فهرست مطالب

فصل اول

الگوریتم

و

فلوچارت

۱- الگوریتم و فلوچارت

۱-۱- 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 کاربرگرفته آنها را با هم دیگر جمع کرده و نتیجه را نمایش دهد.»

1. Start
2. Input(a)
3. Input(b)

} Input a,b

4. Output (a+b)
5. End

1. Start
2. Input(a,b)
3. c ← a+b
4. Output (c)
5. End

شکل ۱ : دو جواب متفاوت برای مثال ۱.

هر الگوریتمی می تواند فقط یک سطر start و یک سطر end داشته باشد یعنی این که هر الگوریتمی در یک سطر شروع شده و در سطر دیگر خاتمه پیدا می کند.

مثال۲: 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 .

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

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

۳) اگر $\Delta < 0$ 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 ۲ : 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. $x1 \leftarrow \frac{-b+\sqrt{\Delta}}{2a}$
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 ۳ : الگوریتم حل معادله درجه ۲

9. start
10. input a,b,c
11. if a=0 then goto 2
12. $\Delta \leftarrow b^2-4ac$
13. if $\Delta<0$ then output (’not result’)
14. if $\Delta \geq 0$ then output ($x1 \leftarrow \frac{-b+\sqrt{\Delta}}{2a}, x_2 \leftarrow \frac{-b-\sqrt{\Delta}}{2a}$)
15. end

شکل ۴ : الگوریتم حل معادله درجه ۲

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

مثال ۳ : 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 ۵ : 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 ۶ : الگوریتم محاسبه a^b با استفاده از حلقه for (برای آشنایی)

۱-۲- فلوچارت

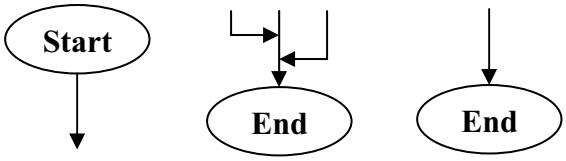
با استفاده از فلوچارت می توانیم روند کاری (الگوریتم) را با استفاده از نمادهای خاص نمایش دهیم که استفاده از این نمادها برای فهم الگوریتم کمک خواهد کرد.

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

در فلوچارتها به ترتیب اشکال زیر وجود خواهد داشت:

۱-۲-۱- 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.

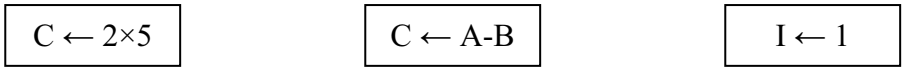


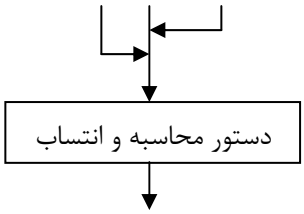
شکل ۷ : نمادهای شروع و پایان

۱-۲-۲- 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 :





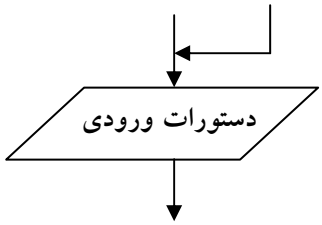
شکل ۸ : نماد جایگزینی و انتساب

۱-۲-۳- 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.

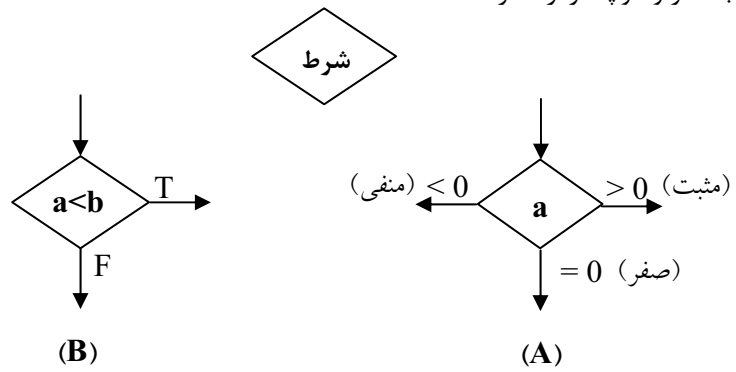
این نماد نیز میتواند چندین فلش ورودی و یک فلش خروجی داشته باشد .



شکل ۹ : نماد ورودی

۱-۲-۴- نماد شرطی

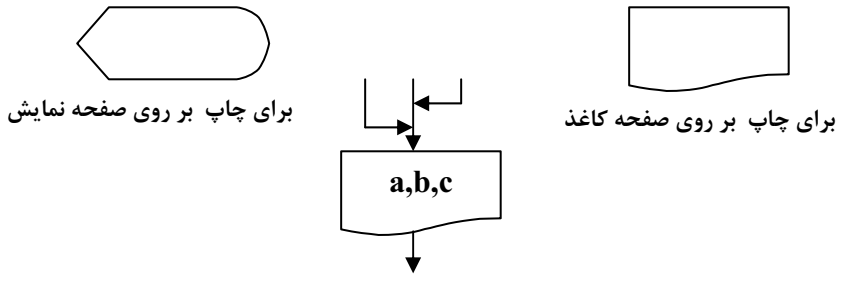
با استفاده از این نماد می توانیم شرط ها را کنترل کنیم و این بسته به شرط (درست یا نادرست) مسیر T (درست) یا F (نادرست) دنبال میشود. شکل 10-A شرط a<b را کنترل می کند این شرط می تواند دو حالت داشته باشد درست یـا نادرسـت و در هـر حالـت مسیری را که ادامه می دهیم با استفاده از f , t نمایش داده شده است . در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



شکل ۱۰ : نماد شرط در دو حالت A) با سه فلش خروجی B) با دو فلش خروجی

۱-۲-۵- 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.

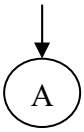


شکل ۱۱: چاپ متغیرهای a، b و c

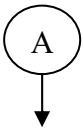
این نماد نیز می تواند مانند اکثر نمادها می تواند چندین فلش ورودی و فقط یک فلش خروجی داشته باشد.

۱-۲-۶- نماد ادامه

مواقعی 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.



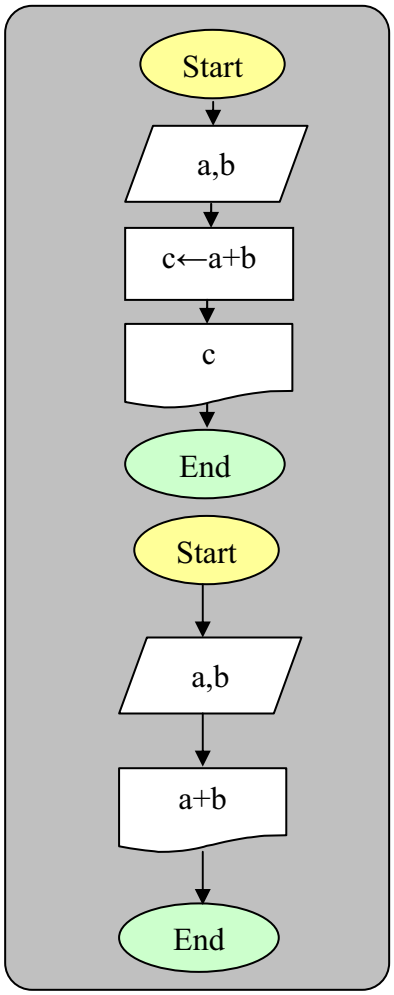
در یک صفحه فلش به این نماد ختم می شود.



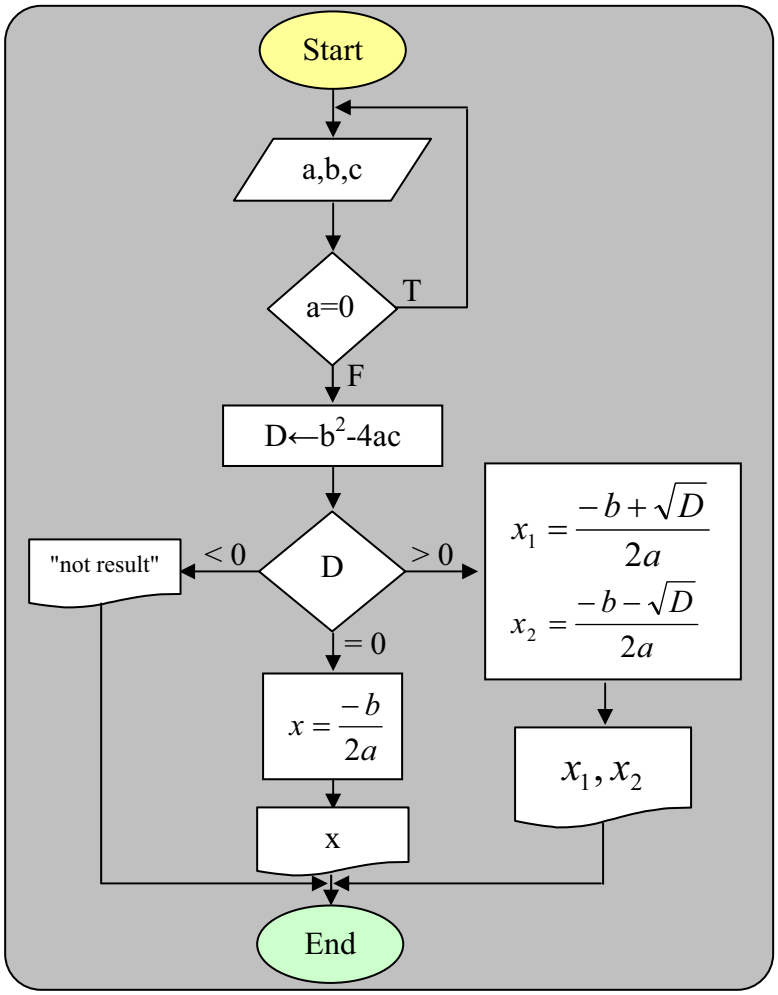
در یک صفحه دیگر فلش از این نماد ادامه می

مثال۴: فلوچارت مثال ۱ را رسم کنید ؟

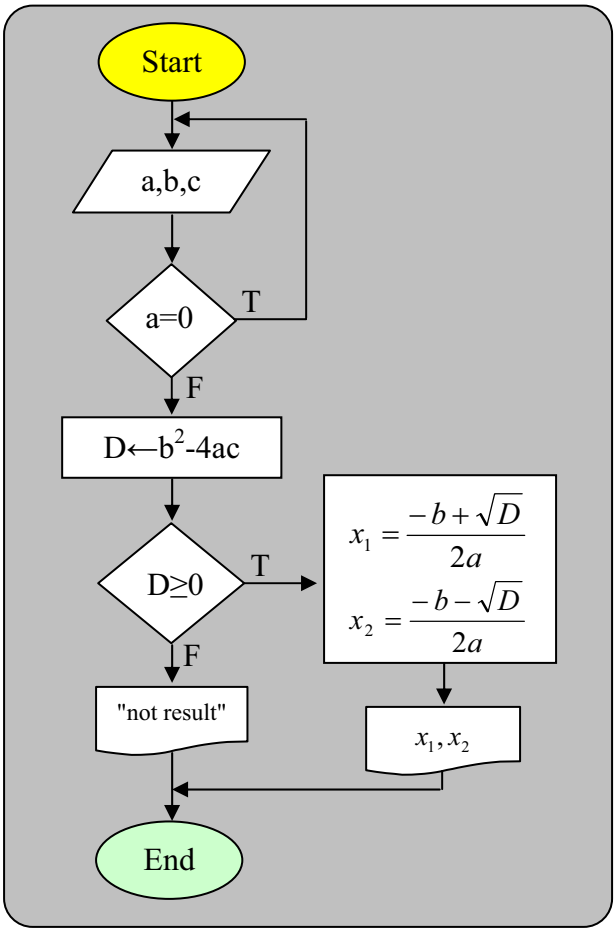
مثال ۵: فلوچارت الگوریتم حل ax2+bx+c را رسم کنید.



شکل ۱۲: فلوچارتهای مربوط به الگوریتمهای شکل ۱



شکل ۱۳: فلوچارتهای مربوط به الگوریتم معادله درجه ۲ (شکل ۲)



شکل ۱۴: فلوچارتهای مربوط به الگوریتم معادله درجه ۲ (شکل ۴)

مثال ۶: فلوچارتی رسم کنید که سه عدد a , b , c را از کاربر گرفته سپس این اعداد را به ترتیب صعودی مرتب کند.

مثال۷: 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 شکلهای ۱۶، ۱۷ و ۱۸ ارائه شده است.

مثال ۸ : فلوچارتی برای محایبه جواب سری مقابل را رسم کنید. $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{(n-1)!} + \frac{1}{n!}$

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

1. n!=1*2*3*…*n
2. n!=(n-1)!*n

شکل ۱۹ نمونه فلوچارتی را با استفاده از روش دوم نشان می دهد برای مشاهنده نمونه فلوچارتهایی از روش اول می توانید به کتاب مراجعه نمایید

مثال ۹ : فلوچارتی برای محایبه جواب سری مقابل را رسم کنید. $e = 1 - \frac{1}{2!} + \frac{1}{3!} + \cdots + (-1)^{n-1} \times \frac{1}{ni}$

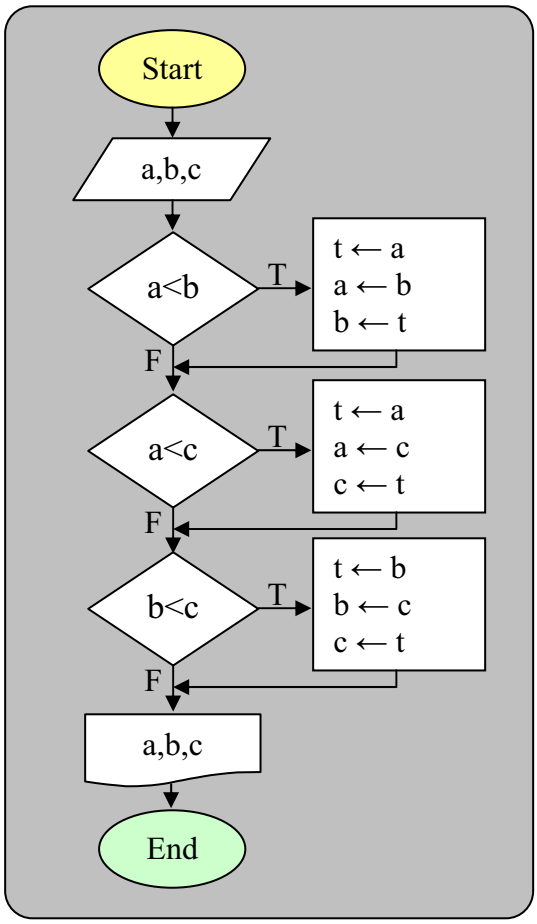
مثال ۱۰: فلوچارتی برای محایبه جواب سری مقابل را رسم کنید. $\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots + (-1)^n \times \frac{x^2 n}{(2n)!}$

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

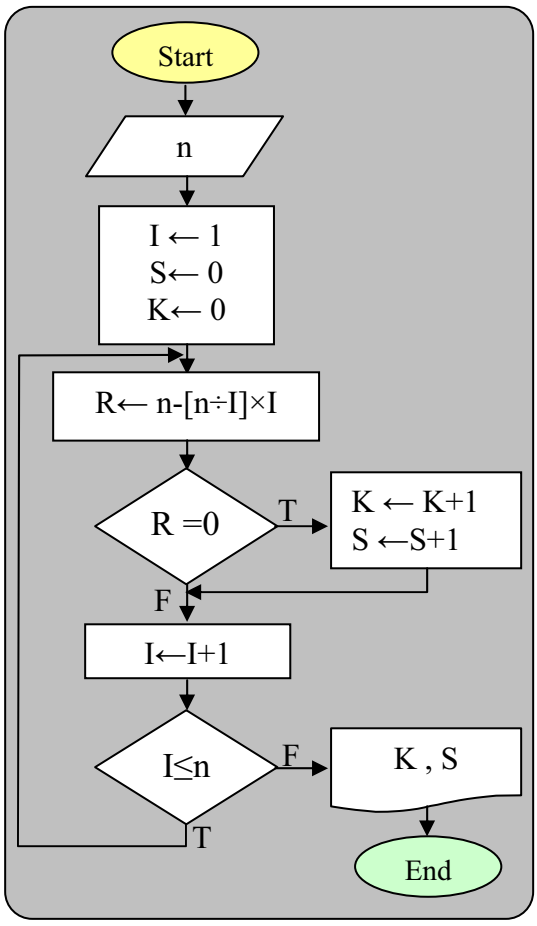
$x^n = (x^{n-2}) \times x \times x$ و $n! = (n-2)! \times (n-1) \times n$

مثال ۱۱: فلوچارت رسم کنید که n جمله از سری فیبوناچی را تولید کنید.

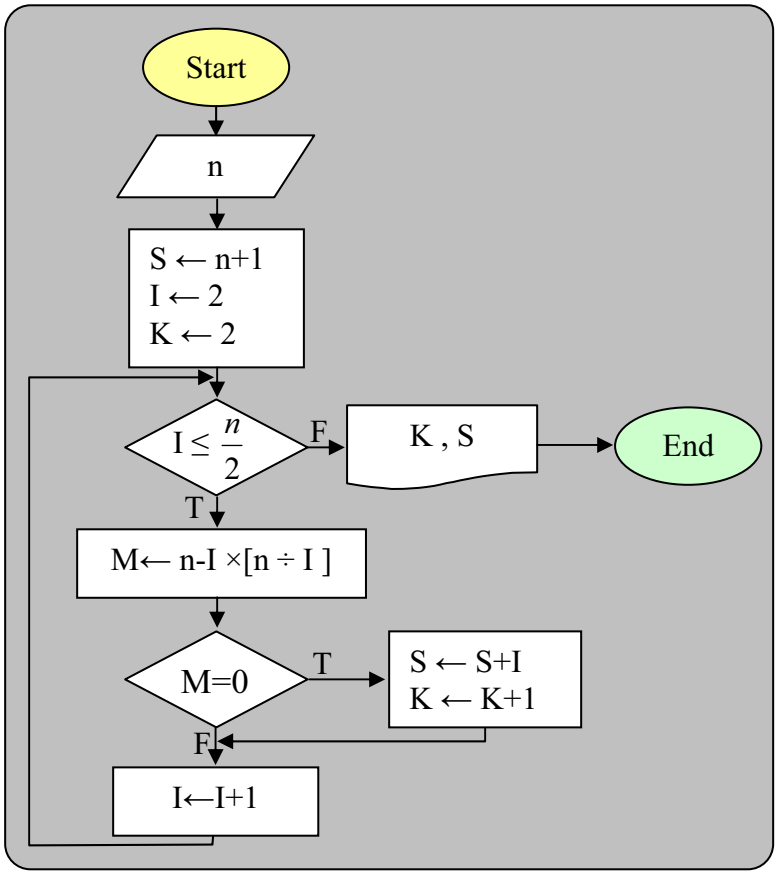
1,1,2,3,5,8,13,. . .



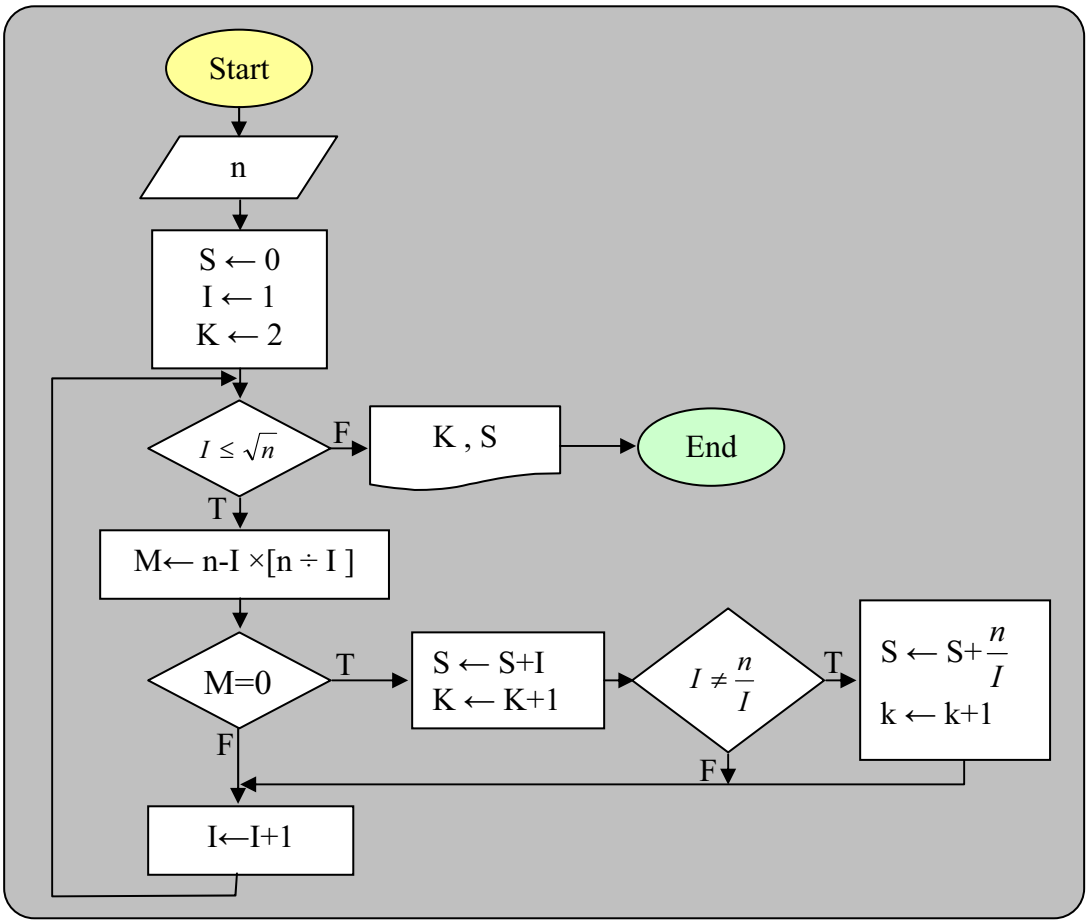
شکل ۱۵: فلوچارت مثال ۶



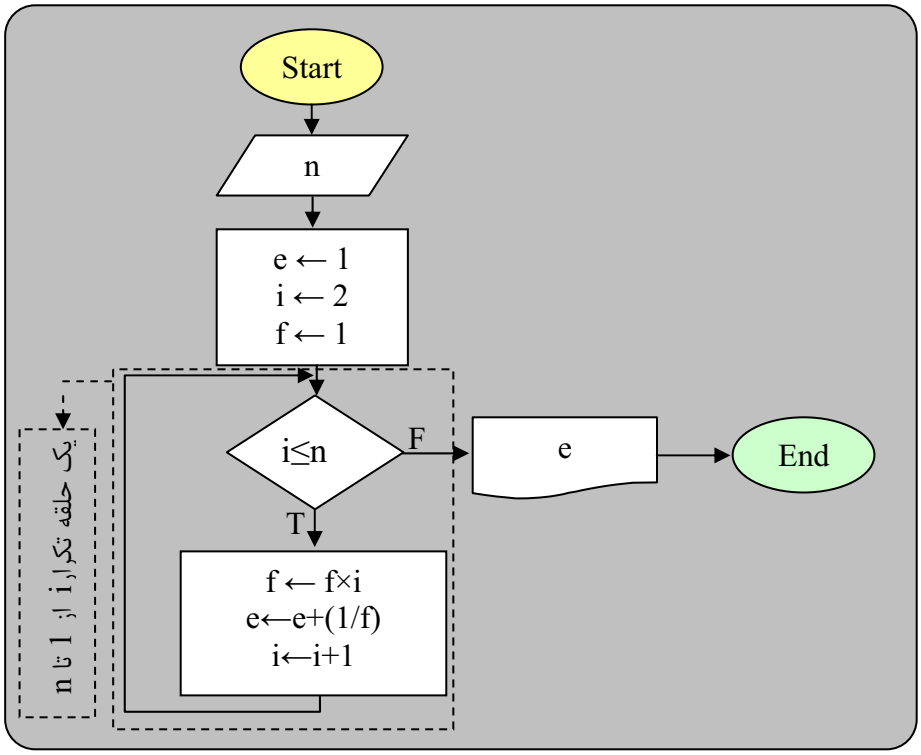
شکل ۱۶: فلوچارت مثال ۷



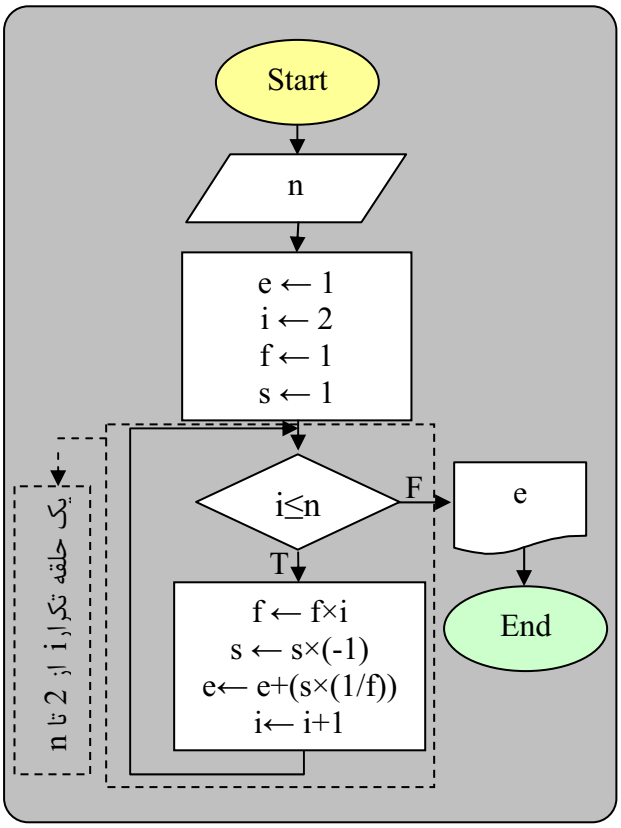
شکل ۱۷: فلوچارت مثال ۷



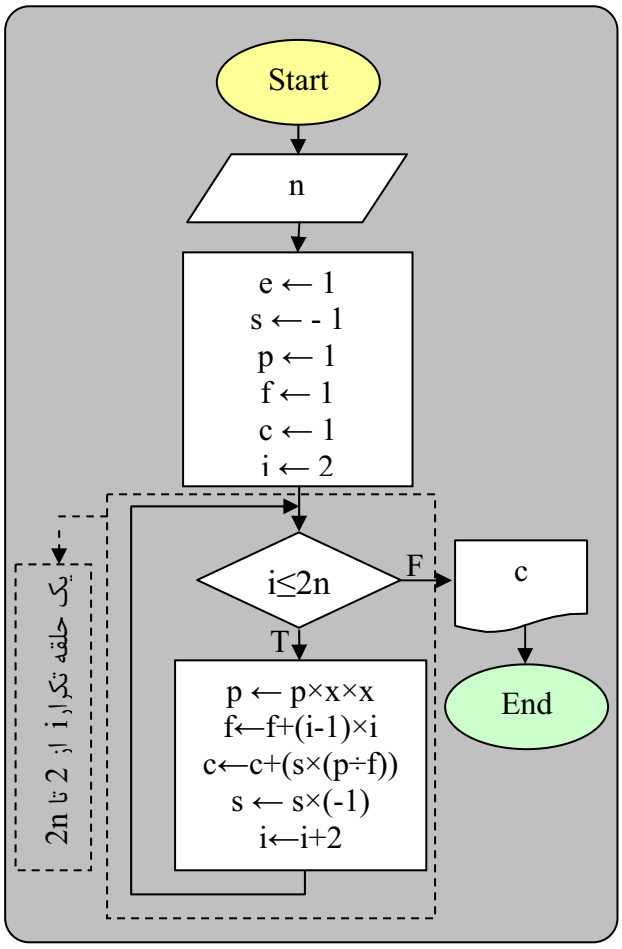
شکل ۱۸: فلوچارت مثال ۷



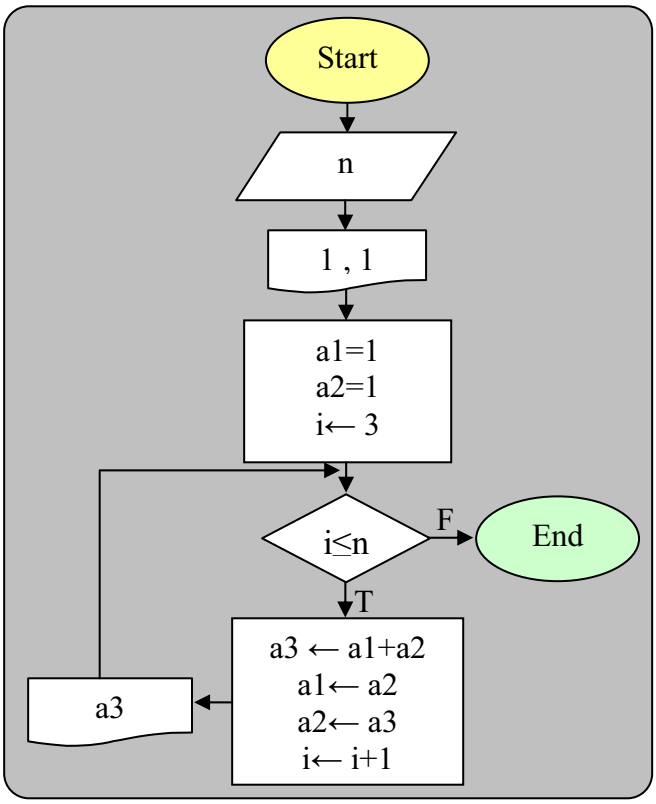
شکل ۱۹: فلوچارت مثال ۸



شکل ۲۰: فلوچارت مثال ۹



شکل ۲۱: فلوچارت مثال ۱۰



شکل ۲۲: فلوچارت مثال ۱۱

۱-۳- 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 معمولا از اعداد استفاده می شود شروع این اعداد می تواند از 0 ویا 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 .

۱-۳-۱- آرایه یک بعدی

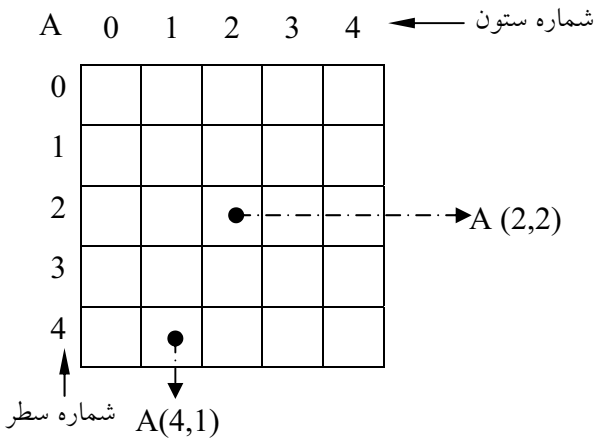
شکل زیر نمونه ای از آرایه یک بعدی با هشت سلول و با نام A را نشان می دهد.

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

با اجرای دستوراتی بعنوان مثال 5 ←A(2) عدد 5 در اندیس 2 (خانه سوم) آرایه A قرار می‌گیرد همچنین با اجرای 0 ←A(7) عدد 0 در اندیس 7 (خانه هشتم) از آرایه A قرار می‌گیرد.

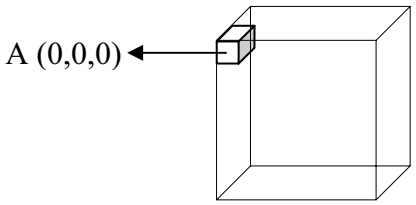
۱-۳-۲- آرایه دوبعدی

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



۱-۳-۳- آرایه سه بعدی

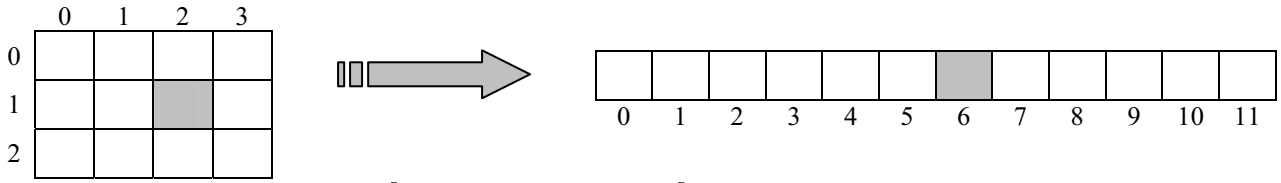
آرایه سه بعدی را می توان بصورت یک مکعب با سلولهای حافظه در نظر گرفت برای دسترسی به هر یک از حانه های آن باید به نرتیب شماره های عمق ، سطر و ستون را مشخص کنیم.



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 درنظر بگیریم، استفاده از آرایه ها این مشکل را برای ما حل کرده است.

هر نوع آرایه باید نهایتا به یک آرایه یک بعدی نگاشت شود. نحوه نگاشت یک آرایه دو بعدی به یک آرایه تک بعدی بصورت زیر می‌باشد.

بعوان مثال فرض کنید یک آرایه 4×3 بصورت شکل زیر وجود دارد برای این آرایه دو بعدی باید یک آرایه متناظرتک بعدی برای نگاشت ایجاد کنیم، در این مثال تعداد خانه‌های آرایه باید برابر 12 خانه (12=4×3) باشد.



اگربخواهیم بعوان 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:

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

مثال۱۲: عدد طبیعی n مفروض است فلوچارتی رسم کنید که معادل این عدد را در مبنای ۲ بنویسد.

مثال ۱۳: فلوچارتی رسم کنید که دو بردار nبعدی (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: $A_0,A_1,A_2,\ldots,A_{n-1}$

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

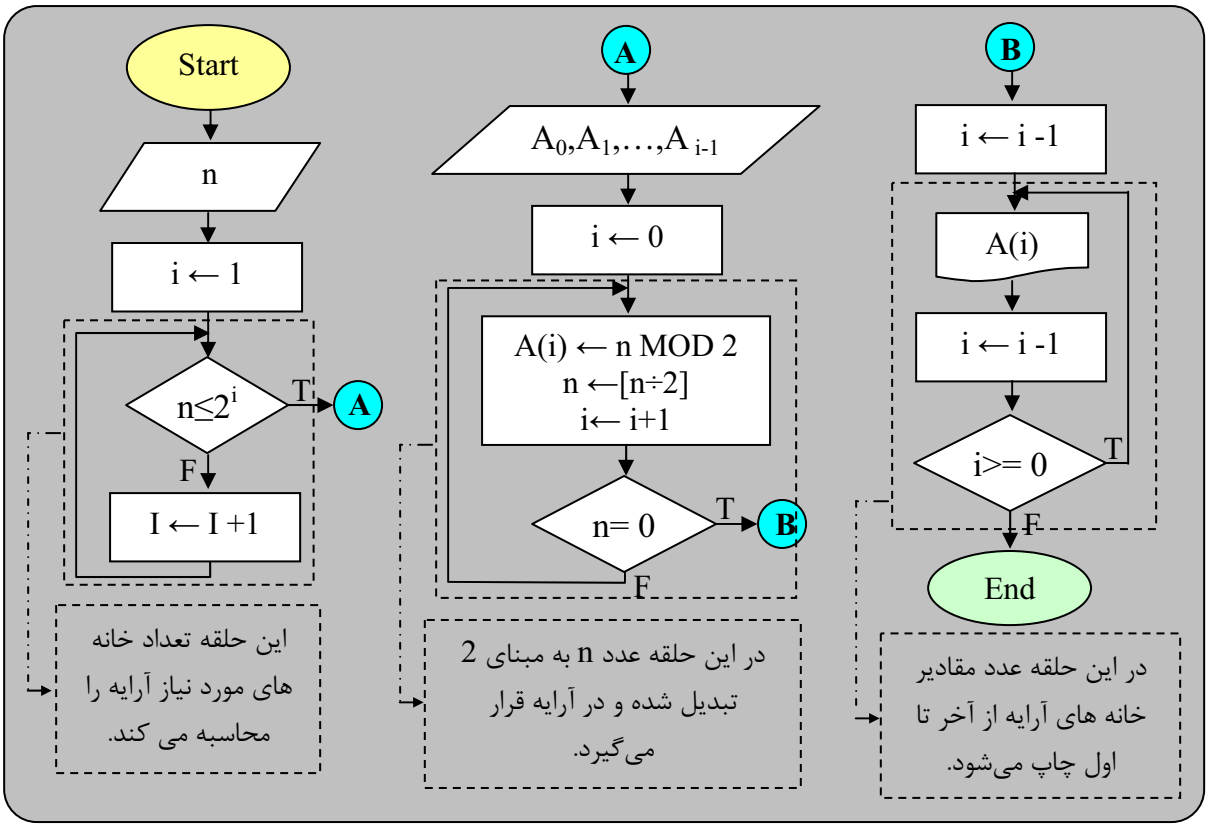
$A+B=(A_1+B_1, A_2+B_2, \ldots, A_n+B_n)$

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

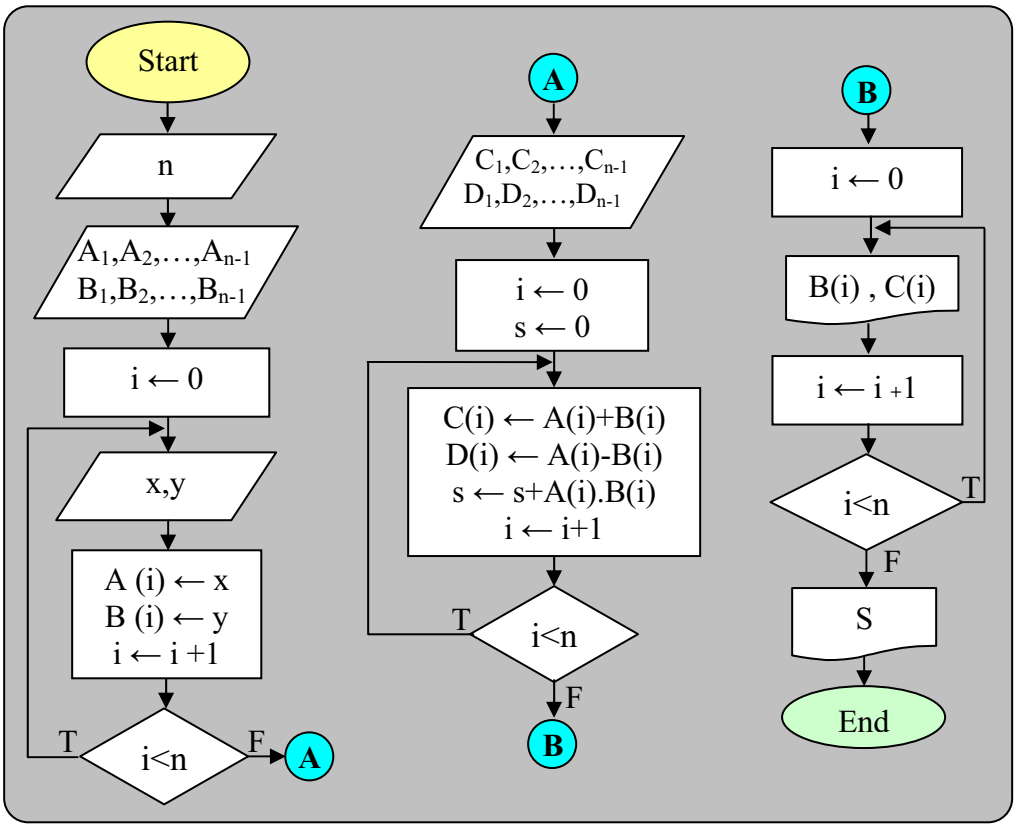
$A \cdot B=(A_1.B_1+A_2.B_2+\ldots+A_n.B_n)$

مثال ۱۴: فلوچارتی رسم کنید که n جمله از سری فیبوناچی را تولید کرده در داخل آرایه قرار دهد.

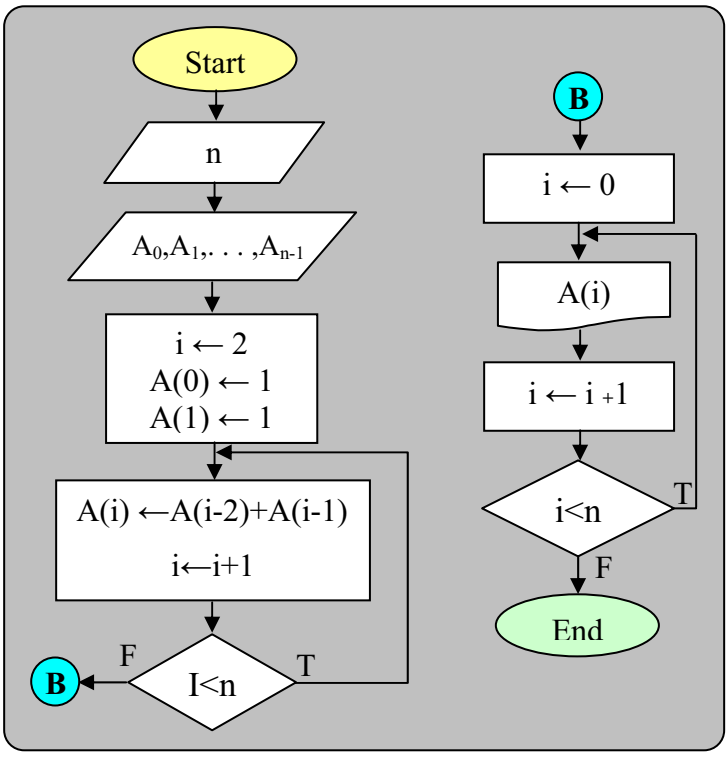
مثال ۱۵: فلوچارتی رسم کنید که عدد طبیعی N را سؤال و N خانه در نظر گرفته و N عدد دلخواه را یکی یکی سؤال نموده و داخل آنها قرار دهد و ابتدا میانگین آنها را محاسبه کند سپس مشخص کند چند عدد از میانگین بزرگتر و چند عدد کوچکتر و چند عدد با میانگین برابر است.



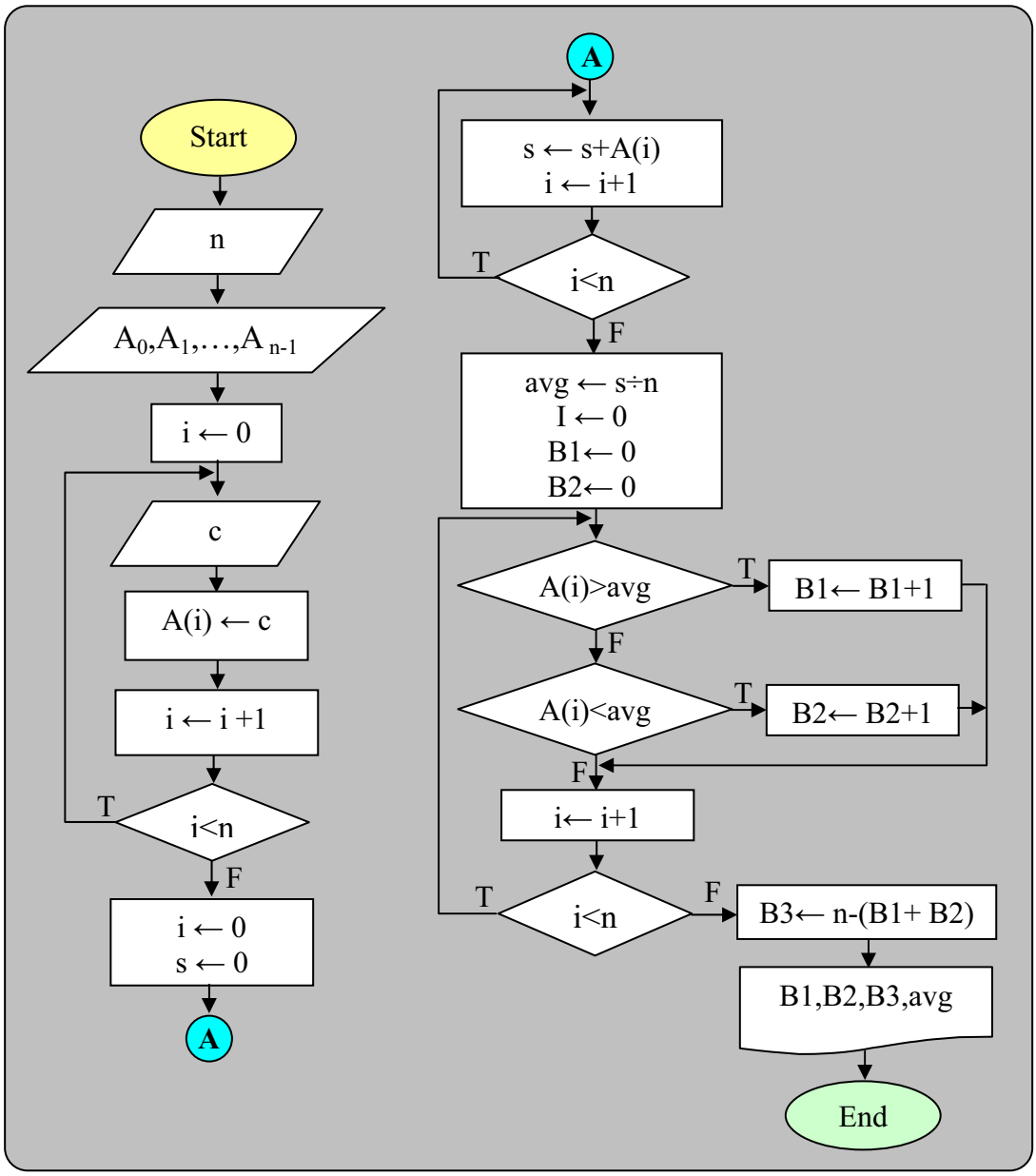
شکل ۲۳: فلوچارت مثال ۱۲



شکل ۲۴: فلوچارت مثال ۱۳



شکل ۲۵: فلوچارت مثال ۱۴



شکل ۲۶: فلوچارت مثال ۱۵

فصل دوم

برنامه نویسی به زبان C

۲- برنامه نویسی به زبان C

۲-۱- 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 ۳-۱ و ۴-۱ کتاب لیست شده است

۲-۲- 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 پایین */ و /* قرار گیرد در همان سطر 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 سط باشد.

۲-۳- انواع داده‌ای

در زبان C پنج نوع داده ی 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 signed (با علامت)، unsigned (بدون علامت)، long و short می‌توان انواع داده‌ای جدیدی تعریف کرد.

بسته به نوع داده‌ای انتخاب شده فضای تعریف شده اختصاص داده می‌شود. بعنوان مثال برای نوع داده ای char

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

'A'=65 کداسکی

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

۲-۴- 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.

۲-۵- 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 درنظر 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 undrline 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.

نوع دادهای نام متغیر;

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 INTIGER (int) نشان می‌دهد.

```
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 عملگز 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 ;(&a, ’’%d ’’ ,scanf از صفحه کلید مقداری را گرفته بر روی متغیر aآنرا 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

۲-۶- 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 عبازت #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 می‌باشد:

```
const <نوع داده> <نام ثابت> = <مقدار>
 const    float   PI=3.14;
```

۲-۷- عملگرها

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

۲-۷-۱- 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 می‌شود
- -	--a یا a--	عملگر ماینس ماینس(--) عمل کاهش یک واحدی را انجام می دهد و در دو حالت استفاده می‌شود

توضیح در مورد ۶و۷: عملگر ++ و -- می تواند به دو حالت برای متغیرها اعمال شود قبل از نام متغیر قرار بگیرد یا بعد از نام متغیر. 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:

```
int   a=5, b=5;
++a;
b++;
```

مثال ۲:

```
int  a=5, b;
b=a++;
```

در این مثال پس از اجرای سطر دوم b=5 , a=6 خواهد بود ، چون اول مقدار a داخل b قرار گرفته سپس a یک واحد افزایش می‌یابد.

مثال ۳:

```
Int   a=5,b;
b=++a;
```

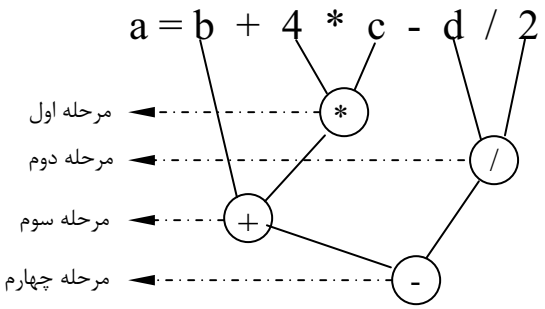
در این مثال پس از اجرای سطر دوم b=6 , a=6 خواهد بود ، چون اول a یک واحد افزایش می‌یابد سپس مقدار a داخل b قرار می‌گیرد.

تقدم این عملگرها به صورت زیر مشخص می شود.

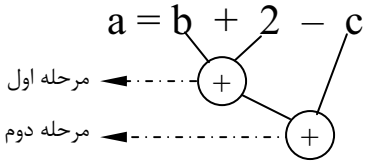
بالاترین تقدم	+ + , - -
	- (منهای یکانی)
	* , / , %
پایین ترین تقدم	+ , -

مثال: باتوجه به تقدم عملگرهای محاسباتی مراحل محاسبه در فرمل 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.



۲-۷-۲- عملگرهای رابطه ای

عملگر	نام	مثال
>	بزرگتر	**a>b**
>=	بزرگتر یا مساوی	**a>=b**
<	کوچکتر	**a<b**
<=	کوچکتر یا مساوی	**a<=b**
= =	متساوی	**a==b**
! =	نامساوی	**a!=b**

۲-۷-۳- عملگرهای منطقی

عملگر	نام	مثال
!	نقیض (not)	**!a**
&&	وَ (and)	**a>b && c<d**
||	یا (or)	**a>b || c<d**

تقدم سه عملگر محاسباتی ،رابطه ای و منطقی:

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

۲-۷-۴- عملگر 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

۲-۷-۵- عملگر بیتی

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

عملکرد هر یک از عملگرهای بیتی بصورت زیر می باشد.

~a	a ^ b	a | b	a & b	b	a
1	0	0	0	**0**	**0**
1	1	1	0	**1**	**0**
0	1	1	0	**0**	**1**
0	0	1	1	**1**	**1**

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

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

جدول ۱۵-۱ کتاب نمونه مثالهیی را از عملگر های بیتی shift left و shift right را نشان می‌دهد. شکل کلی این عملگرها بصورت زیر می‌باشد.

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 بیتهای عدد عمل خواهند کرد. در زبان C ۶ نوع عملگر بیتی وجود دارد که در بالا نوشته شد.

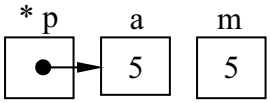
توجه: شیفت ۱ واحد به سمت راست عمل تقسیم بر ۲ را انجام خواهد داد و شیفت ۱ واحد به سمت چپ همان عمل ضرب در ۲ را انجام خواهد داد.

۲-۷-۶- عملگرهای * و &

این عملگرها بیشتر در بخش مربوط به اشاره‌گرها 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 اشاره‌گر 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 mقرار خواهد گرفت.

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



۲-۷-۷- عملگر ؟

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

;<عبارت۳> : <عبارت۲> ? <عبارت۱> = 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 گیرددر غیر این صورت مقدار a-b داخل متغیر a قرار خواهد گرفت.

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

۲-۷-۸- 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.دستورsizeof 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);
```

۲-۸- 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 در این محاسبه 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.

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 داده ای از بین نخواهد رفت اما در غیر این صورت ممکن است در تبدیلات نوع برخی از داده ها 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.

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

۳- 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(هدرفایل) مورد استفاده قرارگرفته (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. }
```

۳-۱- 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.

۳-۱-۱- تابع ( ) printf

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

```
printf("<عبارت۱>",<عبارت۲>);
```

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 داده‌ها 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 ۱-۲ 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 فرمت**: این کاراکترها نوع اطلاعاتی را که در عبارت ۲ قرار دارند و باید بجای این کاراکتر چاپ شوند، مشخص میکند. کاراکترهای فرمت با علامت ٪ شروع می شوند. پرکاربردترین کاراکترهای فرمت در جدول ۱-۲ نشان داده شده اند.

۳. **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 می‌شود 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.r\ 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 می‌دهد.

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

برای ***پاکسازی صفحه تصویر*** می توانیم از تابع clrscr( ) که در فایل stdio.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 هرکدام 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 می‌شود.

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

۳-۱-۲- 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، کهشکل کلی آن در زیر نشان داده شده است.

```
scanf("<عبارت۱>",<عبارت۲>);
```

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

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);
```

۳-۱-۳- 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 می‌شود.

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 می‌توان 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.

۳-۱-۴- 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.

۳-۱-۵- نوشتن کاراکتر با توابع ( )putch و ( )putchar

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

۴- حلقه های تکرار و ساختارهای تصمیم

۴-۱- 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.

```
for  (گام حلقه ; شرط حلقه ; مقداراولیه اندیس حلقه)
{
    دستور۱
    دستور۲
    ...
    دستورn
}
```

در بخش اول متغیرحلقه مقدار دهی اولیه می‌شود. 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 می‌شود. 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 مقداردهی اولیه کنیم یا اینکه اصلا 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 مقداردهی اولیه انجام دهیم بخش اول را خالی می‌گذاریم. 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 می‌توان شرط پایان حلقه را حذف کرد که در این حالت حلقه به صورت بی‌نهایت به کار خود ادامه خواهد داد در این حالت برای پایان کار حلقه از دستورات شکنندهٔ حلقه استفاده می‌کنیم.

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

در حلقه‌های زبان C این امکان وجود دارد که در گام‌های حلقه بیش از یک متغیر به روزرسانی شود در این حالت باید مابین هر یک از بخش‌ها از کارکتر کاما "," استفاده کنیم. همچنین میتوان ابن بخش را از حلقهٔ for حذف کرد در این حالت باید عمل به روزرسانی در داخل بدنهٔ حلقهٔ 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;<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.

مثال۱: 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 ave;
 clrscr( );
 for(i=0;i<20;i++)
 {
  printf("\n Enter a number %d:",i+1);
  scanf("%d",&num);
  sum+=num;
 }
 ave=(float)sum/num;
 printf("the average is:%6.2f ",ave);
 getch( );
 return 0;
}
```

برنامه ۱: میانگین ۲۰ عدد

۴-۲- 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 نمی‌دهد.

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

مثال ۲: برنامه ای بنویسید که یک جدول ضرب ۱۰×۱۰ 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 ۲ : جدول ضرب 10×10

۴-۳- حلقه while

ساختار تکرار while شرط را کنترل کرده و تا زمانیکه این شرط برقرار باشد دستورات بدنه حلقه اجرا می شود و بـه محـض نقـض شدن شرط ، اجرای این حلقه خاتمه پیدا می کند. در حلقه های while شرط حلقه در ابتدای حلقه کنترل مـی شـود. دوحالـت بـرای استفاده از حلقه while وجود دارد.

```
while ( شرط )
      دستور

while ( شرط )
{
     دستور۱
     دستور۲
     ...
     دستورn
}
```

مثال۳: برنامه ای بنویسید که تا زمانی که کاربر بخواهد اعدادی را از کاربر گرفته و میانگین این اعداد را محاسبه و چاپ نماید.

```
#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("Avy=%f",(float)s/i);
  return 0;
}
```

برنامه ۳: میانگین تعدادی عدد

۴-۴- حلقه do…while

ساختار تکرار do…while مانند ساختار تکرار while است، با این تفاوت که در ساختار while، شرط حلقه در ابتـدای حلقـه تـست 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",(++);
while(i<100);
```

۴-۵- 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 نمی‌شود. 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 for, do… while, while, if لزومی ندارد که فقط از عملگرهای رابطه ای یا منطقی استفاده کنیم. می توان از هر عملگر دیگر نیز استفاده کرداگر نتیجه محاسبه عدد صفر باشد به مفهوم 0=false , هر عدد غیر صفر به مفهوم ~0=True می باشد برای مثال:

if(!A) اگر A=0 باشد

if(A) اگر A≠0 باشد

مثال۴: برنامه ای بنویسید که عدد صحیح و مثبت N 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 ۴ : برنامه مثال ۴

۴-۶- دستورbreak

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 ساختارswitch می باشد.

۴-۷- 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.

۴-۸- ساختار تصمیم switch

ساختار کلی این دستور بصودت زیر می باشد:

```
switch  (عبارت)
{
        case  <مقدار۱>:
                <دستورات۱>
                break;
        case  <مقدار۲>:
                <دستورات۲>
                break;
                  .
                  .
                  .
        defult:
                <دستوراتn>
}
```

شاید برخی مواقع نیاز باشد که بخواهیم مقدار یک متغیر را برای حالتهای مختلف بررسی کنیم در این حالت اگر ازدستور if استفاده کنیم نیاز به استفاده از چندین else if 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 caseها 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 مقدارcase بعدی or شود مثال زیر مقدار متغیر c را برای حالتهای R,G,Y تست می‌کند و بسته به حالت مورد نظر پیام مناسب را چاپ خواهد کرد. این مثال با استفاده از دستور switch, if به صورت زیر خواهد بود:

```
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

```
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

دستور switch راحتترو انعطاف پذیرتر از if…else می‌باشد. بعنوان مثال اگر در مثال بالا اگر بخواهیم مابین حروف کوچک یا بزرگ تفاوت وجود نداشته باشد می‌توان 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 وهمچنین دستورات switch را می توان به صورت تو در تو استفاده کرد.

مثال۵: 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 '/ ' :
   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 ۵: جواب مثال ۵

۵- توابع و کلاسهای حافظه

۵-۱- توابع

در پایین ساختار توابع در زبان C آورده شده است.

```
<نوع تابع>  نام تابع  (لیست پارامترها)
{
        <دستور۱>
        <دستور۲>
        .
        .
        .
        <دستورn>
}
```

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 می‌گیرد در این لیست می‌توانیم ۰ تا N پارامتر را قرار دهیم و باید مابین پارامترها از عملگر "," استفاده کنیم.

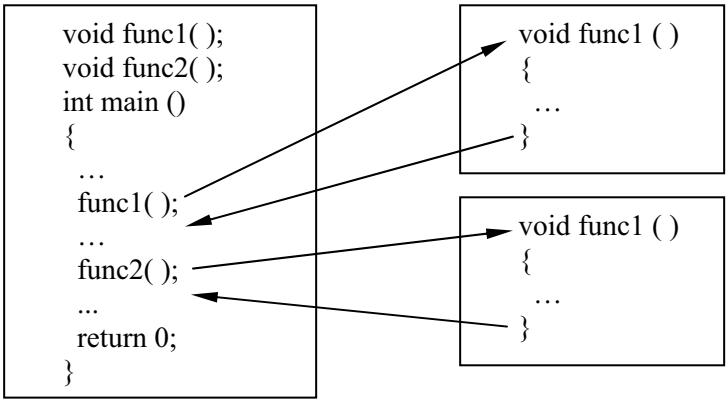
بدنه توابع : این بخش مشخص کننده بدنه توابع بوده و شامل دستورات اجرایی زبان C می‌باشد.

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.

```
#include<stdio.h>
void somple(int x , int y);          الگوی تابع
int main( )
{                          آرگومانهای تابع
 int a,b ;
 ...
 sample(a,b);              فراخوانی تابع
 ...
 return 0;                          پارامترهای تابع
}
void sample (int x , int y)          عنوان تابع
{
 printf("\n x = %d, y = %d", x, y);          بدنه تابع
…
}
```

شیوه به کارگیری توابع در برنامه



چگونگی عملکرد توابع.

شکل بالا چگونگی عملکرد توابع را نشان می‌دهد، هنگام فراخانی 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.

مثال۶ : تابعی بنویسید که سه عدد را به عنوان ارگومان 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 ۶ : برنامه مربوط به مثال ۶

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 ۷: 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 ۷: برنامه مثال ۷ برای محاسبه مجموع ارقام عدد اعشاری تا ۴ رقم اعشار.

۵-۱-۱- روش های ارسال پارامتر ها به توابع

به دو طریق می‌توان 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 می‌شود پس هر گونه تغییر روی پارامتر ها در روی آرگومان‌ها هیچ تاثیری ندارد، اما در روش فراخوانی با ارجاع آدرس آرگومان ها به پارامترها منتقل می‌شود پس در این حالت هر گونه تغییر روی پارامترها باعث خواهد شد ارگومان 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 می‌توان مقادیر را با استفاده از آرگومان ها برگشت داد.

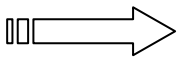
در زیر نمونه مثالی از دو روش ارسال پارامتر آورده شده است تابع f1() بروش فراخوانی با مقدار فراخوانی می‌شود بهمین دلیل همچنان که خروجی را مشاهده می‌کنید با تغیر پارامترa داخل تابع i هیچ تغیری نمی‌کند (همچنان 0 باقی می‌ماند) اما تابع f2() بروش فراخوانی با ارجاع فراخوانی می‌شود بهمین دلیل همچنان که خروجی را مشاهده می‌کنید با تغیر پارامترa* داخل تابع i تغیر می‌کند (برابر 5 می‌شود).

```
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
}
```

فراخوانی با مقدار

فراخوانی با ارجاع

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;
}
```

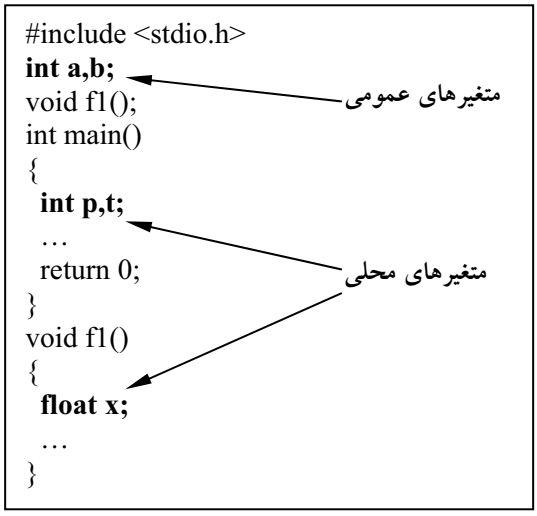
برنامه ۸: برنامه مثال ۷ برای محاسبه مجموع ارقام عدد اعشاری.

۵-۱-۲- متغیرهای محلی و عمومی

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

۱. داخل توابع
۲. خارج از توابع

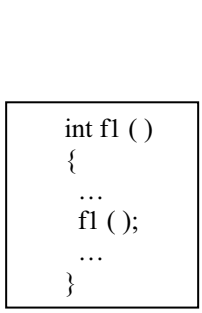
متغیر هایی که داخل توابع تعریف شوند به عنوان ***متغیرهای محلی (local variables)*** شناخته می شوند و متغیر هایی که خارج از توابع تعریف شوند به عنوان ***متغیرهای عمومی (global variables)*** شناخته می‌شوند. متغیرهای محلی فقط برای همان تابع شناخته شده و خارج از تابع داخل توابع دیگر قابل دسترسی و تغییر نمی‌باشند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

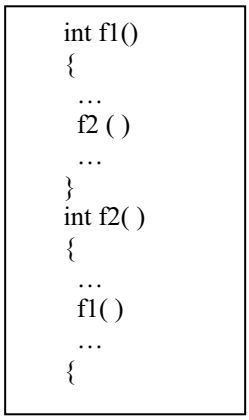


اگر داخل توابع متغیر محلی همنام با متغیر سراسری تعریف شود داخل آن تابع متغیر سراسری همنام با متغیر محلی دیگر قابل دسترسی و استفاده نخواهد بود بلکه از متغیر محلی استفاده می شود.

۵-۱-۳- روش بازگشتی

اگر تابع مستقیماً خودش را فراخوانی کند و یا با واسطه خودش را فرا خوانی کند مثلا تابع f1 و f2 را فراخوانی کرده و f2 نیز f1 را فراخوانی کند به این روش نوشتن توابع روش بازگشتی گفته می‌شود.





مثال۸: تابع فاکتوریل را به دو روش بازگشتی و غیر بازگشتی بنویسید.

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

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

روش بازگشتی

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

روش غیر بازگشتی

برنامه ۹: جواب مثال ۸

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 if (n==1) این عمل را انجام میدهد در غیر این صورت برنامه داخل حلقه بی‌نهایت گیر خواهد کرد.

مثال۹: 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×B می‌توان A ها را به تعداد B بار با هم جمع کرد (A × B = A+A+A+…+A) یا B ها را به تعداد A بار با هم جمع کرد (A × B = B+B+B+…+B).

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

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

روش بازگشتی

```
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

برنامه ۱۰: جواب مثال ۹

مثال : 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÷B می‌توان از A هر بار B را کم کرد تا زمانی که A≥B است تعداد این کم کردنها حاصل تقسیم را مشخص می‌کند.

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

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

روش بازگشتی

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

روش غیر بازگشتی

برنامه ۱۰: جواب مثال ۹

مثال ۱۰ : برنامه ای بنویسید که n جمله از سری فیبونانجی را به روش بازگشتی تولید کند ؟

```
void fib(int a , int b , int i, int n)
{
  int t;
  if (i== n) return;
  else
  {
    t =a+b;
    printf (" , %d ",t);
    fib(b,t,i+1,n);
  }
}
```

برنامه ۱۰: جواب مثال ۱۰

توضیحات: در این برنامه پارامترهای a و b دو جمله قبلی پارامتر i تعداد جمله هایی که تولید شده و پارامتر n مشخص کننده تعداد جملاتی است که باید تولید شود می‌باشد.

۵-۲- 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

منظوراز حوزه متغیر محل هایی از برنامه است که متغیر قابل دسترسی می‌باشد و منظور از طول عمر مدت زمانی است که متغیر در حافظه باقی می‌ماند (زمانی که متغیر در حافظه ایجاد شده و زمانی که از حافظه حذف می‌شود).

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;
```

۵-۲-۱- کلاس حافظه اتوماتیک

متغیرهایی که داخل توابع تعریف می‌شوند یا همان متغیر های محلی به عنوان متغیرها با کلاس حافظه اتوماتیک می‌باشند. حوزه برای این نوع کلاس حافظه داخل تابع می‌باشد و طول عمر ان 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;

۵-۲-۲- 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 متغیر ها 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 محدودیت هایی دارد که عبارتند از :

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

۵-۲-۳- 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 متغیر ها فقط یک بار عمل مقدار دهی اولیه انجام می شود. مثال زیر نمونه ای 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;
  print f ("\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 می‌کنید مقادیر تولید شده برای b متفاوت از مقادیر a می‌باشد، به این دلیل که کلاس حافظه b از نوع استاتیک محلی می باشد.

برای ***متغیر های استاتیک سراسری*** حوزه متغیر داخل توابعی است که بعد از تعریف متغیر قرار گرفته انـد و طـول عمـر آن مـدت اجرای برنامه می‌باشد در مثال زیر متغیر های a,b از نوع متغیر استاتیک سراسری می‌باشد و برای توابع f1 ,f2 قابل دسترسی بـوده امـا برای تابع main قابل دسترسی نمی‌باشد.

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

۵-۲-۴- کلاس حافظه خارجی

این گونه متغیر ها با شروع اجرای برنامه ایجاد شده و تا پایان اجرای برنامه در حافظه باقی می مانند و همچنـین از سراسـر برنامـه قابل دسترسی می‌باشند.

دو مثال زیر نمونه از کاربرد این گونه متغیرها می‌باشد در مثال شماره ۱ برنامه به دو فایل f1.c و f2.c تقسیم شده است که متغیر static , a, b فقط داخل f1 شناخته شده و قابل استفاده می‌باشد اما متغیر m ,n داخل هر دو فایل قابل استفاده می‌باشد، ذکر عبـارت extern int m,n ; داخل تابع f2.c برای این منظور میباشد 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 میباشد کـه تـابع f1 تشخیص دهد خارج از این تابع متغیر k از نوع int به صورت سراسری تعریف شده است نوشتن این عبارت ضروری نمی‌باشد چون خود کامپایلر به صورت اتوماتیک این حالت را تشخیص می‌دهد .

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

کاربرد extern جهت اعلان متغیرهای عمومی

فایل f2.c

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

فایل f1.c

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

کاربرد متغیرهای عمومی و استاتیک عمومی

۶- آرایه ها و رشته ها

تا حال که متغیر ها تعریف می‌کردیم 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 می‌شود بتوانیم چندین سلول حافظه با یک نام را تعریف کنیم آرایه با ابعاد مختلف وجود دارد

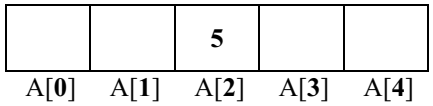
- آرایه های تک بعدی
- آرایه های دو بعدی
- و .....
- و آرایه های n بعدی

۶-۱- آرایه های تک بعدی

مثال زیر نمونه ای از تعریف یک آرایه تک بعدی به طول ۵ سلول حافظه از نوع int می‌باشد برای دسترسی به هر یک از سلول های حافظه می‌توانیم از نام آرایه و اندیس آن استفاده کنیم در زبان C شماره اندیس های آرایه از عدد صفرشروع می‌شود.

int A[5];

A[2]=5;



مثال ۱۱: برنامه ای بنوسید که نمرات ۲۰ دانشجوی کلاسی را در درس برنامه نویسی گرفته و از بین نمرات بیشترین نمره و کمترین 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 ۱۱: جواب مثال ۱۱

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

اگرعدد بازه مشخصی داشته باشد به عنوان مثال نمره (مابین صفر تا بیست) می‌توان 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 می‌دهیم اما اگر بازه اعداد مشخصی نباشد مانند عدد صحیح (از منفی بینهایت تا مثبت بینهایت) در این حالت معمولا اولین عدد گرفته شده هم به عنوان max و هم به عنوان min در نظر گرفته می‌شود سپس مقایسات انجام می‌شود طبق مثال زیر :

مثال۱۲ : برنامه ای بنوسید 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 ۱۲: جواب مثال ۱۲

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

راه حل اول ارسال آرایه با طول مشخص به توابع؛ در این حالت داخل تابع می‌توانیم طول آرایه را تشخیص دهیم.

void f1(int arr[15])

راه حل دوم ارسال آرایه با طول نامشخص به توابع؛ دراین حالت باید پارامتر دیگری داشته باشیم تا از طریق آن داخل تابع طول آرایه را تشخیص دهیم.

void f2(int arr[],int len)

و راه حل سوم استفاده از اشاره‌گرها (این راه حل بعدا در مبحث اشاره‌گرها توضیح داده خواهد شد.)

۶-۲- مرتب سازی آرایه ها

۶-۲-۱- روش مرتب سازی حبابی

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

(در این مثال مرتب سازی به صورت صعودی می‌باشد)

```
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 ۱۳: مرتب سازی حبابی

۶-۲-۲- 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 می‌شود، سپس این عدد در محل مناسب خود قرار گرفته و دوباره همین عمل بر روی اعداد باقی‌مانده انجام می‌شود 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 ۱۴: مرتب سازی انتخابی

۶-۳- 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 می‌توان نوشت در صورتی که آرایه نامرتب باشد می‌توان از روش **جستجوی ترتیبی** استفاده کرد و در صورتی که آرایه مرتب باشد یکی از الگوریتمهایی که وجود دارد **جستجوی دودویی** می باشد.

۶-۳-۱- جستجوی ترتیبی

در این الگوریتم با شروع از اولین عنصر تا آخرین عنصر بترتیب تمام عناصر با عنصر مورد نظر مقایسه می‌شود 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 ۱۵ : مرتب سازی ترتیبی

۶-۳-۲- جستجوی دودویی

در این روش جستجو که بر روی مجموعه اعداد مرتب اعمال می‌شود در بازه اعداد جستو 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 می‌شود.

در صورتی که عدد مورد نظر از عنصر وسطی بزرگتر باشد ، عمل جستجو روی بازه اعداد سمت راست عنصر وسطی انجام می‌شود.

در صورتی که عدد مورد نظر با عنصر وسطی برابر باشد در این صورت عدد مرد نظر پیدا شده است، و مراحل جستجو متوقف می‌شود .

در صورتی که این نصف کردن‌ها 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 ۱۶ : مرتب سازی دودویی

۶-۴- 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 (۱۲=۳×۴) از نوع int اختصاص داده می شود.

بدلیل خصوصیات حافظه، سلولهای حافظه پشت سر هم بصورت خطی قرار گرفته اند پس امکان پیاده‌سازی 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 دذر مبحث فلوچارتها به آن اشاره شد استفاده می شود. برای مشخص کردن تعداد خانه های آرایه تک بعدی معدل می‌توان از رابطه زیر استفاده کرد :

تعداد خانه های حافظه مورد نیاز = تعداد سطرها × تعداد ستونها

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

اندیس آرایه تک بعدی = (x × تعداد ستونها) + y

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

۶-۴-۱- 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 (حتما باید تعداد ستون را مشخص کنیم بنابه دلایلی که قبلاً توضیح داده ایم) در این حالت حتما باید پارامتردیگری برای مشخص کردن تعداد سطرها داشته باشیم.

مثال: void f1(int A[][4],int row);

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

۶-۵- مقداردهی اولیه آرایه ها

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

```
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 };
```

۶-۶- آرایه های n بعدی

در زبان C می‌توان آرایه های سه بعدی و بیشتر را تعریف کرد شکل تعریف آرایه ها با بعدها بیشتر بصورت زیر می‌باشد.

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 بع آرایه n-2 بعدی تبدیل می‌شود و به این ترتیب انجام می‌شود تا نهایتاًبه یک آرایه تک بعدی برسیم.

۶-۷- 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 نهایتا برای مشخص کردن انتهی رشته به انتهای رشته کاراکتر 0\ (NULL) افزوده شده است.

```
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\')

۶-۷-۱- مقدار دهی اولیه رشته ها

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

۶-۷-۲- 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 stedio.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);
```

۶-۷-۳- 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 ۱۳: تابعی بنویسید که رشته ای را دریافت کرده و داخل ان 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;
}
```

برنامه ۱۷ : برنامه مثال ۱۳

اگر بعنوان مثال ورودی این تابع رشته "Computer" باشد خروجی آن رشته "COMPUTER" خواهد بود.

مثال ۱۴: برنامه ای بنویسد که رشته را از کاربر دریافت کرده سپس ان 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 ۱۸: معکوس کردن رشته

مثال۱۵: برنامه ای بنویسید که رشته ای را ازکاربر 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 ۱۹: حذف کاراکتر فاصله از رشته

۶-۷-۴- انتساب رشته ها

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

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.

۶-۷-۵- مقایسه رشته ها

مقایسه رشته ها در زبان C نمیتوان به سادگی با استفاده از عملگرهای مقایسه‌ای 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 می‌دهد، اگر این دو رشته برابر باشند عدد صفر برگشت داده می‌شود اگر s1 کوچکتر از s2 باشد یک عدد منفی و اگر s1 بزرگتر از s2 باشد یک عدد مثبت برگشت داده خواهد شد.

۶-۷-۶- 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 می‌کند 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.

۶-۷-۷- آرایه ای از رشته ها

برای تعریف آرایه ای از رشته ها می‌توانیم از آرایه دو بعدی استفاده کنیم در مثال زیر یک آرایه دو بعدی با نام name تعریف شده است این آرایه می‌تواند ۵ رشته که هر رشته می‌تواند حداکثر ۱۲ کاراکتر (۱۲=۱-۱۳) داشته باشد ذخیره سازی کند.

char name[5][13];

۷- اشاره‌گرها

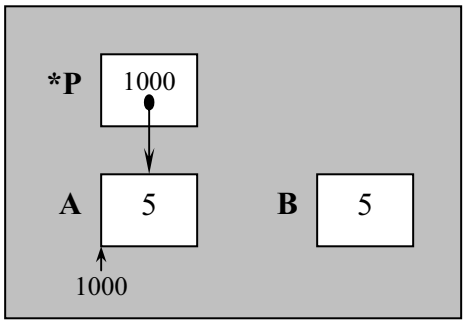
۷-۱- متغیرهای اشاره‌گر

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

```
int *p;
```

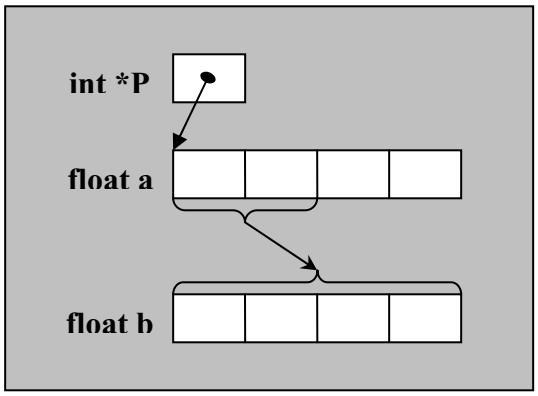
اشاره‌گری با نام p از نوع int تعریف می‌کند. مثال زیر نمونه ای از کاربرد اشاره‌گر را نشان می دهد

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



تذکر: تعریف نوع اشاره‌گر مهم است به این دلیل که نشان دهنده نوع سلول حافظه ای است که به آ ن 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 اشاره‌گر 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 می‌شود دلیل این اتفاق این است که فقط دو بایت از متغیر a داخل b (به اندازه ۴ بایت) قرار می گیرد که باعث می‌شود که عمل انتصاب بدرستی انجام نشود.

۷-۲- اعمال روی اشاره‌گرها

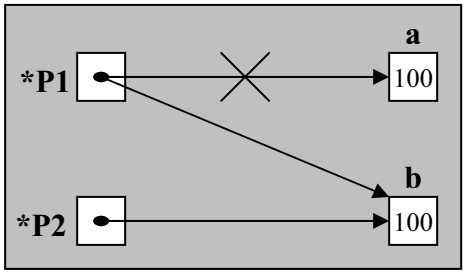
روی اشاره‌گرها می توان اعمال مختلفی را مانند متغیرهای معمولی انجام داد که عبارتند از :

۱. عمل انتساب اشاره‌گرها به یکدیگر
۲. اعمال محاسباتی جمع و تفریق
۳. اعمال مقایسه‌ای

۷-۲-۱- عمل انتساب اشاره‌گرها 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;
```

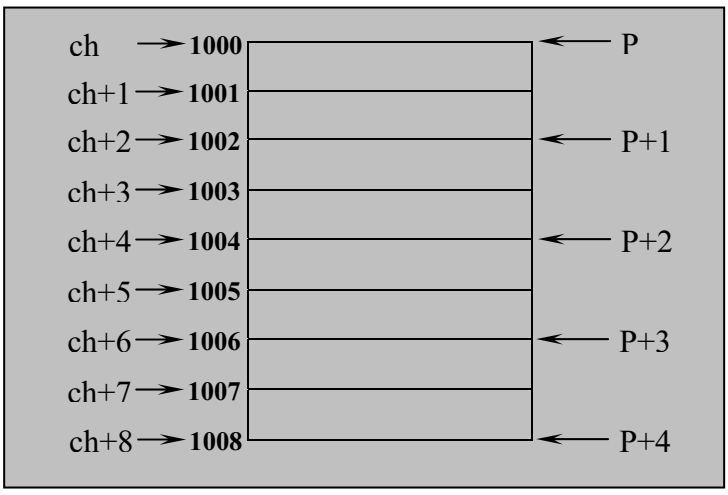


تذکر: در مثال بالا پس از اجرای سطر ۴ محتوای خانه ای که P2 به آن اشاره دارد (متغیر b) داخل سلول حافظه‌ای قرار می گیرد که P1 به آن اشاره دارد (متغیر a) . پس از اجرای سطر ۵ مقدار اشاره‌گر P2 داخل اشاره‌گر P1 قرار خواهد گرفت. پس، بعد از اجرای این دستور، P2 به هر جا که اشاره دارد P1 نیز به همان جا اشاره خواهد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

۷-۲-۲- اعمال محاسباتی جمع و تفریق

روی اشاره‌گرها می‌توان اعمال جمع و تفریق را انجام داد. اگر P یک اشاره‌گر 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;
```



۷-۳- 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.

اشاره‌گر = (نوع) malloc(size);

این تابع به اندازه خواسته شده سلول حافظه را جدا کرده و آدرس آن را برگشت می‌دهد. این آدرس برگشتی از نوع void می باشد، 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 اشاره‌گر P قرار می گیرد.

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

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

```
free (p);
```

۷-۴- اشاره‌گرها و توابع

مثال۱۶: 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 ۲۰: جواب مثال ۱۶

در مثا ل بالا برای تابع ()swap دو پرا متر با نام a,b از نوع اشاره‌گر 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 می باشد.

۷-۵- اشاره‌گرها و آرایه‌ها

در زبانC نام آرایه اشاره‌گری به اولین عنصر آرایه می باشد. پس با آرایه ها می‌توان همانند اشاره‌گرها و با اشاره‌گرها می توان همانند آرایه ها برخورد کرد، به عنوان مثا ل اگر اشاره‌گری با نام p را داشته با شیم:

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 می باشد.

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

در زبان C می توان عباراتی بر حسب ;"char *S="Computer را نوشت ،در این حا لت S* معادل تعریف آرایه با بعد نـا مـشخص می باشد. (char *S معادل[ ]char S)

مثا ل ۱۷: برنامه ای بنویسید که آدرس های تولید شده برای اشاره‌گرهایی از نـوع char وfloat وint وdouble را در اعمـال محاسـباتی 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 ۲۱: جواب مثال ۱۷

مثال ۱۸: 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 ("\n Error in allocate memory.");
    exit(1);
  }// end if
  b = (int * ) malloc (sizeof (int));
  if(!b)
  {
    printf ("Error in allocation memory.");
    exit(1);
  }//end if
  printf ("\n Enter two number:");
  scanf ("%d%d",a,b);
  c = *a**a + *b**b ;
  printf ("\n result = %d ", c);
  free(a);
  free(b);
  return 0;
}//end main
```

برنامه ۲۲: جواب مثال ۱۸

۷-۶- آرایه پویا

برای ایجاد آرایه به صورت پویا می‌توان از تابع ()malloc استفاده کرد بطور مثال اگر بخواهیم یک آرایه پویا بـه تعـداد ۱۰ خانـه 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 اشاره‌گریا آرایه‌ای استفاده کنیم ، به عنوان مثال اگر بخواهیم بـه خانـه ی ششم (اندیس 5) دسترسی داشته باشیم حالت اشاره‌گر برابر (P+5)* و حالت آرایه ای برابرP[5] خواهد بود ، نهایتاً با استفاده از تابع ()free می توان حافظه ی اختصاص داده شده را به سیستم برگرداند.

مثال ۱۹: برنامه ای بنویسید که شبیه تابع strcpy عمل کرده رشته ی دوم را داخل پارامتر اول copy کند با این تفاوت که تخـصیص حافظه باید به صورت پویا انجام شود (به تعداد مورد نیاز حافظه اختصاص داده شود).

```
#include<stdio.h>
#include<stdlib.h>
void strcopy(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);
  strcopy(str2 , str1);
  printf("\n Str2 = %s",str2);
  free(str2);
  return 0;
}
//************************************
void  strcopy(char *s1,char *s2)
{
  int i;
  for(i=0 ; s2 [i] ; i++)
    s1[i] = s2[i];
  s1[i] = '\0';
}
```

برنامه ۲۳: جواب مثال ۱۹

مثال ۲۰ : 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 وآدرس این آرایه را برگشت دهد.

۷-۷- ارزش دهی اولیه به اشاره‌گرها

برای ارزش دهی اولیه به اشاره‌گرها می توانیم به صورت زیر عمل کنیم در این مثال ها S آرایه ای بـا بعـد نامـشخص برابـر همـان اشاره‌گر 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 نا مشخص به توابع) باز می توان بجای آرایه ی با بعد نا مشخص از اشاره‌گر استفاده کرد.

```
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;
}
```

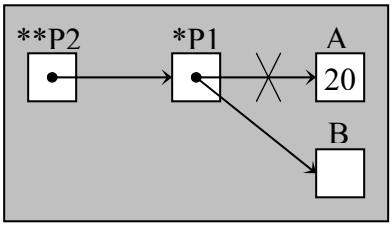
برنامه ۲۴: جواب مثال ۲۰

۷-۸- اشاره‌گر به اشاره‌گر

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

مثال زیر نمونه ای از تعریف این نوع اشاره‌گرها 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 ۲۱: مساله ی شبیه سازی تابع strcpy را به صورتی تغییر دهید که تخصص حافظه داخل خود تابع انجام شود .

هنگام استفاده از اشاره‌گرها 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 اشاره‌گرP به کدام محل از حافظـه اشـاره می کند پس هنگام قرار دادن عدد ۵ داخل P* ممکن است محلی از حا فظه که اجازه دسترسی به آن را نداریم تخریب شود.

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

۷-۹- آرگومان های تابع ()main

تا حالا توابع ()main هایی را که تعریف می کردیم بدون آرگومان ورودی بودند ، برای تابع ()main می‌توان آرگومان‌هـایی را قـرار 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 test 1 test 2

تایپ شده است،

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

```
#include<stdio.h>
#include<stdlib.h>
void strcopy(char  **, char *);
int main()
{
 char str1[50] , *str2;
 printf("\n Enter a string :");
 gets(str1);
 strcopy(&str2 , str1);
 printf("\n str2 = %s",str2);
 free(str2) ;
 return 0;
}
//*********************************
void strcopy( 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 ۲۵: جواب مثال ۲۱

توجه : حد اقل مقدار برای Argc برابر1 خواهد بود و همیشه دراندیس 0 ام Argv نام برنامه قرارمی گیرد.

مثال۲۲: 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 ۲۶: جواب مثال ۲۲

۸- ساختمان ها

۸-۱- 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 دادهای 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 ساختمنها در زبان C بصورت زیر می‌باشد.

```
Struct  <ساختمان> {
        بدنه ساختمان
};
```

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

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

این حالت فقط جنبه تعریف دارد و هیچ متغیری از آن ایجاد نمی‌شود (یعنی هیچ حافظه‌ای اختصاص داده نمی‌شود). 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 می‌شود، 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 ا ز 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 انتسا ب آنها را روی همدیگرکپی کنیم ، به عنوان مثال اگر بنویسیم P1=P2 باعث خواهد شد کلیه ی عناصر P2 داخل P1 کپی شوند.

مثال ۲۳: 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 ۲۷: جواب مثال ۲۳

۸-۲- مقداردهی اولیه ساختمان

برای مقدار دهی اولیه ی متغیرهای نوع ساختمان می توان همانند متغیرهای معمولی عمل کرد ، مقابل متغیر عناصر آنـرا بترتیـب قرار خواهیم داد در مثال زیر برای نام P1 رشته ی "Ali" بـرای fname رشـته ی "Kazemi" و بـرای P1.ID عـدد 10 قـرار داده مـی 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 سـاختمان بـا عـدد 0 مقـدار دهی شوند.

struct personel P2 = {0};

همانند متغیرهای معمولی نیز می توانیم برای ساختمانها آرایه تعریف کنیم به عنوان مثال در مورد ساختمان پرسنل اگر بخـواهیم اطلاعات 10 پرسنل را (آرایه ۱۰ عنضری) 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 ]
```

۸-۳- 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
```

همانند متغیرهای معمولی می‌توانیم اشاره‌گرهایی از نوع ساختمان تعریف کنیم، به عنوان مثال:

```
struct  student  *P
```

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

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

۸-۴- اختصاص حافظه به صورت پویا برای اشاره‌گر 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. مثا ل زیرنمونه ای از این حالت را نشان می دهد:

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

مثال ۲۴: برنامه ای بنویسید که اشاره‌گر 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 ۲۸: جواب مثال ۲۴

۸-۵- ساختمانهای بیتی

تا حالا میزان فضای حافظه ای که برای متغیرها اختصاص داده می شد حداقل انـدازه ی آن در حـد یـک بایـت بـود، بـرای اینکـه بتوانیم این اختصاص ها را به صورت بیتی انجام دهیم (یا در حقیقت بتوانیم به بیتهای یک بایت دسترسی پیدا کنـیم.) مـی تـوانیم 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 ، سـپس 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 یـک بیـت نمـی 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 می‌توان برای برخی از فیلدها که تنها یک بیت لازم دارند می‌توان از ساختمانهای بیتی استفاده کرد. بعنوان مثال :

۱-فارغ ا لتحصیل (شده یا نه؟) ۲- مشروطی (شده یا نه؟) ۳- تعهد(مجرد یا متا هل؟)

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 ;
};
```