اصول طراحی و برنامه نویسی به زبان C

الگوریتم و فلوچارت

برنامه ای که می نویسید به هر زبانی که باشد و برای هر که عملی باشد باید ابتدا یک الگوریتم برای آن طراحی کنید و این اولین گام در نوشتن برنامه است . واژه الگوریتم از نام دانشمند پر آوازه ایرانی ، خوارزمی گرفته شده است ، او اولین کسی بود که علم جبر را معرفی کرد . در واقع با الگوریتم شما به کامپیوتر (یا هر پردازنده دیگر) خواسته های خود را می فهمانید و به او می گویید که چه عملی (در شرایطی که آن را هم شما برای او تعریف می کنید) انجام دهد . در هر برنامه ای که می نویسید اولین گام شما درک مساله و فهمیدن کامل مساله است . بدون آگاهی از صورت مسئله و درک حدود و ابعاد آن، نمی توان به راه حل مطلوبی دست یافت. یکی از اساسی ترین عوامل شکست برنامه ها، عدم فهم و درک کامل آنها می باشد. پس از اینکه شما درک درستی از برنامه پیدا کردید و فهمیدید که می خواهید چه کاری را انجام دهید نوبت به پیاده سازی منطق برنامه می رسد . برنامه نویس باید راه حل مناسبی برای حل مساله پیدا کند و در این امر باید حوصله به خرج دهد .

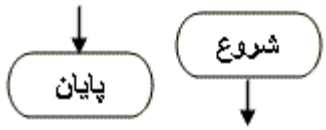
هنگامی که یک الگوریتم می نویسید دقت کنید که موارد زیر را حتما رعایت کنید :

- آغاز و پایان الگوریتم به طور دقیق مشخص باشد .
- مراحل دارای جزئیات کافی باشند .
- مراحل به زبانی دقیق نوشته شوند .
- مراحل به ترتیب و درست نوشته شوند .

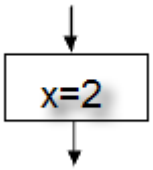
فلوچارت

در عمل برای نمایش الگوریتم از یک فلوچارت استفاده می شود . در حقیقت فلوچارت روش تصویری و استاندارد نمایش الگوریتم است . دررسم فلوچارت از علائم و نمادهای استانداردی استفاده می شود که هر کدام دارای معانی ویژه ای هستند .

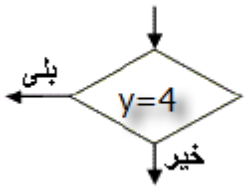
از بیضی برای شروع و پایان عملیات استفاده می شود .



از مستطیل برای نمایش دادن عملیات محاسباتی و مقداردهی استفاده می شود که دارای چند ورودی و فقط یک خروجی است .

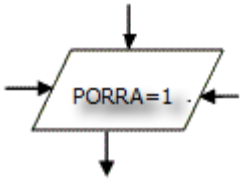


لوزی نیز جهت پرسیدن سئوال و ایجاد شرط استفاده می شود .

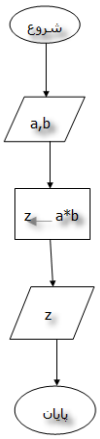


برای نشان دادن ورودی و خروجی نیز از متوازی الضلاع استفاده

می کنیم .

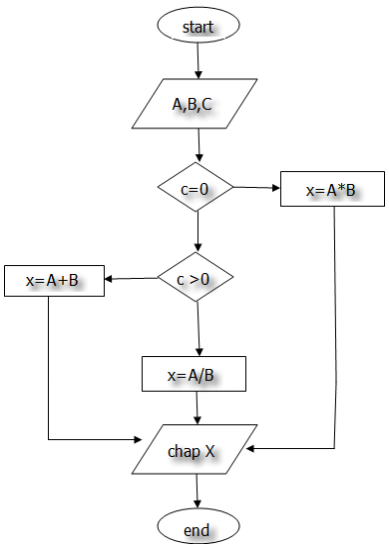


در مثال زیر می خواهیم که مقادیر دو متغییر A,B را در هم ضرب کرده و در متغییر Z بریزیم و Z را نمایش دهیم :



مثال :

می خواهیم دو عدد را از متغییر های A وB بگیریم . اگر C برابر صفر بود آنها را در هم ضرب کنیم ، اگر C بزگتر از صفر بود با هم جمع کنیم و اگر کوچکتر از صفر بود بر هم تقسیم کنیم :



زبان C

برای نوشتن برنامه به هر زبانی باید ابتدا قواعد و برخی از مفاهیم آن زبان را به خوبی بلد باشید و سپس برنامه نویسی را آغاز کنید . در این بخش ما به ارائه مفاهیم و قوائد زبان C می پردازیم و با ارائه مثال هایی به درک بهتر شما از این زبان کمک خواهیم کرد .

این مطالب و دستورات شاید در ابتدا زیاد به نظر برسند اما شما با نوشتن برنامه های مختلف و کارکردن با این دستورات با آنها آشناتر خواهید شد و این دستورات ملکه ذهن شما خواهند شد .

در ابتدا چند نکته ضروری که باید حتما از آنها اگاهی داشته باشید :

- ✓ در زبان C بین حرف کوچک و بزرگ تفاوت وجود دارد . به جز نام رجیسترها ، پورت ها و پین ها که با حروف بزرگ نوشته می شوند دستورات و دیگر اعمال با حروف کوچک نوشته می شوند .
- ✓ در انتهای هر دستور باید سمیکالان (" ; ") بگذاریم .
- ✓ توجه داشته باشید که علامت مساوی (=) سبب می شود که مقدار محتویات سمت راست در سمت چپ ریخته شود . مثلا در a=b ، محتویات b در a ریخته می شود .

کامپایلرهای C نسبت به خطوط خالی حساس نیستند و وقتی کد برنامه را می خوانند در جستجوی کاراکترهای جمله و ختم شدن به سمیکالان هستند و فاصله ها را ندیده خواهند گرفت(ثابت های رشته ای مستثنا هستند) مثلا جملات زیر با هم معادل هستند

```
x= 2+3 ;
x = 2    +    3    ;
x  =                                                    یا
2
+
3;
```

- ✓ در یک سطر می توان چندین دستور نوشت مثلا :

```
a = a+b ; c = 4 * a ;
```

- ✓ اگر در یک سطر و قبل از دستور ، عبارت " // " گذاشته شود آن دستور توسط CPU اجرا نمی شود و اگر می خواهید چندین سطر را غیر فعال کنید تا

اجرا نشوند در اولین سطر */ و در آخرین سطر که باید حذف شود/* بگذارید .

ساختار برنامه

هر برنامه ای که شما می نویسید باید دارای یک چهارچوب و ساختار همانند آنچه که در زیر آمده است باشد . رعایت این ترتیب و روال ضروری بوده و باید آن را رعایت کنید . هریک از این قسمت ها ابتدا به طور مختصر شرح داده شده و سپس به صورت مفصل توضیح داده خواهند شد .

```
#include <نام قطعه .h>                    معرفی کتابخانه و نوع قطعه
#include < کتابخانه های مورد نظر .h >
#define   rw   PORTC.1                    // معرفی ثابت ها
unsigned char a,b;                        // معرفی متغیرها
void get _char() {                        // توابع موردنظر
a = 0 ;                                   //دستورات .
void main (void){                         // تابع اصلی برنامه
a = b * 2 ;   }                           // برنامه اصلی
```

کتابخانه ها

بعد از اینکه نوع قطعه خود را در ابتدای برنامه مشخص کردید می بایست کتابخانه های لازم را فراخوانی کنید . باید توجه داشت که نحوه فراخوانی کتابخانه باید به صورتی که در صفحه قبل گفتیم باشد . کتابخانه ها در واقع برنامه هایی از قبل نوشته شده هستند که توسط طراحان کامپایلر نوشته شده است تا کار برای برنامه نویسان و کاربران راحت

شود البته شما هم به نحوی که بعدا گفته خواهد شد می توانید کتابخانه مورد نظر خود را بسازید و آن را فراخوانی کنید .

ثابت ها

توسط ثابت ها می توان یک شبه تابع ایجاد کرد یا اینکه نامی را برای پورت یا پینی گذاشت که به جای آن پورت یا پین از نام مستعار آن استفاده کرد . به مثال های زیر توجه کنید :

```
#define  enable  PORTC.1 ;

#define  x  12 ;

#define    or(a,b)   a|b ;

x=or(0x10,0x65);
```

در مثال یک ، پین یک پورت C را enable معرفی کرده ایم و می توانیم در خلال برنامه به جای PORTC.1 از کلمه enable استفاده کنیم .

در مثال دوم x را برابر 12 قرار داده ایم و حرف x در خلال برنامه به عنوان عدد 12 شناخته می شود .

در مثال سوم برنامه 0x10 را در a و0x65 را در b می ریزد و سپس آنها را به صورت بیت به بیت OR کرده و حاصل را در x می ریزد .

توجه کنید که ثابت را همانگونه که از نامش پیداست نمی توان در خلال برنامه تغییر داد مثلا اگر x را 12 درنظر گرفته ایم در برنامه نمی توانیم x=13 قرار دهیم .

متغیر ها

نام هایی هستند که ما برای خانه های حافظه در نظر می گیریم تا دسترسی به انها راحتر باشد . این نام ها می توانند با حروف بزرگ یا کوچک باشند اما نمی توان از نام های کلیدی استفاده کرد و همچنین نباید در ابتدای نام مورد نظر عدد باشد اما می توان در انتهای نام از عدد استفاده کرد . متغیرها انواعی دارند که لیستی از آنها در جدول ()آمده است :

نوع متغیر	تعداد بیت	محدوده
bit	1	0 یا 1
Char	8	127تا -128
Int یا short int	16	32767 تا -32768
Float یا double	32	$3.37*10^{38}$ تا $-8.43*10^{-38}$

جدول1-2 نوع متغیرها

نکته : توجه شود که اگر متغیر شما از نوع عدد اعشاری است باید آن را از نوع float تعریف کرد .

متغیرها را می توان در ابتدای برنامه به طور سراسری تعریف کرد تا در کل برنامه شناخته شود یا آن را می توان به صورت محلی تعریف کرد (مثلا در یک تابع) که در این صورت فقط در آن تابع قابل استفاده می باشد . نحوه معرفی یک متغیر :

نوع متغییر نام دلخواه ;

نوع متغییر نام دلخواه = مقدار ;

مثال :

```
char  a , b =10  ;
```

نکته : اگر در ابتدای نام نوع متغیر کلمه کلیدی unsigned (به معنای بی علامت) به کار ببریم محدوده منفی حذف و به محدوده مثبت اضافه می شود مثلا char محدوده 128- تا 127 دارد اما unsigned char محدوده 0 تا 255 را شامل می شود .

نکته : اگر در ابتدای نوع متغیر کلمه کلیدی flash یا const بنویسیم متغیر تبدیل به یک ثابت می گردد و دیگر در خلال برنامه قابل تغییر نمی باشد مثلا :

```
const char x=10;
```

دراین صورت دیگر نمی توان در طول برنامه مقدار x را برابر 4 یا هر عدد دیگری قرارداد .

فرمت متغیرها

- **دسیمال** : اگر متغیر ما دارای مقدار بود و قبل از عدد چیزی ننویسیم عدد نظر به عنوان عدد دسیمال شناخته می شود مانند: unsigned char a=13;
- **هگز** : اگر قبل از عدد حرف 0x قرار دهیم به عنوان عدد هگز شناخته می شود .

```
unsigned char a=0xf1;
```

- **کد اسکی** : در صورتی که یک کاراکتر (حرف یا عدد) داخل کوتیشن (‘ ’) قرار گیرد کد اسکی معادل ان کاراکتر در متغیر ریخته می شود .

```
unsigned char a='w';
```

- **آرایه :** اختصاص چند بایت از حافظه برای یک گروه متغیر را آرایه می گویند . آرایه می تواند شامل چند عدد یا رشته شود یا اینکه دارای یک یا دو بعد باشد . آرایه یک بعدی به صورت زیر تعریف می شوند :

نام دلخواه[تعداد اعضای آرایه] = {اعضا};

مثال:

```
unsigned char a[3]={3,4,6} ;

x = a[1] ;
```

در مثال فوق عضو یکم متغیر آرایه a (عدد 4) را در x می ریزد. در نظر داشته باشید که شماره اعضا از صفر شروع می شود . اگر بخواهیم در آرایه چیزی قرار دهیم به صورت زیر عمل می کنیم :

```
unsigned char    a [ ] ;

a [3] = ‘ r ’ ;
```

که در اینصورت عضو شماره سه آرایه a کد اسکی حرف r خواهد بود .

آرایه دو بعدی نیز به صورت زیر تعریف می شوند :

```
نام دلخواه [اندازه بعد یک] [ اندازه بعد دوم ] ;
```

مثال :

```
unsigned char a [2] [4]={ {1,2,3,4} , {5,6,7,8} } ;

x= a[1] [1];
```

در این مثال عدد 6 در متغیر a ریخته می شود .

عملگرها

عملگرها نمادهایی هستند که برای انجام اعمال خاصی مورداستفاده قرار می گیرند. عملگرها در زبان C از تنوع زیادی برخوردارند . در زبان C عملگرها شامل عملگرهای محاسباتی ، مقایسه ای ، منطقی ، بیتی و منطقی و انتسابی یا ترکیبی تقسیم می شوند که آنها را بررسی می کنیم .

تقدم در عملگرها

تقدم در اجرای عملگرها امر مهمی است که باید به آن توجه داشته باشید . تقدم بدین معنی است که اگر در یک خط برنامه چند عملگر وجود داشت ، پردازنده ابتدا کدام را انجام می دهد ، مثلا اگر یک برنامه به صورت x=3*2%4+1 وجود داشته باشد جواب چه عددی خواهد بود ؟ با توجه به اینکه شما ابتدا کدام عمل را انجام دهید جواب های مختلفی بدست خواهد آمد . به همین جهت میکرو برای انجام اعمال عملگرها وسایر دستورات ، برای آنها اولویت قرار میدهد . در مثال بالا ابتدا عدد 3 در 2 ضرب می شود و حاصل بر 4 تقسیم می شود سپس عدد بدست آمده با یک جمع می شود . در اینجا تقدم هر عملگر را به همراه عملی که انجام می دهد را در جداولی گرد آورده ایم . اگر در یک خط برنامه دو یا چند عملگر با تقدم یکسان وجود داشته باشد ، عملگری تقدم دارد که در سمت چپ باشد .

توجه داشته باشید که عمل داخل پرانتز دارای بالاترین اولویت می باشد .

- **عملگرهای محاسباتی**

این عملگر ها ، محاسبات و اعمال ریاضی را برای ما انجام می دهند .

لیستی از این عملگرها و تقدم آنها در زیر آمده است :

عملگر	عملکرد	مثال	نتیجه	تقدم
++	یک واحد افزایش	a++	a با یک جمع می شود	**اولویت اول**
--	یک واحد کاهش	a--	از a یکی کم می شود	**اولویت اول**
-	قرینه کردن	-a	a در یک منفی ضرب می شود	**اولویت دوم**
*	ضرب کردن	a * b	a در b ضرب می شود	**اولویت سوم**
/	خارج قسمت تقسیم را به ما می دهد	10/3	3	**اولویت سوم**
%	باقیمانده تقسیم را می دهد	10%3	1	**اولویت سوم**
-	تفریق	10-4	6	**اولویت چهارم**
+	جمع کردن	10+2	12	**اولویت چهارم**

جدول2-2 عملگرهای محاسباتی

• عملگرهای مقایسه ای

عملگرهای مقایسه ای همانطور که از نامشان پیداست جهت مقایسه به کار می روند . کاربرد این عملگرها بیشتر در ایجاد شرط برای انجام اعمالی خاص می باشد .

عملگر	عملکرد	تقدم
<	کوچکتر	**اولویت اول**
>	بزرگتر	**اولویت اول**
<=	کوچکتر یا مساوی	**اولویت اول**
>=	بزرگتر یا مساوی	**اولویت اول**
==	مقایسه برابری	**اولویت دوم**
!=	مقایسه عدم برابری	**اولویت دوم**

جدول3-2 عملگرهای مقایسه ای

مثال :

می خواهیم به ماشین بفهمانیم که اگر a بزرگتر یا مساوی عدد دو بود مقدار a را یک واحد افزایش دهد . (در ادامه دستورات شرطی را توضیح خواهیم داد)

```
if (a>=2) {  a++  ;  }
```

یا اینکه می خواهیم به میکرو دستور بدهیم که اگر a هر عددی به غیر از دو بود a را یک واحد افزایش دهد .

```
if (a!= 2) { a++; }
```

• عملگرهای منطقی

برای انجام اعمال منطقی بر روی داده ها استفاده می شوند .

عملگر	عملکرد	تقدم
!	عمل NOT	**اولویت اول**
&&	به عنوان 'و'	**اولویت اول**
||	به عنوان 'یا'	**اولویت اول**

جدول4-2 عملگرهای منطقی

مثال : می خواهیم اگر a بزرگتر از یک و b کوچکتر از دو بود مقدار a برابر صفر شود .

```
if (a>1&&b<2) {a=0;}
```

- **عملگرهای انتسابی**

عملگر	عملکرد	مثال	نتیجه	تقدم
~	مکمل یک	~(0xf0)	0x0f	**اولویت اول**
>>	شیفت به راست	0b00001100>>2	0b00000011	**اولویت دوم**
<<	شیفت به چپ	0x00000011<<2	0b00001100	**اولویت دوم**
&	AND بیتی	1000&1111	1000	**اولویت سوم**
^	XOR بیتی	1001&1010	0011	**اولویت چهارم**
|	OR بیتی	1001|1010	1011	**اولویت پنجم**

جدول5-2 عملگرهای انتسابی

اگر بخواهید در نوشتن عملگر ها خلاصه نویسی کنید می توانید عمل مورد نظر را قبل از علامت مساوی بنویسد به مثال های زیر توجه کنید :

a=a&b ⟶ a&=b

a=a>>3 ⟶ a>>=3

a=a+b ⟶ a+=b

- **عملگر ( ) sizeof**

تعداد بایت های مورد استفاده هرمتغیر یا نوع داده ای را می توان توسط این عملگر بدست آورد .

مثال :

```
unsigned char   a , b ;
b= sizeof (a) ;
```

در اینجا چون متغییر ما از نوع unsigned char می باشد طول متغیر ما یک بایت است و مقدار b برابر یک می شود .

- **عملگر * و &**

عملگر * اشاره به محتوای آدرس یک متغیر و عملگر & اشاره به آدرس یک متغییر دارد مثلا b = &a یعنی آدرس متغییر a در حافظه را در b ریخته شود و x =*y یعنی به آدرس y در حافظه برو و محتوای آن را در x کپی کن . به فرض اینکه اگر y=0x80 باشد ، برنامه به آدرس 0x80 حافظه رفته و محتویات آن را در x کپی می کند .

- **عملگر کاما (,)**

از این عملگر به عنوان جدا کننده برای جداسازی متغییر ها در هنگام تعریف آنها و هچنین جداسازی عمل ها و دستورات استفاده می شود . اگر از این عملگر برای جداسازی دستور استفاده می کنید باید توجه داشته باشید که ابتدا دستور سمت چپ اجرا می شود . مثلا :

```
unsigned char  x , y ;
z=( x+2 ,  y++ ) ;
z=( x=2 ; y/x )  ;
```

در خط دوم برنامه ابتدا x با دو جمع می شود سپس متغیر y یک واحد افزایش پیدا می کند و در z ریخته می شود . در خط سوم ابتدا مقدار x برابر دو می شود و سپس y بر x تقسیم شده و خارج قسمت در z ریخته می شود .

دستورها

به کمک دستورها ما تغیرات مورد نظرمان را بر روی متغیرها در شرایط دلخواهی که می توانیم خودمان آن را تعیین کنیم اعمال کنیم . توجه داشته باشید که دستورا را با "{" آغاز و با "}" به پایان برسانید که در غیر اینصورت با Error مواجه خواهید شد .

دستور شرطی if

if در زبان انگلیسی به معنای "اگر" می باشد . در واقع شما با این دستور یک شرط را تعیین می کنید که اگر ارضا شود دستورات اجرا خواهند شد فرم کلی این شرط به صورت زیر است :

```
if(شرط)  {
دستورات مورد نظر ; }
```

دستور شرطی if _ else

توسط این دستور در قسمت if یک شرط را تعیین می کنیم که اگر برآورده شود دستوراتی انجام می شوند و در غیر اینصورت (else) دستورات دیگری انجام می شود .
توجه کنید که دستورات else فقط در صورت عدم برقراری شرط if انجام خواهند شد .
فرم کلی این دستور به صورت زیر می باشد :

```
if (شرط)  {
دستور اول ; }
else {
```

{ ; دستور دوم

مثال :

```
if (a==10) {                    // اگر a برابر ده بود
a=0;                            // a را صفر کن
b++;  }                         // b را یک واحد افزایش بده
else  {                         // در صورتی که a برابر ده نبود
a++;                            // a را یک واحد افزایش بده
b=0;  }                         // b را صفر کن
```

دستور حلقه for

این حلقه شامل یک مقدار اولیه ، یک شرط برای تداوم حلقه و یک عمل است که پس از انجام یک دور حلقه ، انجام می شود و پس از آن شرط چک می شود و در صورت برقراری شرط حلقه دوباره تکرار می شود ، این عمل می تواند شمارشگر حلقه باشد تا حلقه به تعداد دلخواه انجام شود . توجه کنید که در اولین بار فقط شرط چک می شود و در صورت برقراری ، دستورات درون حلقه اجرا می شوند و هیچگونه تغییری در عمل حلقه یا همان شمارشگر حلقه ایجاد نمی شود . پس در این صورت شماره حلقه از صفر شروع می شود .

for(شمارنده حلقه; شرط حلقه ; مقدار اولیه)

{ عملی که پس از هر دور انجام می شود}

مثال :

```
for ( x=0 ; x<=4 ; x++ )  {
a++;
}
```

در مثال فوق متغیر a پنج بار به میزان یک واحد افزایش می یابد زیرا شروع شمارش از صفر شروع می شود و پس از پنج بار انجام دستور ، CPU از حلقه خارج می شود .

نکته :

حلقه های for را می توان به صورت تو در تو قرار داد . نحوه کار حلقه های تو در تو را با مثال زیر توضیح خواهیم داد .

```
Unsigned char a=0, b=0, x, y ;

for(x=0 ; x<4 ; x++)  {
a++;
for (y=0 ; y<4 ; y++)  {
b++ ;
}}
```

در مثال فوق ابتدا برنامه وارد حلقه اول می شود و مقدار متغیر a را یک واحد افزایش داده و سپس وارد حلقه دوم می شود و متغیر a را چهار بار و هر بار به اندازه یک واحد افزایش می دهد سپس برنامه از حلقه دوم خارج شده و x را یک واحد افزایش داده و شرط را چک می کند در صورت برقراری شرط (x<4) سیکل بالا دوباره تکرار می شود . درنهایت هنگامی که برنامه از حلقه تو در تو خارج شود مقدار متغیر a برابر چهار بوده و مقدار متغیر b برابر شانزده می باشد .

حلقه while

While در انگلیسی به معنای زمان و هنگام می باشد . توسط این دستور تا هنگامی که شرط برقرار باشد حلقه تکرار می شود . ساختار این حلقه به صورت زیر می باشد :

```
while (شرط)  {
 دستورات ; }
```

مثال :

```
while (a < 10)  {
a++ ; }
```

در مثال فوق حلقه ده بار اجرا می شود و نهایتا مقدار a برابر ده می شود و برنامه از حلقه خارج می شود .

نکته :

توسط while می توان یک حلقه ایجاد کرد که بی نهایت بار تکرار شود . فرم چنین حلقه ای به صورت زیر می باشد :

```
while (1) {
 دستورات ; }
```

دستور حلقه شرطی do -while

نحوه کار این دستور همانند دستور while می باشد با این تفاوت که ابتدا حلقه یکبار انجام می شود سپس شرط چک می شود در صورتی که در دستور while ابتدا شرط چک می شد و سپس حلقه اجرا می شد . فرم این دستور به صورت زیر است :

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

مثال :

```
Unsigned char a=1;
do {
a++;
 }  while (a==0) ;
```

در این مثال با وجود عدم برقراری شرط برنامه وارد حلقه شده و مقدار a را یک واحد افزایش می دهد وسپس از حلقه خارج می شود .

دستور break

Break به معنای شکستن بوده و برای شکستن و خارج شدن بی قید و شرط از حلقه می باشد . هر گاه در حلقه ، برنامه به این دستور برسد از حلقه خارج شد ه و به اولین دستور بعد از حلقه می رود .

مثال :

```
while (1)  {
a++ ;
c=b-a;
if (a==1 ) {  break ;  }

}
```

دستور goto

با رسیدن CPU به این دستور ، CPU بدون قید شرط به Label مشخص شده پرش می کند . فرم این دستور به صورت زیر می باشد :

```
goto     موردنظر label  ;
.
.
مورد نظر label  :
دستورات  ;
```

دستور continue

هر گاه در حلقه while برنامه به این دستور دستور برسد ، به ابتدای حلقه رفته و مجددا شرط را چک می نماید و در صورت برقراری شرط حلقه را دوباره از ابتدا اجرا می کند .

مثال :

```
unsigned char a,b ;
while (a<100)  {
a+=b ;
if (b==13)
continue ;
else
a++;}
```

در این مثال هرگاه b برابر سیزده شود برنامه شرط را یک بار چک می کند و در صورت درستی شرط برنامه را از نوع اجرا کرده و در غیر اینصورت از حلقه خارج می شود .

دستور switch _ case

با این دستور بر روی case های مختلف switch می کنیم . به عبارت دیگر ما بررسی می کنیم که متغیر مورد نظرمان (که باید یک مقدار ثابت باشد) با کدام یک از case ها برابر است و در صورت برابری ، برنامه مقابل آن case را اجرا می کنیم . قابل ذکر است که می توان برای جلوگیری از تلف شدن وقت CPU وجلوگیری از چک کردن سایر case ها ، در آخر هر دستور از break استفاده شود . در این صورت با switch کردن بر روی case مورد نظر پس از اجرای دستورات ، برنامه از ساختار switch خارج خواهد شد. همچنین می توان از دستور default استفاده کرد تا زمانی که مقدار switch با هیچ یک از case ها برابر نبود دستور خاصی را انجام دهد . فرم دستور switch_case به صورت زیر می باشد :

```
switch (متغیر مورد نظر ما)
{
case   مقدار اول  :  دستور اول ;  break ;
case   مقدار دوم  :  دستور دوم ;  break ;
.
.
case  مقدار n ام  :  دستور n ام ;  break ;
default :   دستور   ;
```

}

مثال :

```
unsigned char  a , b ;
switch (a)          {
case  13  b=b/a ; break ;
case   2   b=b*a ; break ;
default  : b=0 ;     }
```

در این مثال مقدار a اگر برابر 13 باشد خارج قسمت تقسیم b بر a در b ریخته خواهد شد . اگر a برابر دو باشد b در a ضرب شده و در b ریخته می شود و در صورتی که a با هیچکدام از case ها برابر نباشد مقدار b صفر می شود .

دستور typedef

از این دستور برای تغییر نام نوع متغیر به نام دلخواه استفاده می شود .

typedef نام قدیمی نوع متغیر نام جدید نوع متغیر ;

مثال :

```
typedef  unsigned char  dade  ;
```

در اینجا دیگر به جای unsigned char می توان واژه dade را به کار برد .

ساختارها (structure)

برای تعریف گروهی از متغیرها که در نام و متغیرشان وجه اشتراک دارند از ساختار استفاده می کنیم . در واقع در این روش دیگر لازم نیست برای هر متغیر مورد استفاده،

آن را معرفی کرده و نوع متغیر را تعریف کرد . برای دسترسی به این اعضا از نقطه "." استفاده می کنیم شکل ساختار به صورت زیر می باشد .

```
Struct  نام دلخواه  {
اعضای ساختار    ;    }
 نام ساختار      نام گروه   ;
 نام گروه          نام متغیر  ;
```

مثال :

```
struct   class {
unsigned char    nam [10] , micro1 , micro 2 ;
}
class group1 [50] ;
class group2 [30] ;
group1 [5] . name[ ] =" ali " ;
group 1 [5]. micro1=13 ;
group2 [10] . name[ ] =" reza " ;
group 2 [10]. Micro2=16  ;
```

در این مثال پس از تعیین ساختار، گروه های group1 و group2 را ایجاد کردیم سپس عضو پنجم group1 را ali نامیده و نمره میکرو آن را 13 قرار دادیم . همچنین عضو دهم group2 را reza نامیده و نمره میکرو او را 16 در نظر گرفته ایم . همچنین می توان مطابق مثال زیر برای structure ها طول بیتی تعریف کرد . مثال :

```
unsigned char   x:2  ;
unsigned char   y:3  ;
unsigned char   x:5  ;
```

که طول بیتی هر متغیر در برابر آن نوشته شده است.

اتحادها (یونیون ها)

کارکرد اتحادها نیز همچون ساختارها می باشد با این تفاوت که اعضای گروه از مکان حافظه اتحاد به طور مشترک استفاده می کنند و شکل کلی آن به صورت زیر می باشد :

```
union      نام اتحاد {
اسامی اعضا ;
اسامی متغییرها ; }
```

شمارش (enum)

از enum ها جهت شماره گذاری تعدادی متغیر استفاده می شود برای درک این موضوع به مثال زیر توجه کنید :

```
enum (x , y , z ) ;
```

در این مثال به x عدد صفر ، به y عدد یک و به z عدد دو داده می شود و اگر تعداد متغیر های بیشتری داشته باشیم به همین ترتیب شماره گذاری می شوند . توجه کنید که اگر خودمان به متغیری مقدار عددی بدهیم برنامه شروع شمارش را از آن عدد شروع می کند . مثلا اگر به x عدد 13 را می دادیم شمارش از 13 شروع می شد .

توابع کتابخانه ای

در این قسمت به معرفی کتابخانه های زبان C که در کامپایلر Code Vision نیز دارای این کتابخانه ها می باشد خواهیم پرداخت .

- **کتابخانه math.h**

از این کتابخانه همانطور که از نامش پیداست برای انجام محاسبات توسط توابع ریاضی خاص همچون توابع مثلثاتی استفاده می شود . در جدول زیر توابعی از این کتابخانه که

باید برای محاسبه توابع مثلثاتی استفاده شود را آورده ایم . همانطور که قبلا اشاره شد در جاهایی که شما می خواهید از یک عدد اعشاری استفاده کنید یا اینکه نتیجه محاسبات شما یک عدد اعشاری است باید نوع متغیرتان را float انتخاب کنید از این رو چون خروجی توابع مثلثاتی اعشاری است باید متغیرتان را اعشاری تعریف کنید .

نکته :خروجی توابع مثلثاتی کتابخانه math.h بر حسب رادیان می باشد .

عملیات ریاضی	تابع لازم
sin x	float sin(float x)
cos x	float cos(float x)
tan x	float tan(float x)
arc sin x	float asin(float x)
arc coc x	float acos(float x)
arc tan x	float atan(float x)

جدول6-2 توابع کتابخانه math.h

مثال :

```
#include <math.h>
float  a , b ;
a = sin (b) ;
```

سایر توابع که غیر مثلثاتی می باشند به صورت زیر هستند :

عملیات ریاضی	تابع لازم
$\log x$	float log10(float x)
e^x	float exp(float x)
ln(x)	float log(float x)
a^b	float pow(float a , float b)
$)\sqrt{x}$	float sqrt(float x)

جدول7-2 توابع کتابخانه math.h

مثال :

```
#include <math.h>
float  a , b ;
```

```
a = log10 (b) ;
```

و چند تابع دیگر از این کتابخانه :

Char max (char a , char b)

این تابع دو متغیر را می گیرد و مقدار بزرگتر را بر می گرداند . متغیر ها می توانند از هر نوع دیگر (int یا unsigned char) نیز باشند .

Char min (char a , char b)

این تابع دو متغیر را می گیرد و مقدار کوچکتر را بر می گرداند . متغیر ها می توانند از هر نوع دیگر (int یا unsigned char) نیز باشند .

مثال :

```
#include<math.h>
char   a, b , c ;
c=max (a,b) ;
```

- **کتابخانه delay.h**

این کتابخانه یکی از پرکاربردترین کتابخانه ها می باشد به طوری در بیشتر برنامه ها از آن برای ایجاد تاخیر زمانی استفاده می کنیم . تاخیرهایی که می خواهیم ایجاد کنیم باید بر حسب میلی ثانیه یا میکرو ثانیه باشد و فرم نوشتن آن به صورت زیر است :

```
delay_ms(زمان تاخیر بر حسب میلی ثانیه)  ;
```

```
delay_us(زمان تاخیر بر حسب میکرو ثانیه)  ;
```

مثلا برای تاخیر 10 ms دستوری به صورت delay_ms(10) را می نویسیم .

نکته : در صورت استفاده از تابع delay_ms() اگر watch dog timer فعال باشد هر یک میلی ثانیه یک بار مقدار این تایمر را Reset میکند . نحوه استفاده از این تایمر را فصل های بعدی توضیح خواهیم داد .

کتابخانه bcd.h

از این کتابخانه برای تبدیل اعداد و کدها استفاده می شود .

unsigned char bcd2bin(unsigned char n)

از این تابع برای تبدیل کد BCD به کد باینری استفاده می شود به طوری که متغیر n را که BCD است را به کد باینری تبدیل کرده و در متغیر مقصد می ریزد .

unsigned char bin2bcd(unsigned char n)

از این تابع برای تبدیل کد باینری به کد BCD استفاده می شود، بر عکس حالت قبل.

- **کتابخانه stdlib**

این کتابخانه برای تبدیل اعداد در نوع های مختلف به رشته و عکس آن استفاده می شود کاربرد این کتابخانه بیشتر در نمایش اعداد بر روی lcd است که در فصل های بعد توضیح خواهیم داد . توجه کنید که اگر می خواهید عددی را به رشته تبدیل کنید متغیر مقصد باید به صورت آرایه ای تعریف شود همچنین اگر می خواهید یک رشته را به عدد تبدیل کنید و عدد از نوع اعشاری است فراموش نکنید که متغیر مقصد را از نوع float تعریف کنید . توابع این کتابخانه به در زیر آمده است :

int atoi(char *str)

از این تابع جهت تبدیل رشته str* به عدد استفاده می شود .

long int atol(char *str)

از این تابع جهت تبدیل رشته به عدد بلند استفاده می شود .

Void itoa (int n , char *str)

از این تابع برای تبدیل عدد به رشته استفاده می شود ، بعنوان مثال:

```
int   a = 100 ;
char b[ ] ;
itoa (a , b );
```

در این مثال عدد 100 تبدیل به رشته شده و در متغیر b ریخته می شود .

void ltoa(long int n,char *str) : عملکرد این تابع شبیه تابع قبل بوده با این تفاوت که عدد مورد نظر از نوع بلند long int می باشد .

void ftoa (float n ,unsigned char decimals ,char *str)

از این تابع جهت تبدیل یک عدد اعشاری به رشته استفاده می شود به این صورت که عدد اعشاری n را گرفته و با دقتی که توسط یک عدد دسیمال مشخص می شود تبدیل به رشته کرده و در رشته str می ریزد ، بعنوان مثال:

```
float    a = 2.14   ;
char     b= [10]  ;
ftoa (a , 2 , b);
```

در این مثال عدد اعشاری 2.14 با دقت دو رقم اعشار به رشته تبدیل شده و در متغیر b ریخته می شود .

float atof(char *str)

از این تابع برای تبدیل یک رشته به عدد اعشاری تبدیل می شود .

int rand (void)

بسته به نوع متغیر مقصد از صفر شروع به شمارش کرده و یک عدد را به صورت تصادفی در متغییر مقصد می ریزد . مثلا اگر متغیر مقصد شما از نوع int باشد عددی بین 0 تا 32767 انتخاب می شود .

void srand(int seed)

مانند تابع rand عمل می کند با این تفاوت که عدد شروع شمارش (x) رابه او می دهیم .

کتابخانه ctype .h

unsigned char isalnum(char c)

اگر ورودی c حروف الفبا یا رقم باشد متغیر مقصد یک و در غیر اینصورت صفر می باشد.

unsigned char isalpha(char c)

اگر c حروف الفبا لاتین باشد متغیر مقصد یک و در غیر اینصورت صفر خواهد بود .

unsigned char isascii(char c)

اگر c کد اسکی اعداد0 تا 127باشد متغیر مقصد یک و در غیر اینصورت صفر خواهد بود.

unsigned char iscntrl(char c)

اگر c عددی در محدوه0تا 127باشد متغیر مقصد یک و در غیر اینصورت صفرخواهد بود.

unsigned char isdigit(char c)

اگر c یکی از اعداد دسیمال0تا9باشد متغیر مقصد یک و در غیر اینصورت صفرخواهد بود.

unsigned char islower(char c)

اگر c یکی از حروف الفبای کوچک لاتین باشد متغیر مقصد یک و در غیر اینصورت صفر خواهد بود .

unsigned char isprint(char c)

اگر c یک کاراکتر قابل چاپ در محدوه 32 تا 127 باشد متغیر مقصد یک و در غیر اینصورت صفر خواهد بود .

unsigned char ispunct(char c)

اگر c یک کاراکتر علامت باشد متغیر مقصد یک و در غیر اینصورت صفر خواهد بود .

unsigned char isspace(char c)

اگر c یک کاراکتر فضای خالی باشد متغیر مقصد یک و در غیر اینصورت صفر خواهد بود.

unsigned char isupper(char c)

اگر c یکی از حروف بزرگ الفبا لاتین باشد متغیر مقصد یک و در غیر اینصورت صفر خواهد بود .

unsigned char isxdigit(char c)

اگر c یک عدد هگزا دسیمال باشد متغیر مقصد یک و در غیر اینصورت صفر خواهد بود .

unsigned char toint(char c)

این تابع مقدار c را بر حسب هگزا دسیمال دریافت کرده و معادل دسیمال آن را در متغیر مقصد می ریزد .

char tolower(char c)

اگر c حرف بزرگ الفبا لاتین باشد آن را به حرف کوچک تبدیل کرده و در متغیر مقصد

می ریزد و اگر c حرف کوچک باشد همان را در متغیر می ریزد .

char toupper(char c)

این تابع عکس تابع قبلی عمل می کند .

کتابخانه string .h

از این کتابخانه جهت کار با رشته ها استفاده می شود . در اینجا ما فقط چند تابع که ممکن است مورد استفاده قرار گیرد را شرح می دهیم .

char *strcat(char *str1,char *str2)

از این تابع جهت پیوند دو رشته به هم استفاده می شود به طوری که رشته str2 را به انتهای رشته str1 متصل می کند .

char *strcatf(char *str1,char flash *str2)

این تابع رشته str2 را که در حافظه fash یا ثابت ذخیره شده است را به انتهای رشته str1 متصل می کند .

char *strncat(char *str1,char *str2,unsigned char n)

این تابع n کاراکتر از رشته str2 را به انتهای رشته str1 متصل می کند .

char *strncatf(char *str1,char flash *str2,unsigned char n)

این تابع n کاراکتر از رشته str2 را که در حافظه fash یا ثابت ذخیره شده است به انتهای رشته str1 متصل می کند .