AVR

برنامه نویسی میکرو کنترلر های AVR **به زبان**

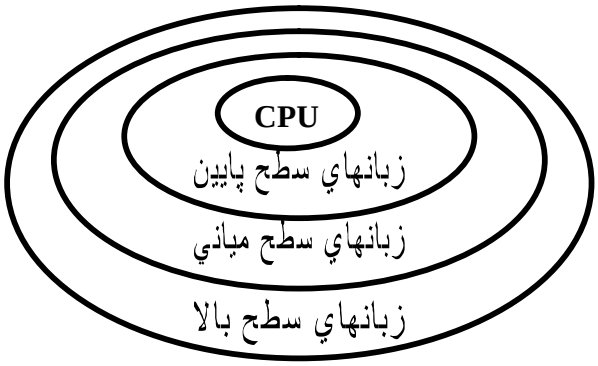
توسط نرم افزار CodeVision

مقدمه :

زبان C یک زبان برنامه نویسی ساده و پر قدرت میباشد که در سال 1972 توسط دنیس ریچی طراحی شده است.

زبان های برنامه نویسی به سه دسته زیر تقسیم میشوند :

- زبانهای سطح بالا :دستورالعملهای این زبانهای برنامه نویسی به زبان محاوره ای انسان نزدیک است ولی دسترسی مستقیم به حافظه ، ثباتهای پردازشگرو... ندارند ، مانند زبانهای BASIC ، PASCAL و...
- زبانهای سطح پایین : بوسیله این زبانهای برنامه نویسی زبان اسمبلی
- زبانهای سطح میانی : زبان C



زبانهای برنامه نویسی به دو دسته ساخت یافته و غیر ساخت یافته نیز تقسیم بندی میشوند. در زبان برنامه نویسی ساخت یافته با استفاده از حلقه های تکرار(for ، while ، do...while) میتوان برنامه ای نوشت که قابلیت خوانایی بالائی داشته باشند.

خصوصیات زبان C :

- یک زبان برنامه نویسی میانی است.
- یک زبان برنامه نویسی ساخت یافته است.
- یک زبان قابل انعطاف و قدرتمند است.
- ارتباط تنگاتنگی بین زبان C و اسمبلی وجود دارد.
- تعداد کلمات کلیدی این برنامه کم (30 کلمه کلیدی) میباشد.

- زبان برنامه نویسی C نسبت به حروف کوچک و بزرگ حساس است . تمام کلمات کلیدی آن با حروف کوچک نوشته میشوند. برای مثال int یک کلمه کلیدی است اما INT کلمه کلیدی نیست.

در شکل زیر کلمات کلیدی زبان C نمایش داده شده است:

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

دستورالعمل های زبان C دارای خصوصیات زیر میباشد.

- هر دستور زبان C به ; ختم میشود. مثال :

```
int a=1;
float pi=3.14;
```

- حداکثر طول یک دستور 255 کاراکتر است.
- هر دستور میتواند در یک یا چند سطر ادامه داشته باشد.
- در هر سطر میتوان چند دستور تایپ کرد.

توضیحات :

در یک برنامه نوشته شده با اضافه کردن توضیحات خوانائی و درک برنامه بالا میرود . توضیحات توسط مترجم (کامپایلر) نادیده گرفته میشود. در زبان C توضیحات میتوانند در بین /* و */ قرار بگیرند یا بعد از // نوشته شوند.

```
مثال 1:   /* This is a sample for command
                for C program language. */
مثال2:   // This lne is a sample.
```

ساختار برنامه C:

برنامه های زبان C از مجموعه ای از دستورات و تعدادی تابع تشکیل میشود. بدنه اصلی برنامه زبان C ، تابع main() می باشد. هر برنامه باید دارای این تابع باشد. در شکل زیر ساختار یک برنامه C نمایش داده شده است که در آینده این ساختار با معرفی قسمتهای دیگر زبان C تکمیل تر میشود.

```
فایلهای سرآیند
void  main(void)
{
اعلان  متغیر ها
دستورات اجرائی
}
```

فایلهای سرآیند : در برنامه نویسی زبان c توابع و کتابخانه هایی وجود دارند که قبلاً نوشته شده اند و در فایلهایی (که معمولا در پوشه include یا inc محل نصب کامپایلر میباشند) ذخیره شده اند و قبل از نوشتن تابع اصلی برنامه c این فایل ها باید به تابع اصلی استفاده کننده از توابع این فایلها ضمیمه شوند . به این فایلها فایلهای سرآیند گفته میشود. برای ضمیمه کردن یک فایل بصورت زیر عمل میشود.

```
#include    <نام فایل سرآیند>
```

مثال : توابع ورودی خروجی در فایل stdio.h ذخیره شده اند . در صورت استفاده از توابع ورودی خروجی در برنامه اصلی باید این فایل بصورت زیر به برنامه اصلی ضمیمه شود.

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

در زبان c برنامه نویس میتواند ، فایهای سرآیند تولید کند. حال برای ضمیمه کردن فایل تولید کرده ود بصورت زیر عمل میکند :

```
#include    "نام فایل سرآیند"
```

با روش تولید فایل سرآیند بعداً آشنا میشوید.

انواع داده:

در زبان C پنج نوع داده وجود دارد که این انواع داده با پیشوند های short (کوتاه) ، long (بلند) ، signed (علامتدار) و unsigned (بدون علامت) همراه میشود .

- char (کاراکتر) : این نوع داده برای ذخیره کاراکترهائی مانند 'a' ، '2' ،'x' و... استفاده میشود.
- int (اعداد صحیح) : برای ذخیره اعداد صحیح مانند 12، -34 و... بکار میرود.
- float (اعداد اعشاری) : برای ذخیره اعداد اعشاری مانند 12,5 ، 3.1415 استفاده میشود.
- double (اعداد اعشاری بزرگ): برای ذخیره اعداد اعشاری بزرگ تر از float بکار میرود.

در زیر انواع داده و محدوده آنها نمایش داده شده است:

Type	Size (Bits)	Range
bit	1	0 , 1
unsigned char	8	0 to 255
signed char	8	-128 to 127
int	16	-32768 to 32767
short int	16	-32768 to 32767
unsigned int	16	0 to 65535
signed int	16	-32768 to 32767
long int	32	-2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
signed long int	32	-2147483648 to 2147483647
float	32	±1.175e-38 to ±3.402e38
double	32	±1.175e-38 to ±3.402e38

v در CodeVision :

داده نوع بیت : برای تعریف متغیر های بیتی استفاده میشود. محل ذخیره مغیر های بیتی به ترتیب از بیت صفر ثبات GPIOR1 و سپس GPIOR2 و سپس ثباتهای R2 تا R14 میباشد.

مثال :

```
bit alfa=1; /* bit0 of GPIOR0 */
bit beta;   /* bit1 of GPIOR0 */
```

تعریف متغیر :

متغیر ها نامی برای محل ذخیره داده ها میباشند. تعریف متغیر در C بصورت زیر است :

```
<نوع داده>        <نام متغیر>;
<نوع داده>        < 1نام متغیر >,< 2نام متغیر >,...,< n نام متغیر>;
```

مثال :

```
int   A;
float pi,E,i,j;
unsigned long int    S;
```

نکته : نام متغیر میتواند با حروف بزرگ نوشته شود . فقط کلمات کلیدی زبان C که در قبل گفته شد باید با حروف کوچک نوشته شود.

- در CodeVision :
  - تعریف متغر در حافظه eeprom : برای تعریف متغیر سراسری(در آینده شرح داده میشوند) بصورت زیر انجام میشود:

```
eeprom   <نوع داده>     <نام متغیر>;
```

مثال:

```
eeprom char beta;
eeprom long array1[5];
```

  - تعیین محل ذخیره متغیر های سراسری(در آینده شرح داده میشوند) در حافظه SRAM : این کار بصورت زیر انجام میشود.

```
<نوع داده>        <نام متغیر>@  <آدرس محل ذخیره>;
```

مثال :

```
/* the integer variable "a" is stored in SRAM at address 80h */
int a @0x80;
```

  - برای ذخیره متغیر در ثبات از دستور register استفاده میشود:

```
register   <نوع داده>    <نام متغیر>;
```

  - برای دستور دادن به کامپایلر که متغیری را درثبات ذخیره نکند از دستور volatile بصورت زیر استفاده میشود:

```
volatile    <نوع داده>    <نام متغیر>;
```

مقدار دهی متغیر :

مقداردهی به متغیر به دو صورت زیر انجام میشود:

1. هنگام تعریف متغیر (مقدار دهی اولیه):

```
int   j=0,  a=24;
```

```
char   ch1='a';
float Pi=3.1415,e=2.71;
```

2. بعد از تعریف متغیر با علامت انتساب = .

```
int a,b;
a=23;
b=10;
```

تعریف ثوابت (مقدار ثابت) :

ثوابت مقادیری هستند که در برنامه از آنها استفاده میشود و قابل تغییر نمیباشند. برای تعریف ثوابت در زبان C به روش های زیر عمل میشود:

1. استفاده از دستور #define :

```
#define     <نام ثابت>      <مقدار>
```

مثال :

```
#define          PI    3.14
```

نکته : دستور #define از دستورات پیش پردازنده است و نیازی به ; ندارد.

2. استفاده از دستور const:

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

مثال :

```
const       int      a      =    25;
```

v در CodeVision :

ثوابتی که با const و flash تعریف میشوند در در حافظه flash ذخیره میشوند.

مثال :

```
flash int   integer_constant=1234+5;
flash char char_constant='a';
flash long long_int_constant1=99L;
flash long long_int_constant2=0x10000000;
flash int   integer_array1[]={1,2,3};
flash int   integer_array2[10]={1,2};
flash int   multidim_array[2][3]={{1,2,3},{4,5,6}};
flash char string_constant1[]="This is a string constant";
const char string_constant2[]="This is also a string constant";
```

روشهای نمایش داده ها:

داده ها در زبان C به روش های زیر نمایش داده میشوند:

نوع نمایش	علامت	مثال
دسیمال (دهدهی)	هیچ علامت	23,458,-11,12.5
باینری (دودویی)	با 0b یا 0B شروع میشوند.	0b11110101, 0B10111110
اکتال(هشت هشتی)	با صفر شروع میشوند.	0143,0745,0777
هگزا دسیمال (شانزده شانزدهی)	با 0x یا 0X شروع میشوند.	0x12A1,0XF2DE
بولی(boolean)	0یا false و 1 یا true	true , false
کاراکتر	در بین ‘ ’ قرا میگیرد.	‘s’ , ’%’ , ’$’ ,’@’ , ’g’
رشته	در بین “ ” قرار میگیرد.	“ali” , “FAJR”

مثال :

```
int a=0xffff;
const unsigned int  b=0b01110110;
```

عملگر ها :

عملگر ها نمادهایی هستند که اعمال خاصی را روی عملوند های خود انجام میدهند. عملگر ها در زبان C به دسته های زیر تقسیم میشوند:

1. **عملگر های محاسباتی** : اعمال محاسباتی (ریاضی) مانند جمع ، ضرب ، تقسیم و... را روی عملوند های خود انجام میدهند. این عملوند ها در جدول زیر نمایش داده شده اند:

عملگر	عملکرد	مثال
+	جمع	a+b
-	تفریق	a-b
*	ضرب	a*b
/	تقسیم	a/b
%	باقیمانده تقسیم	a%b
++	افزایش	a++ یا ++a
--	کاهش	a-- یا --a

عملگر باقیمانده تقسیم (%) برای محاسبه باقیمانده تقسیم بکار میرود .

مثال :

```
int a=12,b=5,c=0;
```

```
c=a%b;
```

نتیجه مثال : **بعد از اجرای دستور** c=a%b **مقادیر متغیر های** a **و** b **و**c **بصورت زیر میشود**:

```
a=12  ,  b=3  ,  c=2
```

عملکر افزایش (++) **یک واحد به عملوند خود میافزاید و نتیجه را در همان عملوند قرار میدهد و عملگرکاهش** (--) **یک واحد از عملوند خود کاسته و نتیجه را در همان عملوند قرار میدهد.**

مثال :

```
int  a=20,b=10,c=7,d=15;
a++; ++b;--c;d--;
```

نتیجه مثال : **پس از اجرای دستورات مثال مقدار متغیر ها بصورت زیر میشوند**:

```
a=21 , b=11 , c=6 , d=14
```

نکته : **عملکرد عملگر ها یافزایش و کاهش در عبارات محاسباتی کمی متفاوت است. عبارت محاسباتی ترکیبی از متغیر ها ، ثوابت و عملگر ها است .**

مثال :

```
int a,b,c,d; c=a+b; d=2*a; a=(5+c)/3;
```

اگر در عبارت محاسباتی عملگر(++ **یا** --) **قبل از عملوند ظاهر شوند ، ابتدا مقدار عملوند افزایش یا کاهش یافته و سپس نتیجه عبارت با مقدار جدید عملوند ، محاسبه میشود ، ولی اگر بعد از عملگر ظاهر شود ابتدا با مقدار فعلی عملوند عبارت محاسبه شده و سپس مقدار عملوند افزایش یا کاهش می یابد. در مثال زیر این نکته توضیح داده شده است**:

مثال : **در مثال زیر نتیجه اجرای هر خط نمایش داده شده است . به نتایج دقت کنید**:

```
int  a=20,b=0,c=2,d=0;
b=++a;              // a=21 , b=21 پس از اجرای این سطر
d=c++;              // c=3 , d=3پس از اجرای این سطر
```

2. **عملگر های رابطه ای** : **ارتباط بین عملوند های خود را مانند تساوی ، کوچکتر ، بزرگتر و... را مشخص میکنند.**

عملگر	عملکرد	مثال
>	بزرگتر	a>b
>=	بزرگتر یا مساوی	a>=b

<	کوچکتر	a<b
<=	کوچکتر یا مساوی	a<=b
==	متساوی	a==b
=!	نا متساوی	a!=b

نکته : نتیجه عبارات رابطه ای مقادیر 1 (true) یا 0 (false) است. از این عبارات برای تست شرطهای تساوی ، عدم تساوی ، بزرگتر ، کوچکتر و... استفاده میشود.

مثال :

```
bit  a; int c=25,d=7,f=25;
a=c<f;                  // a=false(0 یا)  پس از اجرای این سطر
a=c<=f;                 // a=true(1 یا)  پس از اجرای این سطر
```

3. **عملگر های منطقی** : اعمال منطقی مانند and ، or و... را روی عملوند ها ی خود انجام میدهند.

عملگر	عملکرد	مثال
|	Not	!x
&&	And	a && b
||	Or	a || b

نکته : نتیجه عبارات منطقی مقادیر 1 (true) یا 0 (false) است. از این عبارات برای تست شرطهای تساوی ، عدم تساوی ، بزرگتر ، کوچکتر و... استفاده میشود.

مثال :

```
bit  a,b,c=true;
b=!c;                   // b=false(0 یا)  پس از اجرای این سطر
a=b && c;               // a=false(0 یا)  پس از اجرای این سطر
a=b || c;               // a=true(1 یا)  پس از اجرای این سطر
```

در جدول های شکل زیر نتیجه عبارات منطقی نمایش داده شده است . دقت شود که true همان یک منطقی و false همان صفر منطقی است.

!	0	1
	1	0

&&	0	1
0	0	0
1	0	1

||	0	1
0	0	1
1	1	1

4. **عملگر های ترکیبی** : این عملگر ها ترکیبی از عملگر های محاسباتی و علامت = تشکیل میشوند(اعمال محاسباتی و انتساب). این عملگر ها در جدول زیر نمایش داده شده اند.

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

5. **عملگر های بیتی** : اعمالی منطقی از قبیل شیفت دادن ، and ، or و... را روی بیتهای عملوند انجام میدهد. این عملگر ها در جدول شکل زیر نمایش داده شده است. تفاوت این عملگر ها با عملگر های منطقی در مثال ها مشخص شده است.

عملگر	عملکرد	مثال	نتیجه اگر a=0b10101111 و b=0b11001001 باشد.
&	and	c=a & b;	c=0b10001001
|	or	c=a | b;	c=0b11101111
^	xor	c=a ^ b;	c=0b01111110
~	not	c=~ a;	c=0b01010000
>>	شیفت به راست	c=a<<2;	c=0b10111100
<<	شیفت به چپ	c=b>>3;	c=0b00011001

6. **عملگر** ?: این عملگر بصورت زیر بکار میرود:

```
<متغیر>  =  <عبارت شرطی>  ?  <عبارت 1>  :  <عبارت2>;
```

اگر نتیجه عبارت شرطی درست باشد، نتیجه عبارت 1 و اگر نتیجه عبارت شرطی نادرست باشد ، نتیجه عبارت 2 را در متغیر قرار میدهد.

مثال :

```
int  a=4,b;
b=a>3 ? a*3 : a/4; // b=12  پس از اجرای این سطر
```

در این مثال شرط عبارت a>3 را بررسی می کند و به دلیل اینکه a>3 درست است ، نتیجه عبارت a*3 را در متغیر b قرار میدهد.

7. **عملگر** sizeof : این عملگر طول یک متغیر یا نوع داده را بر حسب بایت مشخص کند. این دستور به صورت زیر بکار میرود:

```
sizeof        متغیر ;
sizeof        (نوع داده);
```

مثال :

```
int a,b;double c;
a=sizeof  c;          // a=4  پس از اجرای این سطر
b=sizeof(int);                // b=2  پس از اجرای این سطر
```

تقدم عملگر ها:

درعباراتی که چندین عملگر وجود دارد تقدم عملگر ها بصورت جدول زیر میباشد. دقت شود برای بالا بردن تقدم عملگر از پرانتز استفاده میشود. پرانتز تقدم عملگر های داخل خود را بالا میبرد.

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

مثال :

```
int a=2,b=4,c=7,d=1;
a=++b + c / 2 + d;
```

در این مثال با توجه بالاتر بودن تقدم ++ ابتدا عبارت b++ محاسبه میشود و سپس c/2 محاسبه میشود سپس نتایج b++ و c/2 با d جمع میشود. برای بهتر خوانده شدن این عبارت و جلوگیری از اشتباه میتوان با استفاده از پرانتز بصورت زیر تقدم را در این مثال نمایش داد:

```
A=(++b)+(c/2)+(d);
```

تغییر نام انواع داده با typedef :

با این دستور میتوان نام انواع داده موجود را تغییر داد. الگوی استفاده از این دستور بصورت زیر است :

typedef <نوع داده موجود> <نام جدید>

مثال :

```
typedef int integer;
integer a=10,b=49;
```

اشاره گر ها:

آدرس خانه ای از حافظه که متغیر در آن ذخیره شده را اشاره گرگویند . آدرس اولین بایتی از حافظه که به متغیر اختصاص یافته است را آدرس متغیر گویند .

متغیر های اشاره گر :

اشاره گر (آدرس یک متغیر در حافظه) میتواند در متغیری ذخیره شود. این متغیر را متغیر اشاره گر گویند. الوی تعریف متغیر اشاره گر در C بصورت زیر است :

```
<نوع>  *<نام متغیر>;
```

نوع متغیر اشاره گری که میخواهد آدرس یک متغیر را در خود ذخیره کند ، با ید با نوع متغیر یکی باشد.

مثال :

```
int  *p;
```

p متغیر اشاره گری از نوع int است. P آدرس خانه ای از حافظه که محتوای آن از نوع int است در خود نگهداری میکند. P میتواند به خانه هایی از حافظه که محتوی آن خانه ها از نوع int است اشاره کند.

عملگر های اشاره گر :

دو عملگر & و* در عملیاتهای اشاره گر ها استفاده میشوند. عملگر & آدرس عملوند خود را مشخص میکند. و عملگر * محتویات جایی را مشخص میکند که که عملوندش به آن اشاره میکند.

مثا ل:

```
int  *p,m,s;
m=5;
p=&m;
s=*p;
```

v در CodeVision :

اشاره گر ها در CodeVision بصورت زیر تعریف میشوند:

```
<نام متغیراشاره گر>; [<محل ذخیره اشاره گر >]*   <نوع>  [<نوع محل ذخیره داده>]
```

یا

```
<نام متغیراشاره گر>; [<محل ذخیره اشاره گر >]*   [<نوع محل ذخیره داده>]  <نوع>
```

نکته :اگر محل ذخیره تعیین نشده باشد بطور پیش فرض SRAM است.

مثال :

```
//اشاره گر به یک رشته از کاراکتر که در SRAM قرار گرفته شده
char *ptr_to_ram=”This string is placed in SRAM”;
//اشاره گر به یک رشته از کاراکتر که در Flash قرار گرفته شده
flash char *ptr_to_flash1=”This string is placed in FLASH”;
char flash *ptr_to_flash2=”This string is also placed in FLASH”;
//اشاره گر به یک رشته از کاراکتر که در EEPROM قرار گرفته شده
eeprom char *ptr_to_eeprom1="This string is placed in EEPROM";
char eeprom *ptr_to_eeprom2="This string is also placed in EEPROM";
//اشاره گر ذخیره شده در FLASH که به یک رشته از کاراکتر اشاره میکند
char * flash flash_ptr_to_ram=”This string is placed in SRAM”;
//اشاره گر ذخیره شده در FLASH که به یک رشته از کاراکتر که در FLASH قرار گرفته اشاره میکند
flash char * flash flash_ptr_to_flash=”This string is placed in FLASH”;
//اشاره گر ذخیره شده در FLASH که به یک رشته از کاراکتر که در EEPROM قرار گرفته اشاره میکند
eeprom char * flash eeprom_ptr_to_eeprom="This string is placed in EEPROM";
//اشاره گر ذخیره شده در EEPROM که به یک رشته از کاراکتر که در SRAM قرار گرفته اشاره میکند
char * eeprom eeprom_ptr_to_ram=”This string is placed in SRAM”;
//اشاره گر ذخیره شده در EEPROM که به یک رشته از کاراکتر که در FLASH قرار گرفته اشاره میکند
flash char * eeprom eeprom_ptr_to_flash=”This string is placed in FLASH”;
//اشاره گر ذخیره شده در EEPROM که به یک رشته از کاراکتر که در EEPROM قرار گرفته اشاره میکند
eeprom char * eeprom eeprom_ptr_to_eeprom="This string is placed in EEPROM";
```

v در CodeVision :

در زیر مثالی از دسترسی به حافظه eeprom با استفاده از اشاره گر ها نمایش داده شده است:

```
/* The string is stored in the EEPROM during chip programming */
eeprom char string[]=”Hello”;
void main(void) {
int i;
/* Pointer to EEPROM */
int eeprom *ptr_to_eeprom;
/* Write directly the value 0x55 to the EEPROM */
alfa=0x55;
/* or indirectly by using a pointer */
ptr_to_eeprom=&alfa;
*ptr_to_eeprom=0x55;
```

```
/* Read directly the value from the EEPROM */
i=alfa;
/* or indirectly by using a pointer */
i=*ptr_to_eeprom;
}
```

آرایه ها:

آرایه ها مجموعه ای از عناصر (متغیر یا ثوابت) همنوع تحت یک نام میباشند . هر عضو آرایه دارای اندیسی است که با آن اندیس به عناصر آرایه دسترسی پیدا میکنیم . آرایه میتواند یک ، دو یا چند بعدی باشد. ماتریس آرایه ای n بعدی میباشد.

آرایه تک بعدی : آرایه تک بعدی لیستی از عناصر همنوع است که به صورت زیر تعریف میشود:

```
<نوع عناصر آرایه>    نام آرایه [طول آرایه];
```

عناصر آرایه تک بعدی به صورت زیر دسترسی میشوند :

```
نام آرایه [اندیس];
```

اندیس عناصر آرایه از صفر شروع میشود.

مثال :

```
int  a [5];
```

آرایه این مثال و عناصر آن را میتوان به صورت شکل زیر توصیف کرد:

a

a[0]	a[1]	a[2]	a[3]	a[4]

مقدار دهی عناصر آرایه تک بعدی :

به دو روش زیر میتوان عناصر آرایه را مقدار دهی کرد:

- مقدار دهی اولیه در زمان تعریف آرایه : بصورت زیر انجام میشود:

```
<نوع عناصر آرایه>    نام آرایه [طول آرایه] = {مقدار عنصر1, مقدار عنصر2,...,مقدار عنصرn};
```

نکته : اگر در زمان مقدار ده یبه آرایه طول آرایه مشخص نشده با شد ، طول آرایه برابر مقداری که به نسبت داده شده است ، در نظر گرفته میشود.

مثال :

```
int  a[]={1,2,3};        // طول آرایه 3 در نظر گرفته میشود
```

نکته : اگر تعداد مقادیری که به آرایه نسبت داده شد ، کمتر از طول در نظر گرفته برای آرایه باشد . باقی عناصر آرایه صفر در نظر گرفته میشود.

مثال :

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

- مقدار دهی عناصر آرایه با دسترسی به عناصر آرایه : بصورت زیر انجام میشود:

```
نام آرایه  [اندیس آرایه]=مقدار عنصر آرایه ;
```

مثال :

```
int  a[5]={34,12,35,11,6};
int b[3];
b[0]=2;b[1]=5;b[2]=16;
```

آرایه دو بعدی : آرایه دو بعدی به صورت زیر تعریف میشود:

```
نام آرایه  [طول بعد 1][طول بعد 2] ;< نوع عناصرآرایه
```

عناصر آرایه تک بعدی به صورت زیر دسترسی میشوند :

```
نام آرایه  [اندیس بعد1][اندیس بعد2] ;
```

اندیس عناصر آرایه از صفر شروع میشود.

مثال :

```
int  a [3] [5];
```

عناصر آرایه این مثال آن را میتوان به صورت شکل زیر توصیف کرد:

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

نکته: بعد 1 آرایه دو بعدی را میتوان به عنوان سطر و بعد 2 آن را بعنوان ستون یک ماتریس توصیف کرد.

مقدار دهی عناصر آرایه دو بعدی :

به دو روش زیر میتوان عناصر آرایه را مقدار دهی کرد:

- مقدار دهی اولیه در زمان تعریف آرایه : بصورت زیر انجام میشود:

```
نام آرایه  [طول بعد 1][طول بعد 2]=<نوع عناصرآرایه {مقادیر عناصر};
```

- مقدار دهی عناصر آرایه با دسترسی به عناصر آرایه : بصورت زیر انجام میشود:

```
نام آرایه  [اندیس بعد1]  [اندیس بعد2]=مقدار عنصر آرایه ;
```

مثال :

```
int  a[3][5]={{3,2,3,1,6},{7,9,2,1,3},{1,5,4,6,8}};
int  b[3][7];
b[0][0]=2;b[1][6]=5;b[6][4]=16;…
```

آرایه n بعدی : الگوی کلی تعریف آرایه n بعدی بصورت زیر میباشد :

```
نوع عناصرآرایه>  نام آرایه  [طول بعد 1][طول بعد 2]…[طول بعد n];
```

عناصر آرایه به صورت زیر دسترسی میشوند :

```
نام آرایه [اندیس بعد1][اندیس بعد 2]…[اندیس بعدn];
```

نکته : در میکرو کنترلر ها به علت فضای حافظه محدود از آرایه حداکثر با دو بعد استفاده میشود.

رشته ها : رشته ها آرایه ای از نوع کاراکتر میبا شند که با کاراکتر تهی (\n) به اتمام میرسد.

```
char  نام رشته [طول رشته];
```

نکته : با توجه به اینکه کاراکتر آخر رشته باید کاراکت خالی باشد بنابراین برای رشته ای به طول n حداکثر میتوان n-1 کاراکتر قرار داد.

مثال

```
char  name[10];

name="fajr";        //   | f | a | j | r | \n | ? | ? | ? | ? | ? |

name="emam hossein";  //   | e | m | a | m |  | h | o | s | s | \n |
```

مقدار دهی رشته :

به دو روش انجام میشود:

- قرار دادن رشته در بین کوتیشن .

```
char  نام رشته [طول رشته]="رشته مورد نظر";
```

- مقدار دهی هر یک عناصر رشته به عنوان عنصر آرایه.

```
char  نام رشته[]={'کاراکتر 1','کاراکتر 2',...,'کاراکتر n'};
```

مثال :

```
char name[10]="ali";
char name[10]={'a' , 'l' , 'i'};
```

ساختار :

ساختار مجموعه ای از عناصر غیر همنوع تحت یک نام میباشند. برا ی استفاده از ساختار ابتدا باید نوع ساختار را تعریف سپس متغیر هایی از نوع ساختار ایجاد و سپس از آن متغیر ها در برنامه استفاده کرد.

تعریف نوع ساختار :

قبل از استفاده از ساختار نوع ساختار را تعریف کرد. برای تعریف نوع ساختار از کلمه لیدی struct استفاده میشود . مقدار حافظه اشغال شده توسط یونیون ها(طول ساختار) برابرمجموع اندازه عناصر ساختار میباشد. الگوی تعریف ساختار بصورت زیر میباشد:

```
struct    <نام ساختار>{
          <نوع عنصر 1>    نام عنصر 1;
          <نوع عنصر 2>    نام عنصر 2;
              ...
          <نوع عنصر n>    نام عنصر n;
};
```

تعریف متغیرهایی از نوع ساختار : برای تعریف متغیر هایی از نوع ساختار از دو روش استفاده میشود.

- تعریف متغیر های ساختار در حین تعریف نوع ساختار: بصورت الگوی زیر بالا فاصله بعد از تعریف نوع ساختار متغیر های ساختار تعریف میشوند.

```
struct    <نام نوع ساختار>{
          <نوع عنصر 1>    نام عنصر 1;
          <نوع عنصر 2>    نام عنصر 2;
              ...
          <نوع عنصر n>    نام عنصر n;
}نام متغیر 1,نام متغیر 2,...;
```

- تعریف متغیر های ساختار بعد از تعریف نوع ساختار: بعد از تعریف نوع ساختار با استفاده از کلمه کلیدی struct متغیر های نوع ساختار تعریف میشوند. الگوی تعریف متغیر های نوع ساختار به این روش بصورت زیر است:

```
struct    <نام نوع ساختار>  <نام متغیر1>,<نام متغیر2>,...,<نام متغیر n>;
```

دسترسی به عناصر ساختار : برای دسترسی به هر یک از عناصر نوع ساختار از الوی زیر استفاده میشود:

```
<نام عنصر نوع ساختمان>.<نام متغیرنوع ساختار>
```

مقدار دهی متغیر های نوع ساختار: متغیر های نوع ساختمان مانند آرایه مقدار دهی میشوند.

مقدار دهی به متغیرهای نوع ساختار به دو روش زیر انجام میشود :

- مقدار دهی متغیر های نوع ساختمان در زمان تعریف متغیر: الگوی مقدار دهی به این روش بصورت زیر است:

```
struct  <نام نوع ساختار> <نام متغیر>={مقدارعنصر n,...,مقدار عنصر2,مقدار عنصر1};
```

- مقدار دهی با دسترسی به عناصر نوع ساختار: الگوی مقدار دهی به این روش بصورت زیر است:

```
<نام متغیرنوع ساختار>.<نام عنصر نوع ساختمان>=مقدار عنصر ساختار;
```

مثال :

```
struct  student{
                        char name[10];          //نام دانشجو
                        int mark[7];            //شماره دانشجویی
                        float number[3];   //نمره دروس:ریاضی ,مدار ,الکترونیک
};
struct  student   stu1={“ali”,{8,0,0,0,1,9,0},{15,12.5,17.5}};
struct  student   stu2={“hamed”,{8,0,0,0,1,9,1},{12,13.5,5}};
struct  student   stu3={0};    //همه عناصر صفر (ترک تحصیل)
```

اشاره گر های ساختمان:

تعریف اشاره گر های ساختمان مانند تعریف متغیر های ساختار است فقط از عملگر* قبل از متغیر استفاده میشود.

مثال :

```
struct  student{
                        char name[10];          //نام دانشجو
                        int mark[7];            //شماره دانشجویی
```

```
                        float number[3];   //نمره دروس:ریاضی,مدار,الکترونیک

}person,*p;
```

مثال :

```
struct  student{
                        char name[10];        //نام دانشجو
                        int mark[7];          //شماره دانشجویی
                        float number[3];   //نمره دروس:ریاضی,مدار,الکترونیک
};
struct  student *p;
```

یونیون:

یونیون مکانی از حافظه است که به وسیله یک یا چند متغیر بصورت اشتراکی استفاده میشود. عناصر یونیون از یک محل حافظه شروع میشوند. مقدار حافظه اشغال شده توسط یونیون ها(طول یونیون) برابراندازه عنصری از یونیون که بیشترین طول را دارد ، میباشد. الگوی تعریف یونیون ها در زبان c به صورت زیر میباشد:

```
union  <نام یونیون> {
    <نوع عنصر1>  <نام عنصر 1>;
    <نوع عنصر2>  <نام عنصر 2>;
    ...
    <نوع عنصر n>  <نام عنصر n>;
    };
```

نکته : یونیون ساختاری است که آدرس شروع کلیه عناصر آن از یک نقطه است.

نکته : عملکردهای دیگر یونیون از قبیل تعریف متغیر نوع یونیون ، مقدار دهی متغیر های نوع یونیون و.. کاملا شبیه ساختار است.

انواع داده شمارشی :

بوسیله نوع داده شمارشی عناصر یک مجموعه متناهی شماره گذاری میشود . الگوی استفاده از نوع شمارشی بصورت زیر است:

```
enum    < نام نوع شمارشی >{
        <نام عنصر 1>,
        <نام عنصر 2>,
       ...
        <نام عنصر n>
       };
```

نکته : به استفاده از کاما در تعریف نوع داده شمارشی دقت شود.

نکته : تعریف متغیر های نوع داده شمارشی و مقدار دهی متغیر های نوع داده شمارشی و... مانند ساختار است.

مثال :

```
eeprom enum months{
    January, February, March,April,May, June,July,August
    ,September,October, November, December
} months_of_year;
```

مثال :

```
enum months{
    January, February, March,April,May, June,July,August
    ,September,October, November, December
};
enum  months  months_of_year;
```

به ترتیب از عنصر اول نوع داده شمارشی تا عنصر آخر داده شمارشی یک عدد صحیح نسبت داده میشود .به عنصر اول ، صفر و به عنصر دوم ، یک و... و به عنصر n ام مقدار n-1 نسبت داده میشود.

مثال:

```
/* January=0 , February=1, March=2,..., December=11 */
enum months{
    January, February, March,April,May, June,July,August
```

```
    ,September,October, November, December
} months_of_year;
```

البته برنامه نویس میتواند این مقادیر را تغییر دهد. به مثال زیر توجه کنید:

مثال :

```
/* January=1 , February=2, March=3,..., December=12 */
eeprom enum months{
    January=1, February, March,April,May, June,July,August
    ,September,October, November, December
} months_of_year;
```

حلقه های تکرار :

با استفاده از حلقه های تکرار (تحت شرایط خاص) ، یک یا چند دستور به تعداد مورد نظر تکرار میشود.

- **حلقه for** : **با استفاده از حلقه for تعدادی دستور به تعداد دفعات مشخص تکرار میشود . الگوی استفاده از حلقه تکرار for بصورت زیر است :**

```
for (مقدار اولیه متغیر شمارش حلقه; شرط حلقه ; گام شمارش حلقه)
{
        دستورات حلقه
}
```

متغیر شمارش حلقه : متغیری که در هر بار تکرار حلقه به اندازه گام شمارش حلقه افزایش یا کاهش می یابد.

شرط حلقه : در هر مرحله از تکرار حلقه این شرط بررسی شده و در صورت درست بودن شرط دستورات حلقه تکرار میشود و در غیر این صورت تکرار حلقه به اتمام میرسد.

گام شمارش حلقه : مقداری که پس از هر بار اجرای دستورات حلقه به متغیر حلقه اضافه شده یا از آن کاسته میشود.

مثال :

```
int counter=0;
int i,counter;     //تعریف متغیر شمارش حلقه
for (i=0;i<10,i++)
{
        counter++;
};
```

نکته: اگر حلقه دارای فقط یک دستور باشد نیازی به قرار دادن آن دستور در بیت } و { نیست.

```
for (مقدار اولیه متغیر شمارش حلقه; شرط حلقه ; گام شمارش حلقه) دستور حلقه ;
```

برای ایجاد یک حلقه بینهایت بوسیله for از الگوی زیر استفاده میشود.

```
for ( ; ; )
{
        دستورات حلقه
}
```

- **ساختار تکرار** while :

الگوی ساختار while **بصورت زیر است :**

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

پس از رسیدن برنامه به این ساختار اگر شرط دارای ارزش درستی باشد (نتیجه شرط 1 **باشد) دستورات حلقه اجرا میشود . وقتی که به پایان دستورات رسید دوباره شرط حلقه بررسی شود ، دوباره دستورات لقه اجرا میشوند و این کار تا زمانی که شرط حلقه برقرار باشد اجرا میشوند . و اگر شرط حلقه برقرار نباشد (نتیجه شرط حلقه صفر باشد) برنامه از حلقه خارج میشود.**

نکته : اگر به جای شرط مقدار 1 **قرار داده شود ، بدان معنی است که شرط حلقه همیشه برقرار است و بنابراین حلقه همواره تکرار میشود (حلقه بی نهایت).**

```
while(1)
{
                دستورات حلقه;
}
```

نکته: اگر حلقه دارای فقط یک دستور باشد نیازی به قرار دادن آن دستور در بیت } و { نیست.

```
While (شرط حلقه)        دستور حلقه ;
```

- **ساختار تکرار** do ...while() :

الگوی این ساختار بصورت زیر است :

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

این ساختار مانند ساختار قبل است ، با این تفاوت که وقتی برنامه به do **میرسد دستورت حلقه اجرا میشود و در انتهای اجرای دستورات شرط حلقه بررسی میشود و اگر درست**

باشد ، دوباره دستورات حلقه جرا میشود و اگر نادرست باشد ، برنامه از حلقه خارج میشود.

نکته : الگوی ساختار حلقه بینهایت در اینجا بصورت زیر میباشد :


```
do
{
                دستورات حلقه ;
}while(1);
```

ساختار های انتخاب:

- **ساختار انتخاب** if :

این ساختار شرطی را بررسی میکند ، اگر شرط برقرار باشد ، مجموعه ای از دستورات را اجرا میکند و در غیر این صورت مجموعه ای دیگر از دستورات را اجرا میکند الگوی این ساختار بصورت زیر است:

```
if (شرط انتخاب)
{
                دستورات ساختار انتخاب ;
}
else
{
}
```

نکته: اگر ساختار انتخاب فقط دارای یک دستور باشد نیازی به قرار دادن آن دستور در بیت } و { نیست.

```
if (شرط انتخاب)          دستور ساختار انتخاب ;
else    دستور ساختار انتخاب;
```

- **ساختار انتخاب** if...else if...else... :

این ساختار شرط انتخاب 1 را بررسی میکند ، اگر شرط برقرار باشد ، مجموعه ای از دستورات را اجرا میکند و در غیر این صورت شرط انتخاب 2 را بررسی میکند اگر شرط برقرار باشد ، مجموعه ای از دستورات را اجرا میکند .... اگر هیچکدام از شرط ها برقرا ر نباشد ، مجموعه ای دیگر از دستورات را اجرا میکند. الگوی این ساختار بصورت زیر است:

```
if (شرط انتخاب1)
```

```
{
                دستورات ساختار انتخاب ;
}
else   if(2شرط انتخاب)
{
                دستورات ساختار انتخاب ;
}
...
else   if(n  شرط انتخاب)
{
                دستورات ساختار انتخاب ;
}
else
{
                دستورات ساختار انتخاب ;
}
```

- **ساختار انتخاب** switch…case :

الگوی این ساختار در زیر نمایش داده شده است:

```
switch(عبارت)
{
case  <1 مقدار>  :{1 دستورات;
                        }
case  <2 مقدار>  :{2 دستورات;
                        }
...
case  <n مقدار>  :{   n  دستورات;
                        }
default  :{n+1 دستورات;
                }
}
```

این ساختار عبارتی را بررسی میکند و عبارت را با چند مقدار مقایسه میکند اگر عبارت برابربا یک مقدار باشد ، دستورات مربوطه را اجرا میکند و اگر برابر با هیچ کدام از عبارات نباشد (default) دستورات دیگری را اجرا میکند.

تابع :

در برنامه نویسی به زبان C به منظور ساده کردن برنامه های پیچیده به بخش های کوچک ساده مستقل از توابع استفاده میشود. استفاده از توابع به خوانا تر کردن و ماژولار کردن برنامه کمک میکند. در شکل زیر دو الگویی که از آنها برای نوشتن برنامه ای که از چند تابع تشکیل شده است ، استفاده میشود، نمایش داده شده است:

```
< دستور های پیش پردازنده >
<اعلان تابع 1>;
<اعلان تابع 2>;
...
<اعلان تابع n>;
void main(void){
        دستورات تابع اصلی;
}
<تعریف تابع 1>{
        دستورات تابع 1;
};
<تعریف تابع 2>{
        دستورات تابع 2;
};
...
<تعریف تابع n>{
        دستورات تابع n;
};
```

الگوی 1(تعریف تابع بعد از تابع اصلی)

```
< دستور های پیش پردازنده >
<تعریف تابع 1>
{
        دستورات تابع 1;
};
<تعریف تابع 2>
{
        دستورات تابع 2;
};
...
<تعریف تابع n>
{
        دستورات تابع n;
};
void main(void)
{
        دستورات تابع اصلی;
}
```

الگوی 2(تعریف تابع قبل از تابع اصلی)

در برنامه نویسی به زبان C توابع دارای دو جنبه هستند:

- جنبه تعریف تابع
- جنبه فراخوانی تابع

1. جنبه تعریف تابع : تعریف تابع در برنامه به دو روش انجام میشود :

- **تعریف تابع بعد از تابع اصلی (main) : تعریف تابع در این روش بصورت زیر انجام میشود:**

الف) ابتدا قبل از تابع اصلی ، توابع مورد نظر اعلان میشوند . روش اعلان تابع ، قبل از تابع اصلی از الگوی زیر پیروی میکند :

<نوع تابع> نام تابع ([لیست پارامتر های تابع]);

نکته : به بکارگیری ; در اعلان تابع و تعریف تابع دقت شود.

- **نوع تابع : اگر تابع بخواهد مقداری را برگرداند نوع مقدار بازگشتی در این قسمت بیان میشود . اگر تابع هیچ مقداری را برنمیگرداند برای نوع تابع از کلمه کلیدی void استفاده میشود.**
- **نام تابع : نام منحصر به فردی است که در برنامه برای فراخوانی تابع از آن نام استفاده میشود. نام تابع برای خوانایی برنامه بهتر است با عملکرد تابع همخوانی داشته باشد.**
- **لیست پارامتر های تابع: مقادیری است که در زمان فراخوانی تابع از برنامه به تابع ارسال میشود. اگر تابع فاقد لیست پارامتر ها باشد به جای لیست پارامتر ها کلمه کلیدی void قرار میگیرد و یا بین دو پراتز چیزی قرار نمیگیرد. لیست پارامتر های تابع بصورت زیر نوشته میشوند :**

<نوع پارامتر 1> [<نام پارامتر 1>] ,<نوع پارامتر 2> [<نام پارامتر 2>] ,...,<نوع پارامتر n> [<نام پارامتر n>]

نکته: در اعلان تابع نوشتن نام پارامتر اختیاری است و فقط نوشتن نوع پارامتر کافی است. (در این گزاش بخشهای اختیاری برای نشان دادن اختیاری بودن آنها در بین [] قرا ر داده شده اند) در این روش لیست پارامتر های تابع بصورت زیر تعریف میشود :

<نوع پارامتر 1> , <نوع پارامتر 2> , ... ,<نوع پارامتر n>

- **مقدار بازگشتی از تابع: نتیجه دستورات تابع است که به برنامه ای که تابع را فراخوانی کرده بازگردانده میشود. برای برگرداندن مقداری به برنامه ای که**

تابع را فراخوانی کرده از کلمه کلیدی return استفاده میشود. برگرداندن نتیجه تابع به صورت زیر انجام میشود:

```
return    <مقدار بازگشتی>;
```

ب) بعد از اعلان تابع ، پس از تابع اصلی ، تابع اعلان شده باید تعریف شود . در شکل زیر الگوی ساده ای از تعریف یک تابع نمایش داده شده است :

قسمتهای مختلف الگوی تعریف تابع در قسمت اعلان تابع بیان شده است .

```
<نوع تابع>    نام تابع    ([لیست پارامتر های تابع])
{
        بیان تابع
[return <مقدار بازگشتی ازتابع>;]
}
```


- تعریف تابع قبل از تابع اصلی : در این روش تعریف تابع نیازی به اعلان تابع نیست . الگوی تعریف تابع از الگوی قسمت ب از روش اول تعریف تابع پیروی میکند.

2. جنبه فراخوانی تابع : پس از آنکه تابع تعریف شد ، برای استفاده از تابع در برنامه ای که از آن استفاده میکند ، تابع فراوانی میشود. الگوی استفاده شده برای فراخوانی تابع بصورت زیر است :

```
<نام تابع>  ([لیست آرگومانهای تابع])
```

- نام تابع : همان نام منحصر به فرد اختصاص یافته شده در قسمت تعریف تابع میباشد.
- لیست آرگومانهای تابع : لیست مقادیر مورد نیاز تابع است که در زمان فراخوانی تابع ، به تابع ارسال میشود . این لیست متناظر با لیست پارامتر های تابع در زمان تعریف تابع است . هر آرگومان متناظر با یک پارامتر در تعریف تابع است. ترتیب آرگومانها متناظر با ترتیب پارامتر ها میباشد. اگر تابع فاقد لیست آرگومان باشد (تعریف تابع فاقد لیست پارامتر باشد) جای لیست آرگومانهای تابع خالی نگه داشته میشود. نوع آرگومانهای تابع با نوع پارامتر ها باید یکی باشد. لیست آرگومانهای تابع بصورت زیر تعریف میشود:

```
<آرگومان 1> , <آرگومان 2> , ... , <آرگومان n>
```

مثال: (تعریف تابع قبل از تابع اصلی)

عملکرد تابع : میانگین سه مقدار اعشاری را برمیگرداند.

```
<دستورات پیش پردازنده>
// تعریف تابع
// نوع تابع (نوع مقدار بازگشتی): float
// نام تابع : average_of_number
// لیست پارامتر های تابع : float x,float y,float z
float average_of_number(float x,float y,float z)
{
// بیان تابع
float sum,aver;
sum=x+y+z;aver=sum/3;
// مقدار بازگشتی
return  aver;
};
// تابع اصلی
void main (void)
{
float aver1;
float a=12.5,b=75,c=36.7;
// تابع فراخوانی شده و مقدار بازگشتی آن در aver1 قرار داده میشود.
// آرگومانهای تابع : a,b,c
aver1= average_of_number(a,b,c);
// دستورات دیگر تابع
...
};
```

مثال: (تعریف تابع بعد از تابع اصلی)

عملکرد تابع : میانگین سه مقدار اعشاری را برمیگرداند.

```
<دستورات پیش پردازنده>
// اعلان تابع
// نوع تابع (نوع مقدار بازگشتی): float
// نام تابع : average_of_number
// لیست پارامتر های تابع : float x,float y,float z
```

```
float average_of_number(float x,float y,float z);
// روش دیگر اعلان تابع : float average_of_number(float,float,float);
// تابع اصلی
void main (void)
{
float aver1;
float a=12.5,b=75,c=36.7;
// تابع فراخوانی شده و مقدار بازگشتی آن در aver1 قرار داده میشود.
// آرگومانهای تابع : a,b,c
aver1= average_of_number(a,b,c);
// دستورات دیگر تابع
...
};
// تعریف تابع
// نوع تابع (نوع مقدار بازگشتی): float
// نام تابع : average_of_number
// لیست پارامتر های تابع : float x,float y,float z
float average_of_number(float x,float y,float z)
{
// بیان تابع
float sum,aver;
sum=x+y+z;aver=sum/3;
// مقدار بازگشتی
return aver;
};
```

مثال : (تابعی که مقداری را برنمیگرداند)

عملکرد تابع : روش ایجاد یک تاخیر غیر دقیق در برنامه اصلی . در این برنامه از تعریف تابع به روش اول استفاده شده است.

```
<دستورات پیش پردازنده>
// تعریف تابع
// نوع تابع (نوع مقدار بازگشتی): void (هیچ مقداری را برنمیگرداند)
// نام تابع : delay
// لیست پارامتر های تابع : int x,int y
void delay (int x,int y)
```

```
{
// بیان تابع
int i,j;
// استفاده از حلقه تو در تو برای تاخیر
for(i=0;i<x,i++)   for(j=0,j<y,j++);
};
// تابع اصلی
void main (void)
{
bit a;
// دستورات دیگر تابع
...
while(1)
{
      a=1;
      delay(100,50);
      a=0;
      delay(200,150)
};
```