال اگر بخواهیم به اندیس i ام نام متغیر P1 دسترسی داشته باشیم ، می توانیم بنویسیم :
`P1.name[i]`
 در مورد متغیرهای نوع ساختمان می توانیم همانند متغیرهای معمولی با استفاده از عملگر انتساب آنها را روی همدیگر کپی کنیم ، به عنوان مثال اگر بنویسیم `P1=P2` باعث خواهد شد کلیه ی عناصر P2 داخل P1 کپی شوند.

مثال ۲۶: برنامه ای بنویسید که مشخصات چند دانشجو را همراه با معدل آنها از کاربر گرفته سپس دومین معدل کلاس را پیدا کرده و در خروجی نشان دهد.

```
#include<stdio.h>
int main()
{
    struct student
    {
        char name[21];
        float Avg;
        int ID;
    } M1={0} , M2={0} , T;
    int n,i;
    printf( " Enter n? " );
    scanf ("%d", &n);
    for( i=0 ; i<n ; i++)
    {
        printf("\n Enter name? " ) ;
        scanf("%s",T.name);
        printf("\n Enter ID ? " );
        scanf ("%d", &T.ID);
        printf("\n Enter Avg? " );
        scanf("%f", &T.Avg );
        if(T.Avg > M1.Avg )
        {
            M2 = M1;
            M1 = T ;
        }
        else if(T.Avg > M2.Avg)
            M2 = T;
    }
    printf( "\n M2 name = %s ", M2.name );
    return 0;
}
```

برنامه ۳۱: جواب مثال ۲۶

۲-۷- مقداردهی اولیه ساختمان

برای مقدار دهی اولیه ی متغیرهای نوع ساختمان می توان همانند متغیرهای معمولی عمل کرد ، مقابل متغیر عناصر آنرا بترتیب قرار خواهیم داد در مثال زیر برای نام P1 رشته ی "Ali" برای fname رشته ی "Kazemi" و برای P1.ID عدد 10 قرار داده می شود.باید حتماً دقت داشته باشید که این عمل مقدار دهی برای عناصر ساختمان باید به ترتیب انجام شود.

```
struct personel P1 = {"Ali","Kazemi",10};
```

اگر به جای مقداردهی اولیه عدد 0 بصورت زیر قرار داده شود باعث خواهد شد تا کلیه ی عناصر این ساختمان با عدد 0 مقدار دهی شوند.

```
struct personel P2 = {0};
```

۳-۷- آرایه ای از ساختمان

همانند متغیرهای معمولی نیز می توانیم برای ساختمانها آرایه تعریف کنیم به عنوان مثال در مورد ساختمان پرسنل اگر بخواهیم اطلاعات 10 پرسنل را (آرایه ۱۰ عنصری) ذخیره سازی کنیم ، می توانیم آرایه ساختمان پرسنل را بصورت زیر تعریف کنیم:

```
struct personel P[10];
```

اگر بخواهیم به عنوان مثال به نام اندیس i ام از این آرایه دسترسی داشته باشیم می توانیم به صورت زیر عمل کنیم:

```
P1[i].name
```

همچنین اگر بخواهیم به اندیس j ام نام عنصر i ام دسترسی داشته باشیم می توانیم بنویسیم :

```
P1[i].name[j]
```

۷-۴- ساختمانهای لانه ای

داخل ساختمانها می توانیم از ساختمانهای دیگری نیز استفاده کنیم، یعنی به اصطلاح عنصر یک struct می تواند struct دیگری (غیر از همان struct) باشد ، به عنوان مثال فرض کنیم struct Date به صورت زیر تعریف شده:

```
struct Date
{
    int y;
    int m;
    int d;
};
```

اگر بخواهیم بعنوان مثال برای ذخیره سازی اطلاعات دانشجو از struct Date استفاده کنیم می توانیم به صورت زیر عمل کنیم:

```
struct student
{
    char name [21];
    int ID;
    struct Date bdate;
} S1;
```

برای دسترسی به عناصر این ساختمان های تو در تو می توانید از عملگر " . " استفاده کنیم . به عنوان مثال اگر بخواهیم به سال ، ماه ، یا روز تولد دانشجوی S1 دسترسی داشته باشیم می توانیم بصورت زیر عمل کنیم :

```
S1.bdate.y
S1.bdate.m
S1.bdate.d
```

۷-۵- اشاره گر ساختمان

همانند متغیرهای معمولی می توانیم اشاره گرهایی از نوع ساختمان تعریف کنیم، به عنوان مثال:

```
struct student *P
```

اشاره گری از نوع struct student با نام P تعریف می کند برای دسترسی به عناصر این اشاره گر دو راه حل وجود دارد، یا از عملگر * استفاده کنیم یا بجای عملگر " . " از عملگر " -> " استفاده کنیم . مثال زیر نمونه ای از کاربرد اشاره گر را نشان می دهد:

```
struct student *P, S1;
P = &S1;
P->ID = 5;      OR      (*P).ID = 5;
```

۷-۶- اختصاص حافظه به صورت پویا برای اشاره گر ساختمان

برای این منظور می توانیم از تابع malloc استفاده کنیم همچنین برای مشخص کردن اندازه ی این ساختمان می توانیم از عملگر sizeof استفاده کنیم ، همچنین برای آزاد سازی حافظه می توان از تابع free استفاده کرد. مثال زیر نمونه ای از این حالت را نشان می دهد:

```
struct student *P;
P = (struct student *)malloc( sizeof(struct student) );
.
.
.
free(p);
```

مثال ۲۷: تابعی بنویسید که اشاره گر ساختمان پرسنلی را دریافت کرده ، داده هایی را از ورودی دریافت کرده و داخل آن قرار دهد.

توجه : داده های ساختمان عبارتند از نام (۳۰ کاراکتر)، ID، حقوق (salary) و تاریخ تولد.

```
struct date
{
    int y;
    int m;
    int d;
};
//////////
struct personel
{
    char name[31];
    int ID;
    long salary;
    struct date bdate;
};
//*****
void getinfo(struct personel *P)
{
    printf("\n Enter name?");
    scanf("%s",p->name);
    printf("\n Enter ID?") ;
    scanf("%d",&P->ID );
    printf("\n Enter salary?") ;
    scanf("%ld", &P->salary?);
    printf("\n Enter year,month,day of birth day?");
    scanf("%d%d%d",&P->bdate.y,&P->bdate.m,&P->bdate.d);
}
```

برنامه ۳۲: جواب مثال ۲۷

۷-۷- ساختمانهای بیتی

تا حالا میزان فضای حافظه ای که برای متغیرها اختصاص داده می شد حداقل اندازه ی آن در حد یک بایت بود، برای اینکه بتوانیم این اختصاص ها را به صورت بیتی انجام دهیم (یا در حقیقت بتوانیم به بیت‌های یک بایت دسترسی پیدا کنیم)، می توانیم از ساختمانهای بیتی استفاده کنیم. حالت کلی تعریف یک ساختمان بیتی بصورت زیر می‌باشد.

```
struct نام ساختمان بیتی {
    < طول فیلد ۱ > : < نام فیلد ۱ > < نوع فیلد ۱ > ;
    < طول فیلد ۲ > : < نام فیلد ۲ > < نوع فیلد ۲ > ;
    .
    .
    .
    < طول فیلد n > : < نام فیلد n > < نوع فیلد n > ;
}
```

اسامی متغیرهای بیتی {

در ساختمانهای بیتی برای هر عنصر ابتدا باید نوع آنرا مشخص کنیم این نوع می تواند int، unsigned و یا signed باشد، سپس نام عنصر و نهایتاً پس از علامت ' : ' تعداد بیتها را باید مشخص کنیم، مثال زیر نمونه ای از تعریف ساختمان بیتی را نشان می دهد.

توجه : اگر عنصری از ساختمان بیتی به طول یک بیت باشد حتماً باید از نوع unsigned تعریف شود. چون برای یک بیت نمی توان حالت علامتدار تعریف کرد.

```
struct device
{
    unsigned active:1;
    unsigned ready:1;
    unsigned xmt_error:1;
};
```

بعنوان مثال برای ذخیره سازی اطلاعات دانشجو برای صرفه جویی در مصرف حافظه می توان برای برخی از فیلدها که تنها یک بیت لازم دارند می توان از ساختمانهای بیتی استفاده کرد. بعنوان مثال :

۱-فارغ التحصیل (شده یا نه؟) ۲- مشروطی (شده یا نه؟) ۳- تعهد(مجرد یا متاهل؟)

همچنین برخی از حالات وجود دارد که برای ذخیره سازی اطلاعات آن تعداد بیت های محدودی مورد نیاز است ، به عنوان مثال شماره ی ترم ، این شماره می تواند برای دوره ی کارشناسی مابین ۱ تا ۱۲ باشد پس برای ذخیره سازی اطلاعات آن چهار بیت کافی می باشد .

توجه : میزان حافظه ای که برای یک ساختمان بیتی مورد نیاز است می تواند از رابطه ی زیر بدست آید:

$$N = \lceil S \div 8 \rceil$$

که در این رابطه N تعداد بایتهای مورد نیاز و S مجموع تعدادبیتهای ساختمان بیتی می باشد.

می توان داخل ساختمان از ترکیب ساختمان های معمولی و ساختمان های بیتی استفاده کرد. برای دسترسی به عناصر ساختمان بیتی نیز همانند ساختمان های معمولی می باشد ، یعنی اینکه می توان با عملگر ' ' به عناصر ساختمان دسترسی پیدا کرد. مثال: یک ساختمان برای ذخیره سازی تاریخ تعریف کنید که مصرف حافظه ی بهینه داشته باشد .

```
struct date
{
    unsigned int year : 7 ;
    unsigned int month : 4 ;
    unsigned int day : 5 ;
};
```