www.sbargh.ir

جزوه برنامه نویسی C ویژه میکروکنترلرها

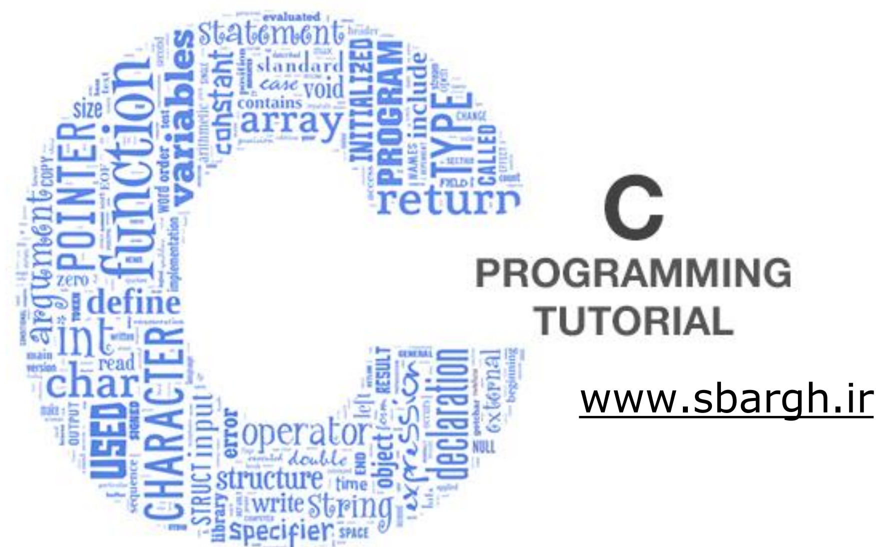
مقدماتی و پیشرفته (از ۰ تا ۱۰۰)

تهیه و تالیف : شجاع داودی

www.sbargh.ir

مقدمه مولف

با توجه به استفاده از زبان برنامه نویسی C به صورت گسترده در انواع میکروکنترلرها به عنوان منعطف ترین زبان برنامه نویسی و لزوم درک این زبان پرکاربرد ، شاهد عدم استفاده از تمام قابلیت های این زبان در پروژه های مربوطه هستیم. در جزوه حاضر سعی شده است تا این قابلیت های برنامه نویسی پیشرفته تشریح و در اختیار همگان قرار گیرد و نکاتی که در برنامه نویسی پیشرفته باید رعایت شود تا سیستم روان ، بدون هنگ و بهینه باشد. همچنین میتوانید نظرات و پیشنهادات خود را در رابطه با بهتر کردن این جزوه به آدرس ایمیل زیر ارسال نمایید.



معرفی کوتاه زبانC

زبان برنامه‌نویسی سی ، زبانی همه منظوره ، ساخت یافته و روندگرا می‌باشد که در سال ۱۹۷۲ توسط دنیس ریچی در آزمایشگاه‌های بل ساخته شد . به طور کلی زبان های برنامه نویسی را می توان در سه سطح دسته بندی کرد: زبان های سطح بالا، زبان های سطح میانی، زبان های سطح پایین. زبان سی یک زبان سطح میانی است که در آن هم می توان به سطح بیت و بایت و آدرس دسترسی داشت و هم از مفاهیم سطح بالا که به زبان محاوره ای انسان نزدیکتر است (مانند حلقه های شرطی if … else و حلقه های تکرار for و while و …) بهره گرفت . در زبان سی هیچ محدودیتی برای برنامه نویس وجود ندارد و هر آنچه را که فکر می کنید ، می توانید پیاده سازی کنید . ارتباط تنگاتنگی بین زبان c و اسمبلی وجود دارد ، به این صورت که می توان از برنامه نویسی اسمبلی در هر کجای برنامه زبان سی استفاده کرد.

کلمات کلیدی در زبانC

زبان سی زبان کوچکی است چرا که در آن تعداد کلمات کلیدی ۳۲ کلمه است. کم بودن کلمات کلیدی در یک زبان مبنی بر ضعف آن نیست . زبان بیسیک حاوی ۱۵۰ کلمه کلیدی است اما قدرت زبان سی به مراتب بالاتر است. زبان سی به حروف کوچک و بزرگ حساس است . (Case Sensitive) در این زبان بین حروف کوچک و بزرگ تفاوت است و تمام کلمات کلیدی باید با حروف کوچک نوشته شوند. در شکل زیر تمام کلمات کلیدی زبان سی را مشاهده می کنید.

auto	double	int	struct
break	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

نکته ۱ : کلمه کلیدی auto از کامپایلر Codevision حذف شده است.

نکته ۲ : کلمات کلیدی زیر به کامپایلر کدویژن اضافه شده است :

bit	defined	interrupt
flash	eeprom	inline
	sfrb	sfrw

ویژگی های یک برنامه به زبانC

-هر دستور در زبان سی با ; به پایان می رسد.

-حداکثر طول هر دستور ۲۵۵ کاراکتر است.

-هر دستور می تواند در یک یا چند سطر نوشته شود.

-برای توضیحات تک خطی از // در ابتدای خط استفاده می شود و یا توضیحات چند خطی در بین */ و /* قرار می گیرد.

ساختار یک برنامه به زبان C در کامپیوتر

هر زبان برنامه نویسی دارای یک ساختار کلی است . این ساختار یک قالب را برای برنامه نویس فراهم می کند . ساختار کلی یک برنامه به زبان C را در زیر مشاهده می کنید.

```
#include < HeaderFiles.h >
محل معرفی متغیرهای عمومی ، ثوابت و توابع
void main (void)
{
محل مربوط به کدهای برنامه
}
```

بنابراین همانطور که مشاهده می شود:

۱. خطوط ابتدایی برنامه ، دستور فراخوانی فایل های سرآمد (Header Files) می باشد . فایل های سرآمد فایل هایی با پسوند h هستند که حاوی پیش تعریف ها و الگو های توابع می باشند.
۲. قالب اصلی برنامه ب مبنای تابعی به نام main بنا شده است . تابعی که اصولا ورودی و خروجی ندارد و کدهای اصلی برنامه را در خود دارد.

www.sbargh.ir

تفاوت برنامه نویسی برای کامپیوتر و میکروکنترلر

ساختار فوق یک قالب کلی در برنامه نویسی زبان C در کامپیوتر (ماشین CISC) با هدف اجرا در کامپیوتر را نشان می دهد .برنامه نویسی میکروکنترلر ها (ماشین RISC) با هدف اجرا در میکروکنترلر ، با برنامه نویسی در کامپیوتر کمی متفاوت است . تفاوت اصلی در کامپیوتر این است که عامل اجرای برنامه ، در زمان نیاز به عملکرد آن ، یک کاربر است . هر زمان که کاربر نیاز به عملکرد برنامه داشته باشد آن را اجرا می کند و نتیجه را بررسی می کند . اما در میکروکنترلر رفتار یک آی سی است که عامل اجرای برنامه منبع تغذیه است . با وصل منبع تغذیه برنامه شروع به کار می کند و تا زمانی که منبع تغذیه وصل است باید کارهای مورد نیاز را دائما اجرا نماید .

ساختار برنامه میکروکنترلر به زبان C

برنامه ای که کاربر می نویسد باید طوری نوشته شود که وقتی روی آی سی پروگرام شد دائما اجرا شود. راه حل این مسئله قرار دادن کدهای برنامه درون یک حلقه نامتناهی است . این عمل باعث می شود تا میکروکنترلر هیچگاه متوقف نشود و بطور مداوم عملکرد طراحی شده توسط کاربر را اجرا کند. بنابراین ساختار یک برنامه به صورت زیر در می آید.

```
#include < HeaderFiles.h >

محل معرفی متغیرهای عمومی ، ثوابت و توابع

void main (void)

{

کدهایی که در این محل قرار میگیرند فقط یکبار اجرا می شوند

معمولا مقدار دهی اولیه به رجیستر ها در این ناحیه انجام می شود

while(1)

{

کدهایی که باید مدام در میکروکنترلر اجرا شوند
```

}

}

متغیرها در زبان C

یک متغیر محدوده ای از فضای حافظه است که با یک نام مشخص می شود. یک متغیر بسته به نوع آن می تواند حامل یک مقدار عددی باشد . یک متغیر می تواند در محاسبات شرکت کند و یا نتیجه محاسبات را در خود حفظ کند. در کل میتوان گفت که نتایج بخش های مختلف یک برنامه ، در متغیر ها ذخیره می شود. در جدول زیر انواع متغیرها ، فضایی که در حافظه اشغال می کنند و بازه مقدار پذیری آنها را در **کامپایلر کد ویژن** مشاهده می کنید.

Type	Size (Bits)	Range
bit	1	0 , 1
bool, _Bool	8	0 , 1
char	8	-128 to 127
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

نحوه تعریف متغیرها

متغیر ها به صورت زیر تعریف می شوند:

نوع متغیر   نام متغیر = مقدار اولیه ;

Unsigned char A=12;

int a,X,j;

توضیح : در خط اول یک متغیر ۸ بیتی بدون علامت با نام A که تنها میتواند مقادیر ۰ تا ۲۵۵ بگیرد ، با مقدار اولیه ۱۲ تعریف شده است . در خط دوم نیز ۳ متغیر علامت دار با نام های a و X و j که هر سه مقدار اولیه ۰ دارند تعریف شده است.

نکته : در صورت عدم تعریف مقدار اولیه در هنگام تعریف یک متغیر ، مقدار اولیه در حالت default برابر ۰ تعریف می شود.

ویژگی های نام متغیر

-اولین کاراکتر نام متغیر عدد نمیتواند باشد.

-نام متغیر بیشتر از ۳۱ کاراکتر مورد استفاده نیست.

-نام متغیر تنها ترکیبی از حروف a تا z و A تا Z و اعداد و کاراکتر _ می تواند باشد.

انواع متغیر ها از نظر محل تعریف در برنامه

متغیرها از نظر مکانی که در برنامه تعریف می شوند ، به دو دسته کلی تقسیم می شوند:

1. متغیرهای عمومی(Global)
2. متغیرهای محلی(Local)

متغیر هایی که قبل از تابع main تعریف می شوند را متغیرهای عمومی گویند و در همه جای برنامه می توان به آن دسترسی داشت . اما متغیرهای محلی در بدنه توابع تعریف می شوند و در بیرون از آن تابع ، دسترسی به آن ممکن نیست. در واقع با تعریف یک متغیر عمومی در ابتدای برنامه ، مقدار مشخصی از حافظه برای همیشه به آن متغیر تخصیص می یابد اما متغیرهای محلی تنها در زمان احتیاج تعریف شده و در حافظه می نشینند و بعد از مدتی از حافظه پاک می شوند.

محل تعریف متغیرها در حافظه میکروکنترلر

زمانی که یک متغیر به صورتی که در بالا گفته شد ، تعریف می شود آن متغیر در اولین مکان خالی در حافظه SRAMذخیره می شود . برای تعریف متغیر در حافظه EEPROM از کلمه کلیدی eeprom و برای تعریف متغیر در حافظه FLASH از کلمه کلیدی flash قبل از تعریف متغیر استفاده می شود . بنابراین باید توجه داشت که با قطع منبع تغذیه کلیه حافظه SRAMپاک خواهد شد و متغیرهایی که باید ذخیره دائمی شوند می بایست در حافظه EEPROM یا FLASH تعریف و ذخیره شوند . مثال:

```
int a;

eeprom char b;

flash float c;
```

نکته : حافظه Flash ، حافظه برنامه کاربر است یعنی برنامه به زبان C بعد از کامپایل و ساخته شدن توسط کدویژن و پروگرام شدن روی میکروکنترلر در حافظه Flash ذخیره می شود . بنابراین برای تعریف متغیر در حافظه Flashتنها در صورت خالی بودن قسمتی از آن امکان پذیر است.

نکته : حافظه SRAM تنها با قطع تغذیه پاک می شود و میتوان کاری کرد که با ریست شدن میکرو متغیرهای عمومی موجود در SRAM ریست نشده و مقدار قبلی خود را حفظ نمایند.

تعیین آدرس ذخیره متغیرها در SRAM

به منظور مدیریت بهتر روی حافظه SRAM ، زمانی که یک متغیر را تعریف می کنید ، میتوانید با استفاده از عملگر @ به صورت زیر ، آدرس محل تعریف آن متغیر در حافظه SRAM را تعیین نمایید. استفاده از این عملگر باید هدفمند باشد و آدرس متغیر تنها در بخشی از حافظه SRAM که مخصوص ذخیره داده ها است ، صورت گیرد.

نوع متغیر نام متغیر @ آدرس محل;

ثابت ها در زبانC

ثابت یک مقدار مشخص است که در ابتدای برنامه قبل از main تعریف می شود و یک نام به آن تعلق می گیرد. این مقدار هیچگاه قابل تغییر توسط کدهای برنامه نمی باشد. معمولا مقادیر ثابت عددی که در طول برنامه زیاد تعریف می شود را یک ثابت با نامی مشخص تبدیل می کنند .برای تعریف ثابت می توان به دو روش زیر عمل کرد.

۱. استفاده از دستورConst

مثال:

```
const float pi=3.14;
```

۲. استفاده از دستور#define

مثال:

```
#define pi 3.14
```

تعریف ثابت ها معمولا در ابتدای برنامه با استفاده از دستور #define صورت می گیرد زیرا این دستور پیش پردازنده بوده و به بهبود برنامه کمک می کند . به طور کلی در زبان سی دستوراتی که با #آغاز می شوند پیش پردازنده هستند یعنی کامپایلر ابتدا آنها را پردازش و سپس بقیه برنامه را کامپایل می کند.

توابع در زبانc

تابع یکی از مهمترین بخش های زبان سی می باشد . یک تابع همانند دستگاهی است که مواد اولیه را دریافت می کند و بعد از انجام عملیات مورد نظر روی آنها خروجی مطلوب را تحویل می دهد . توابع در زبان C یا توابع کتابخانه ای هستند یا توابعی هستند که کاربر بر حسب نیاز برنامه خود اضافه می کند .

زبان سی دارای توابعی است که از قبل نوشته شده‌اند، و توابع کتابخانه‌ای نامیده می‌شوند. در واقع فرایندهایی که پر کاربرد هستند و در اغلب برنامه‌ها مورد استفاده قرار می‌گیرند به صورت توابع مستقل قبلاً نوشته شده‌اند و درون فایلهایی قرار داده شده اند . با اضافه کردن فایل های سرآمد که تعریف آن توابع در آنها قرار دارد می توان از آن توابع استفاده کرد.

www.sbargh.ir

تابع اصلی برنامه نویسی به زبان C تابع mainنام دارد که در تمامی برنامه ها وجود داشته و بدون ورودی و خروجی است . توابع دیگر را می توان در بالای تابع main ، در پایین تابع main و یا در فایل های کتابخانه ای تعریف کرد.

انواع توابع در زبان c

هر تابع مجموعه ای از دستورات است که بر روی داده ها پردازش انجام می دهد. ورودی و خروجی یک تابع مقادیری هستند که تابع در برنامه دریافت می کند و به برنامه باز می گرداند. توابع بر اساس ورودی و خروجی به ۴ دسته زیر تقسیم می شود:

۱. **تابع با ورودی ، با خروجی**

مثال : تابع با دو ورودی از جنس کاراکتر و یک خروجی از جنس کاراکتر

```
Char F1( char x , char y );
```

۲. **تابع با ورودی ، بدون خروجی**

مثال : تابع دارای یک ورودی int و بدون خروجی

```
void F2( int x );
```

۳. **تابع بدون ورودی ، با خروجی**

مثال : تابع بدون ورودی اما دارای خروجیint

```
int F3(void);
```

۴. **تابع بدون ورودی ، بدون خروجی**

مثال : تابع بدون ورودی و خروجی

```
void F4(void);
```

بنابراین در اولین قدم باید مشخص کنیم که این تابع چه خروجی را به ما می دهد (در اصطلاح برنامه نویسی بر می گرداند) و فقط به ذکر نوع خروجی بسنده می کنیم ، یعنی مثلا اگر عدد صحیح برگرداند از int ، اگر کاراکتر برگرداند از char و به همین ترتیب برای انواع دیگر و اگر هیچ مقداری را برنگرداند از void استفاده می کنیم.

در قدم بعدی نام تابع را مشخص می کنیم . یک تابع باید دارای یک نام باشد تا در طول برنامه مورد استفاده قرار گیرد .هر نامی را می توان برای توابع انتخاب نمود که از قانون نامگذاری متغیرها تبعیت می کند، اما سعی کنید که از نامهایی مرتبط با عمل تابع استفاده نمایید.

همینطور باید ورودیهای تابع را نیز مشخص کنیم که در اصطلاح برنامه نویسی به این ورودیها، پارامترها یا آرگومان تابع نیز گفته می شود. اگر تابع بیش از یک پارامتر داشته باشد باید آنها را با استفاده از کاما از یکدیگر جدا نماییم و اگر تابع پارامتری نداشت از کلمه voidاستفاده می کنیم. نام پارامتر نیز میتواند هر نام دلخواهی باشد . بخاطر داشته باشید که قبل از نام هر پارامتر باید نوع آنرا مشخص نماییم و بدانیم که کامپایلر هیچ متغیری بدون نوع را قبول نکرده و در صورت برخورد با این مورد از برنامه خطا می گیرد و در نتیجه برنامه را اجرا نخواهد کرد .

بنابراین در زبان c توابع حداکثر دارای یک خروجی می باشند و اگر تابعی خروجی داشته باشد آن را باید با دستور returnبه خروجی تابع و برنامه اصلی بر گردانیم.

تعریف توابع در زبانc

برای نوشتن و اضافه کردن یک تابع باید دقت کرد که هر تابع سه بخش دارد : اعلان تابع ، بدنه تابع و فراخوانی تابع

نحوه تعریف تابع به یکی از سه صورت زیر است:

1. اعلان تابع قبل از تابع main باشد و بدنه تابع بعد از تابع main باشد.
2. اعلان تابع و بدنه تابع هر دو قبل از تابع main باشد.
3. اعلان تابع و بدنه تابع درون فایل کتابخانه ای باشد.

فراخوانی تابع نیز در درون تابع main صورت می گیرد . در صورتی که اعلان و فراخوانی تابع درون فایل کتابخانه ای باشد فقط نیاز به فراخوانی آن در main است.

نکته : هیچ تابعی را نمیتوان درون تابعی دیگر تعریف نمود و فقط به صورت های گفته شده صحیح است اما میتوان از فراخوانی توابع در داخل یکدیگر استفاده کرد بدین صورت که تابعی که داخل تابع دیگر فراخوانی می شود باید در برنامه زودتر تعریف شود.

اعلان و بدنه تابع:

نوع داده خروجی تابع نام تابع (نوع ورودی اول نام ورودی اول , نوع ورودی دوم نام ورودی دوم , ...);

مثال:

```
void sample ( int x, int y );
```

در صورتی که بخواهیم اعلان تابع قبل از main باشد آن را به صورت فوق اعلان می کنیم و بعد از تابع main به صورت زیر دوباره اعلان را به همراه بدنه تابع می نویسیم .

نوع داده خروجی تابع نام تابع (نوع ورودی اول نام ورودی اول , نوع ورودی دوم نام ورودی دوم , ...) {

بدنه تابع : دستوراتی که تابع انجام می دهد }

مثال:

```
void sample ( int x, int y )
{
   .
   .
   .
}
```

در صورتی که بخواهیم قبل از تابع main اعلان و بدنه تابع باشد ، به همان صورت فوق این کار را انجام می دهیم با این تفاوت که یکجا هم اعلان و هم بدنه قبل از main تعریف می شود.

فراخوانی تابع:

صدا زدن تابع درون برنامه را فراخوانی گویند . در فراخوانی توابع باید نام تابع و مقدار ورودی ها را بیان کنیم که به آن آرگومانهای تابع نیز گفته می شود . در مقدار دهی آرگومان تابع در هنگام فراخوانی دیگر نباید نوع آرگومانها را ذکر کنیم. مثال:

```
void main(void)
{
...
   sample (a, b);
...
}
```

در مورد نوع خروجی تابع دو حالت وجود دارد. اول اینکه اگر تابع بدون خروجی باشد لازم نیست از void استفاده کنیم و دوم اینکه اگر تابع دارای خروجی باشد باید آنرا برابر با مقدار متغیر از همان نوع قرار دهیم تا مقدار برگشتی را در متغیر مذکور ریخته و در جای مناسب از آن استفاده نماییم . مثال:

```
#include <mega32.h>
#include <delay.h>


unsigned char i=0;


unsigned char count (void) {
i++;
delay_ms(300);
return i;
}
```


www.sbargh.ir

```
void main (void) {
DDRA=0xff;
PORTA=0x00;
while(1){
PORTA= count();
}
}
```

توضیح : در برنامه فوق ابتدا هدرفایل مربوط به میکروکنترلر Atmega32 به برنامه اضافه می شود . با اضافه شدن این فایل برنامه تمام رجیسترهای میکروکنترلر را می شناسد . سپس هدر فایل delay برای اینکه بتوان از تابع delay_ms استفاده کرد به برنامه اضافه شده است . سپس یک متغیر ۸ بیتی از نوع عدد بدون علامت (unsigned char) با مقدار اولیه صفر تعریف می شود . یک تابع دارای خروجی و بدون ورودی اعلان و بدنه آن تعریف شده است . در تابع main ابتدا رجیستر DDRA به منظور اینکه تمام ۸ بیت موجود در پورت A خروجی شود ، به صورت 0xffمقدار دهی شده است . مقدار اولیه منطق پایه های خروجی پورت A در خط بعدی همگی برابر ۰ شده است . در نهایت در حلقه بی نهایت while ، تابع فراخوانی شده است و مقدار خروجی که تابع برمیگرداند در درون PORTA ریخته می شود . در صورتی که روی هر پایه از پورت LED قرار دهیم ، برنامه به صورت شمارنده عمل خواهد کرد و در هر مرحله تعدادی LED روشن می گردد.

نکته : نوع متغیر رجیسترها در هدرفایل mega32.h همگی به علت ۸ بیتی بودن رجیسترها در میکروکنترلرهای AVR، از نوع unsigned char می باشد.

کلاس های حافظه متغیرها

کلاس حافظه هر متغیر دو چیز اساسی را برای آن متغیر ، تعیین می‌کند:

•مدت حضور یا همان طول عمر (Life Time) آن متغیر

•محدوده قابل دسترسی بودن متغیر در برنامه(Scope)

پس با توجه به این دو مورد که در بالا ذکر شد، ما می‌توانیم برنامه‌هایی را بنویسیم که:

•از منابع حافظه کامپیوتر به خوبی بهره ببرند و بی مورد حافظه اشغال نشود.

•سرعت اجرای بالاتری دارند.

•دچار خطای کمتر و همچنین عیب یابی آسان تری باشند.

۴ نوع کلاس‌های حافظه در زبان C به صورت زیر تعریف شده است:

•اتوماتیک(Automatic)

•خارجی(External)

•استاتیک(Static)

•ثبات(Register)

نحوه تعریف کلاس حافظه یک متغیر

برای تعیین نوع کلاس حافظه برای متغیرها کافی است نام کلاس مورد نظر را در هنگام تعریف متغیر به ابتدای آن اضافه کنیم:

نوع کلاس حافظه   نوع متغیر   نام متغیر = مقدار اولیه ;

که نوع کلاس حافظه با استفاده از کلمات کلیدی auto (برای کلاس حافظه اتوماتیک) ، static (برای کلاس حافظه استاتیک) ، register (برای کلاس حافظه ثبات) و extern (برای کلاس حافظه خارجی) تعیین می‌گردد . به عنوان مثال در کد زیر دو متغیر a و b (با مقدار دهی اولیه ۱۰ برای b) از نوع عدد صحیح تعریف شده اند که کلاس حافظه آن‌ها static می‌باشد.

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

کلاس حافظه اتوماتیک

این کلاس که پر کاربردترین کلاس حافظه هست با کلمه کلیدی auto مشخص می‌شود. اگر نوع کلاس حافظه متغیری را ذکر نکنیم، کامپایلر خود به خود auto در نظر می‌گیرد . (به همین علت در کدویژن کلمه کلیدی auto حذف شده است) متغیرهایی که در داخل توابع تعریف می شوند از این نوع هستند که با فراخوانی تابع به طور اتوماتیک ایجاد می شوند و با پایان یافتن تابع به طور اتوماتیک از بین می روند . این نوع متغیرها دارای خواص زیر هستند:

۱. به صورت محلی (Local) هستند . یعنی در داخل بلاکی که تعریف شده اند، قابل دسترسی اند.
۲. هنگام ورود یک متغیر به یک تابع یا بلاک، به آن حافظه اختصاص داده می‌شود و این حافظه هنگام خروج از تابع یا بلاک، پس گرفته می‌شود.
۳. چندین بار می‌توانند مقدار اولیه بگیرند.

کلاس حافظه ثبات

متغیرهای کلاس حافظه ثبات (register) در صورت امکان در یکی از ثبات‌های CPU (در AVR یکی از ۳۲ رجیستر همه منظوره) قرار می‌گیرند؛ لذا سرعت انجام عملیات با آن‌ها به علت نزدیک بودن و دسترسی مستقیم CPU به آن بسیار بالاست و در نتیجه موجب افزایش سرعت اجرای برنامه می‌شود . معمولا متغیرهای شمارنده حلقه های تکرار را از این نوع تعریف می کنند . این کلاس دارای ویژگی ها و محدودیت‌های زیر است:

۱. همان طور که در بالا ذکر شد، متغیر از نوع ثبات در صورت امکان در یکی از ثبات‌های CPU قرار می‌گیرد . زیرا به دلیل کم بودن تعداد ثبات‌هایCPU ، تعداد محدودی متغیر می‌توانند در ثبات‌ها قرار بگیرند. پس اگر تعداد متغیر هایی که از نوع کلاس حافظه ثبات تعریف شده اند زیاد باشند، کامپایلر کلاس حافظه ثبات را از متغیرها حذف می‌کند.
۲. کلاس حافظه ثبات تنها می‌تواند برای متغیرهای محلی و همچنین پارامترهای تابع به کار گرفته شود.
۳. انواع متغیر که می‌توانند دارای کلاس حافظه ثبات باشند، در کامپیوترهای مختلف، متفاوت است . دلیل این امر هم این است که متغیرهای مختلف، تعداد بایت متفاوتی را به خود اختصاص می‌دهند.
۴. آدرس در مفهوم کلاس حافظه ثبات بی معنی است زیرا متغیرها در ثبات‌های CPU قرار می‌گیرند و نه در RAM. پس در مورد ان کلاس حافظه، نمی‌توان از عملگر &برای اشاره به آدرس متغیرها استفاده کرد.

www.sbargh.ir

مثال :

```
register char a;
```

نکته ۱ : فقط متغیرهایی از جنس int و char را میتوان از نوع کلاس ذخیره سازی ثبات معرفی کرد.

نکته ۲ : با استفاده از دستور پیش پردازنده #pragma regalloc+ هم میتوان کلاس ثبات را تعریف کرد. مثال :

```
#pragma regalloc+
char a;
```

نکته ۳ : کامپایلر ممکن است به طور اتوماتیک یک متغیر را برای افزایش سرعت اجرای برنامه از نوع رجیستر تشخیص دهد حتی اگر از دستورات فوق برای آن متغیر استفاده نشده باشد. برای جلوگیری از چنین حالتی از کلمه کلیدی volatile در قبل از تعریف متغیر استفاده می شود.

کلاس حافظه خارجی

اگر برنامه‌هایی که می نویسیم، طولانی باشند، می‌توانیم آن را به قسمت‌های کوچکتری تقسیم کنیم که به هر قسمت آن واحد (یا همان Unit) گفته می‌شود . اگر بخواهیم که متغیر هایی را که در واحد اصلی تعریف شده اند را در واحدهای فرعی استفاده کنیم و دیگر آنها را دوباره در واحدهای فرعی تعریف نکنیم، می‌توانیم متغیرهای مورد نظر را با استفاده از کلاس حافظه خارجی تعریف کنیم . بدین منظور باید این متغیرها در واحد اصلی به صورت عمومی تعریف شده باشند و در واحد فرعی از کلمه کلیدی extern قبل از تعریف این متغیرها استفاده کنیم.

طول عمر متغیر هایی که از کلاس حافظه extern هستند، از هنگام شروع برنامه تا پایان آن است و همچنین این متغیرها در سراسر برنامه قابل دسترسی هستند . طول عمر آنها برابر با طول عمل بلاک می باشد . دستور extern به کامپایلر اعلام می کند که برای این متغیر ها ، حافظه جدیدی در نظر نگیرد ، بلکه از همان حافظه ای که در جای دیگر برنامه به آن اختصاص یافته استفاده کند. به مثال زیر توجه کنید :

```
static int x,y;
int m,n;
```

```
void main(void)
{…
}
```

متغیرهای x,y در این مثال با کلاس استاتیک و در بالای تابع main و متغیرهای m و n با کلاس حافظه عمومی معرفی شده اند. اگر کل این برنامه در درون یک فایل باشد هیچ تفاوتی بین این دو تعریف وجود ندارد. اما ممکن است یک برنامه بسیار طولانی باشد و مجبور باشیم آن را در چند فایل قرار دهیم در این صورت متغیرهای عمومی و استاتیک که در یک فایل تعریف شده است فقط و فقط در همان فایل قابل استفاده است. اما با استفاده از کلمه extern به کامپایلر اعلام می کنیم که این یک متغیر خارجی و در یک فایل دیگر تعریف شده است. بنابراین کامپایلر برای این نوع متغیرها حافظه جدیدی در نظر نمی گیرد. به مثال زیر توجه کنید :

فایل اول :

```
static int x,y;
int m,n;
void f1(void){
m=5;
n=10;
x=4;
}
void main(void)
{…
}
```

فایل دوم :

```
extern int m,n;
int f2(void){
int x;
```


www.sbargh.ir

```
x=m/n;

return x;

}
```

در مثال بالا برای استفاده از متغیرهای m,n میتوان دستور extern را به کار برد. در این صورت با اجرای برنامه دوم تغییرات متغیرهای m,n همان جایی که تعریف شده است ، ذخیره می شود.

کلاس حافظه استاتیک

این کلاس را می‌توانیم برای دو دسته از متغیرها به صورت زیر به کار ببریم:

- متغیرهای استاتیک محلی
- متغیرهای استاتیک عمومی

متغیرهای استاتیک محلی

متغیرهای استاتیک محلی در توابعی کاربرد دارند که نمی خواهیم مقداری که متغیر موجود در تابع به خود گرفته با خاتمه عملیات تابع پاک شود . بنابراین در توابعی که متغیرها با کلاس حافظه استاتیک معرفی شده باشند ، بعد از خاتمه عملیات تابع ، مقدار نهایی خود را حفظ کرده و در فراخوانی بعدی تابع آن مقدار را به خود می گیرد .
متغیرهای تعریف شده در این کلاس دارای خواص زیر می باشد:

1. فقط در همان تابعی که تعریف شده اند، قابل دسترسی اند.
2. می توانند مقدار اولیه بگیرند و فقط یکبار مقدار دهی اولیه را دریافت می‌کنند.
3. در هنگام خروج از تابع، مقادیر متغیرها، آخرین مقداری خواهد بود که در تابع به آن‌ها اختصاص یافته است و هنگام اجرای دوباره تابع، مقدار اولیه نمی‌گیرند.

مثال :

```
int f1(void){

        int x=0;
```


www.sbargh.ir

```
x++;
return x;
}
```

در تابع f1 متغیر محلی x دارای حافظه اتوماتیک می باشد یعنی با هر بار فراخوانی f1 ، یک واحد به متغیر x اضافه شده و به خروجی تابع بر می گردد. مقدار اولیه x برابر صفر است. بنابراین با هر بار فراخوانی f1 ، متغیر x به مقدار اولیه خود باز می گردد. بنابراین خروجی تابع f1 همواره ۱ است.

```
int f2(void){
    static int y=0;
    y++;
    return y;
    }
```

در تابع f2 متغیر محلی y دارای حافظه استاتیک است که با اولین بار فراخوانی تابع f2 این متغیر بوجود آمده و مقدار صفر می گیرد. با هر بار فراخوانی تابع f2 یک واحد به آن اضافه می گردد و متغیر y به مقدار اولیه خود باز نمی گردد. بنابراین در اولین فراخوانی y=1 و در دومین فراخوانی y=2 و...

متغیرهای استاتیک عمومی

متغیرهای استاتیک عمومی در خارج از توابع تعریف می شوند و از جایی که تعریف می‌شوند، به بعد قابل دسترسی اند . استفاده از متغیر های استاتیک عمومی از یک طرف موجب می شود تا متغیر ها در جایی که به آنها نیاز است تعریف شوند و از طرف دیگر ، فقط توابعی که به آنها نیاز دارند می توانند از آنها استفاده کنند. به مثال زیر توجه کنید :

```
void f1(void);
void f2(void);
```

```
void main(void){
…
f1();
…
f2();
…
}
static int x,y;
void f1(void){ … }
void f2(void){ … }
```

همانطور که مشاهده می شود ، متغیرهای x,y قبل از بدنه توابع f1 و f2 تعریف شده اند. لذا این متغیرها در تابع main قابل دسترسی نیستند ولی توابع f1 وf2 به آنها دسترسی دارند.

دستورات شرطی در زبان C

دستور شرطی if

```
if( شرط )  دستور;
if( شرط )  {دستورات;}
if( شرط ) {دستور ات;} else { دستورات; }
```

دستور شرطی switch

```
Switch( عامل مورد شرط )
{
Case مقدار اول :
کدهایی که در صورت برابر بودن عامل شرط با مقدار اول باید اجرا شود
Break;
Case مقدار دوم :
کدهایی که در صورت برابر بودن عامل شرط با مقدار دوم باید اجرا شود
Break;
.
.
.
Default :
کدهایی که در صورت برابر نبودن عامل شرط با هیچ یک از مقادیر باید اجرا شود
}
```

حلقه های تکرار در زبان C

حلقه با while

در ابتدا شرط حلقه مورد بررسی قرار می گیرد اگر شرط برقرار باشد یکبار کدهای درون حلقه اجرا می شود و دوباره شرط حلقه چک می شود و این روند تا زمانی که شرط برقرار است ادامه می یابد.

while (شرط حلقه)

{ ; کدهایی که تا زمان برقراری شرط حلقه تکرار می شود }

در حلقه while کامپایلر در ابتدای حلقه شرط را بررسی می کند. برای اینکه بار اول حلقه بدون شرط اجرا شود و سپس شرط در آخر بررسی شود از حلقه do…while استفاده می شود.

do{ ; کدهایی که تا زمان برقراری شرط حلقه تکرار می شود }

while (شرط حلقه)

حلقه for

For(مقدار اولیه شمارنده حلقه ; شرط حلقه ; گام حرکت حلقه)

{ ; کدهایی که تا زمان برقراری شرط حلقه تکرار می شود }

مثال :

```
for(i=0;i<10;i++){
PORTA=i;
}
```

دستور breakو continueدر حلقه ها

برنامه با دیدن دستور break در هر نقطه از حلقه ، در همان نقطه از حلقه خارج شده و کدهای زیر حلقه را اجرا می کند . دستور break در حلقه خروج بدون شرط از حلقه است. برنامه با دیدن دستور continueدر هر نقطه از حلقه ، کدهای زیر دستور continue را رها کرده و به ابتدای حلقه باز می گردد.

آرایه ها در C

آرایه اسمی برای چند متغیر هم نوع می باشد یا به عبارت دیگر آرایه از چندین کمیت درست شده است که همگی دارای یک نام می باشد و در خانه های متوالی حافظه ذخیره می گردند. هر یک از این کمیت ها را یک عنصر می گویند، برای دسترسی به عناصر آرایه باید اسم آرایه و شماره ی اندیس آرایه را ذکر کنیم. آرایه ها در زبان سی از جایگاه ویژه ای برخوردارند، به طوری که در پروژه های خود به طور مکرر به آن برخورد خواهید کرد زیرا ارسال و دریافت داده به صورت رشته (آرایه ای از کاراکترها) و سریال انجام می شود.

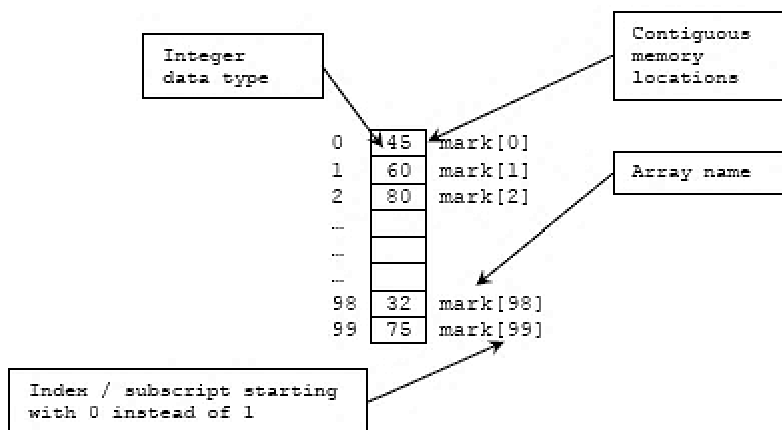
آرایه ها و رشته ها

آرایه مجموعه ای از متغیر های هم نوع است که تحت عنوان یک نام مشخص مورد استفاده قرار می گیرد و به صورت زیر تعریف می شود :

نوع متغیر   نام آرایه [تعداد خانه ها] = { مقدار اول , مقدار دوم , ... };

با تعریف آرایه به همان مقدار خانه های حافظه بسته به نوع متغیر و تعداد خانه های آرایه تخصیص می یابد که در شکل زیر مثالی از آن را مشاهده می کنید . میزان حافظه ای که به آرایه اختصاص داده می شود،به این شکل استفاده می شود:

(طول آرایه) ضرب در (طول نوع آرایه) = میزان حافظه آرایه (برحسب بایت)



برای دسترسی به هر یک از خانه ها آدرس آن را لازم داریم. آدرس هر خانه از 0 تا n-1 است که در آن n تعداد خانه های تعریف شده است . هر عضو آرایه به تنهایی می تواند در محاسبات شرکت کند . مثال:

```
unsigned char a[5]={7,12,0,99,1};

a[0]=a[4]*a[2];
```

نکته ۱ : اندیس آرایه خود می تواند متغیر باشد و این قابلیت می تواند در برنامه ها کاربرد زیادی داشته باشد.

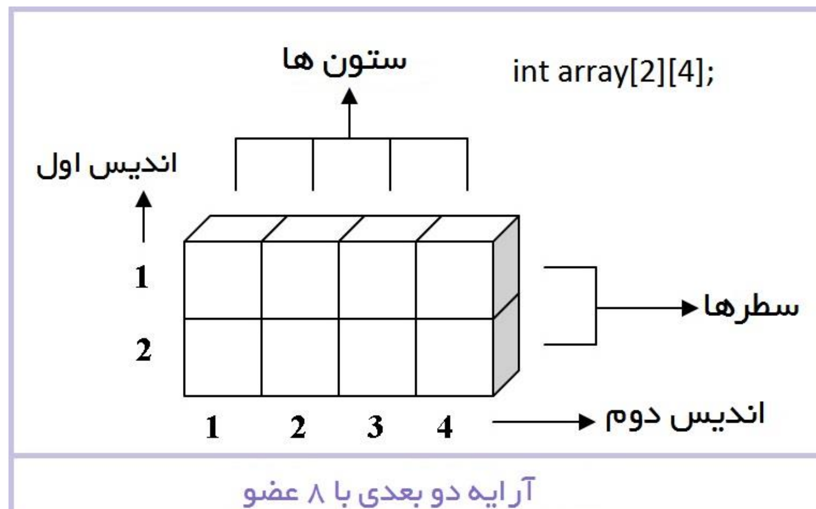
نکته ۲ : در صورتی که می خواهید آرایه به صورت دائمی ذخیره شود (مثلا وقتی که میخواهید لوگو شرکت خود را همیشه داشته باشید) باید به ابتدای تعریف کلمه کلیدی eeprom یا flash را اضافه کنید تا در حافظه های دائمی ذخیره گردد.

نکته ۳ : در صورتی که اندیس آرایه را ننویسیم ، یک آرایه با طول اتوماتیک بوجود می آید. یعنی به تعدادی که در ابتدای برنامه آرایه مقدار می گیرد ، به همان اندازه از حافظه مصرف می شود. مثال :

```
int a[]={1,2,3,4,5,6};
```

آرایه های چند بعدی

در تعریف آرایه دو بعدی باید ۲ اندیس و در تعریف آرایه سه بعدی باید ۳ اندیس و در تعریف آرایه n بعدی باید n اندیس را ذکر کرد .



آرایه دو بعدی با ۸ عضو

www.sbargh.ir

آرایه های چند بعدی در نمایشگر های lcd کاربرد دارند . به عنوان مثال:

```
int table [10] [10];
```

یک آرایه دو بعدی بنام table را تعریف میکند که دارای ۱۰ سطر و ۱۰ ستون است و نوع عناصر آن int است.

```
int k [5] [10] [15];
```

آرایه ای سه بعدی بنام k را تعریف می کند که دارای ۵ سطر ، ۱۰ ستون و ۱۵ ارتفاع است و نوع عناصر آن int می باشد.

نکته : تعریف آرایه ها با مجموعه عناصر زیاد در حافظه SRAM به علت محدودیت در حجم حافظه ممکن است باعث ایجاد مشکل شود. بنابراین معمولا آرایه های با حجم زیاد را در بخش خالی حافظه flash ذخیره می کنند.

مقدار دهی به آرایه های چند بعدی

برای مقدار دهی به آرایه های دو بعدی سطر ها را به ترتیب پر می کنیم . مثال:

```
Int a[2][3]={ {3,1,2} , {7,4,6} }
```

برای مقدار دهی به آرایه های سه بعدی ابتدا سطرهای طبقه اول و سپس سطرهای بقیه طبقات را مقدار دهی می کنیم . مثال:

```
Int a[2][3][4]={ { {1,2,3,4} ,{5,4,3,2} , {2,3,4,5} } , { {7,6,5,4} , {9,0,8,7} , {6,7,4,1}
} }
```

رشته ها

در زبان سی برای نمایش کلمات و جملات از رشته ها استفاده می شود . رشته همان آرایه ای از کاراکتر ها است که حاوی اطلاعاتی می باشد . تمام کاراکتر ها شامل اعداد و حروف و برخی کاراکترهای دیگر که روی صفحه کلید کامپیوتر وجود دارند ، دارای کد شناسایی ASCII (American Standard Code For Information Interchange) می باشند . کدهای اسکی که توسط استاندارد آمریکایی در سال ۱۹۶۷ ابداع شد و در سال ۱۹۸۶ دست خوش تغییراتی شد.

کاراکتر ست اسکی خود به دو نوع تقسیم می‌شود. نوع ۷ بیتی که با نام اسکی استاندارد (Standard ASCII) شناخته شده و دارای ۲ به توان ۷ یعنی ۱۲۸ کاراکتر مختلف است که از ۰ تا ۱۲۷ استفاده می‌شوند. نوع دیگر آن حالت ۸ بیتی است که با نام اسکی توسعه یافته (Extended ASCII)شناخته شده و دارای ۲ به توان ۸ یعنی ۲۵۶ کاراکتر مختلف است که از ۰ تا ۲۵۵ استفاده می‌شود. حالت توسعه یافته جدا از حالت استاندارد نیست بلکه از ۰ تا ۱۲۷ کاراکتر اول آن درست مانند حالت استاندارد بوده و فقط بقیه کاراکترها (از ۱۲۸ تا ۲۵۵) به آن اضافه شده است .کاراکترهای اضافی دارای هیچ استانداردی نبوده و ممکن است در دستگاه‌ها و کامپیوترهای مختلف فرق داشته باشد و به منظور ایجاد کاراکترهای زبان دوم (مثلا زبان فارسی) ایجاد شده است . یعنی ممکن است در یک کامپیوتر کاراکتر اسکی ۱۵۰ معادل حرف û و در کامپیوتر دیگر که روی زبان دوم فارسی تنظیم شده است ، معادل حرف ب باشد . اما کاراکترهای قبل از ۱۲۸ همگی ثابت هستند .کاراکترهای فارسی در اینکدینگ Iranian Systemشرکت ایرانیان سیستم که یکی از قدیمی ترین اینکدینگ‌های ASCII فارسی است را می‌توانید در این لینک ببینید.

در هر دو نوع ذکر شده (۷ و ۸ بیتی) تعداد ۳۲ کاراکتر اول (یعنی از ۰ تا ۳۱) و آخرین کاراکتر (۱۲۷) با عنوان کاراکترهای کنترلی (Control Characters) شناخته می‌شود. این کاراکترها غیرقابل چاپ بوده و فقط برای کنترل متن مورد استفاده قرار می‌گیرد (مثلاً مشخص کننده ابتدای هدر، حذف، کنسل و ...). بقیه کاراکترها یعنی از ۳۲ تا ۱۲۶ قابل چاپ هستند. این کاراکترها شامل نمادها، حروف و اعداد انگلیسی هستند .(در حالت توسعه یافته، از کاراکترهای ۱۲۸ تا ۲۵۵ نیز قابل چاپ هستند.)

در شکل زیر این ۹۵ کاراکتر قابل چاپ انگلیسی را به همراه کد اسکی آن در مبنای دسیمال مشاهده می کنید.

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
32	[space]	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	[backspace]

تعریف یک کاراکتر

تعریف یک کاراکتر توسط نوع متغیر char صورت می گیرد و مقدار اولیه آن داخل کوتیشن (' ') قرار می گیرد . در زبان سی وقتی یک حرف بین ' و ' قرار می گیرد کد اسکی آن درون متغیر ذخیره می شود . مثال:

```
char c='H';
```

تعریف رشته (آرایه ای از کاراکتر ها)

برای تعریف رشته از آرایه ای از کاراکترها استفاده می شود و مقدار اولیه آن داخل دابل کوتیشن (" ") قرار می گیرد .مثال:

```
char s[10]="Hello!";
```

اگر تعداد خانه های آرایه ذکر شود ، آرایه سایز مشخصی دارد و در صورتی که تعداد کاراکترهای عبارت یا جمله ای که درون آن میریزیم بیشتر از سایز آرایه باشد ، عبارت ناقص ذخیره خواهد شد و در صورتی که تعداد کاراکترهای مورد نظر کمتر باشد بقیه آرایه خالی خواهد بود . اگر تعداد خانه های آرایه ذکر نشود یعنی سایز آرایه بر اساس مقداری که درون آن ریخته می شود محاسبه شود . مثال:

```
char str[]="Hello!..."
```

عملگرها

با استفاده از عملگرها می توان روی اعداد ، متغیرها ، آرایه ها ، رشته ها و ... عملیات حسابی ، منطقی ، مقایسه ، بیتی ، بایتی و ... انجام داد.

عملگرهای محاسباتی

عملگرهای محاسباتی، عملگرهایی هستند که اعمال محاسباتی را روی عملوندها انجام می‌دهند. عملگر ٪ برای محاسبه باقی‌مانده تقسیم به کار می‌رود. این عملگر عملوند اول را بر عملوند دوم تقسیم می‌کند (تقسیم صحیح) و باقیمانده را برمی‌گرداند. در جدول‌های زیر، عملگرهای محاسباتی و تقدم آن‌ها در یک معادله مشاهده می‌شود:

عملگر	نام	مثال
-	تفریق و علامت منفی	z = x - y یا x -
+	جمع	z = x + y
*	ضرب	z = x * y
/	تقسیم	z = x / y
%	باقیمانده تقسیم	z = x % y
--	یک واحد کاهش	x-- یا x --
++	یک واحد افزایش	x++ یا x ++

تقدم	عملگر
۱	++ و --
۲	- علامت منفی
۳	/ و * و %
۴	+ و -

دو عملگر ++ و-- همان‌طور که مشاهده می‌کنید در طرف چپ و راست متغیر قرارگرفته‌اند. اگر به‌تنهایی و در یک خط دستور به‌کاربرده شوند، فرقی نمی‌کند اما اگر در یک معادله به‌کاربرده شوند، اینکه طرف راست یا چپ قرار گیرند، متفاوت است. به مثال زیر توجه کنید:

```
x++;

++x;

y = ++x;

y = x++;
```

همان‌طور که در مثال مشاهده می‌کنید، در دو دستور اول به متغیر x یک واحد اضافه می‌شود. در دستور سوم، ابتدا یک واحد به مقدار متغیر x اضافه‌شده و سپس درون متغیر y قرار می‌گیرد. در دستور چهارم، ابتدا مقدار متغیر x درون متغیر y قرارگرفته و سپس یک واحد به آن اضافه می‌شود. اگر تا قبل از رسیدن به دستور چهارم، مقدار x برابر با ۱۰ باشد، پس از گذشتن از دستور چهارم، مقدار y برابر ۱۰ و x برابر با ۱۱ است.

عملگرهای مقایسه‌ای و منطقی

عملگرهای مقایسه‌ای ارتباط بین عملوند ها را مشخص می کنند و عملگر های منطقی بر روی عبارات منطقی عمل می کنند. عبارات منطقی دارای دو ارزش درستی و نادرستی اند و زمانی که باید یک شرط مورد بررسی قرار بگیرد، استفاده می شوند. به طور مثال برای بررسی مساوی بودن دو متغیر از عملگر == استفاده می شود. در زبانc ، ارزش نادرستی با ۰ و ارزش درستی با مقادیر غیر صفر مشخص می شود.

عملگر	نام	مثال
>	بزرگتر	x > y
>=	بزرگتر مساوی	x >= y
<	کوچکتر	x < y
<=	کوچکتر مساوی	x <= y
==	متساوی	x == y
!=	نا مساوی	x != y
!	نقیض	!(x > y)
&&	و	x > y && z > w
||	یا	x > y || z > w

تقدم	عملگر
۱	!
۲	> و >= و < و <=
۳	!= و ==
۴	&&
۵	||

عملگرهای ترکیبی

این عملگر ها ترکیبی از عملگر مساوی و عملگرهای دیگر هستند. به طور مثال عملگر =+ در نظر بگیرید، اگر به شکل x += a نوشته شود، برابر با x = x+a است؛ یعنی هر بار متغیر x را با متغیر a جمع می کند و درون متغیر x قرار می‌دهد. برای دیگر عملگر های ترکیبی نیز به همین صورت است. این عملگر ها پایین ترین تقدم را در بین عملگرهای دیگر دارند. این عملگر ها، عملگر های حسابی را شامل می شوند. در جداول زیر عملگرهای ترکیبی و تقدم آنها را میبینید.

عملگر	نام	مثال
+=	انتساب جمع	x += y
-=	انتساب تفریق	x -= y
*=	انتساب ضرب	x *= y
/=	انتساب تقسیم	x /= y
%=	انتساب باقیمانده تقسیم	x %= y

تقدم	عملگر
۱	/= و *= و %=
۲	+= و -=

تعریف عملگرهای بیتی

عملگر های بیتی بر روی هر بیت یک بایت اثر می گذارند . به طور مثال ، aیک متغیر ۸ بیتی با مقدار باینری ۱۱۱۰۰۱۰۱ است ، اگر عملگر not را روی a اثر دهیم ، نتیجه معکوس شدن هر بیت هست a =~ a . برابر با مقدار باینری ۰۰۰۱۱۰۱۰ است . عملگرهای دیگر نیز به همین شکل اثر خود را اعمال می‌کنند.

x	y	x & y	x | y	x ^ y	~x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

جدول صحت عملگر های بیتی

در جدول‌های زیر عملگرهای بیتی و تقدم آن‌ها را مشاهده می‌کنید:

عملگر	نام	مثال
&	And	x = 11101101 , y = 00001111 → x & y = 00001101
|	Or	x = 11101101 , y = 00001111 → x | y = 11101111
^	Xor	x = 11101101 , y = 00001111 → x ^ y = 11100010
~	Not	x = 00001111 → x = ~ x → x = 11110000
<<	شیفت به چپ	x = 00000111 → x = x << 2 → x = 00011100
>>	شیفت به راست	x = 11000000 → x = x >> 3 → x = 00011000

تقدم عملگرهای بیتی

تقدم	عملگر
۱	~
۲	<< و >>
۳	&
۴	^
۵	|

تقدم کلی در عملگرها

تقدم	عملگر	تقدم	عملگر
۸	&	۱	()
۹	^	۲	! و ~ و ++ و -- و sizeof
۱۰	|	۳	/ و * و %
۱۱	&&	۴	+ و -
۱۲	||	۵	<< و >>
۱۳	?	۶	> و >= و < و <=
۱۴	/= و *= و %= و -= و +=	۷	!= و ==

www.sbargh.ir

همانطور که مشاهده می شود عملگر پرانتز دارای بیشترین اولویت است . با استفاده از این عملگر میتوان انجام محاسبات را اولویت داد به طوری که اولویت اول همیشه با داخلی ترین پرانتز است و سپس به ترتیب تا پرانتز بیرونی اولویت دارند. مثال :

```
y=(x+(a/(3+g)))*2;
```

همان‌طور که در مثال بالا مشاهده می‌کنید، ابتدا g+3 انجام می‌شود، در مرحله بعد a/(3+g) انجام می‌شود، در مرحله بعدی x+(a/(3+g)) انجام می‌شود و در انتها (x+(a/(3+g)))*2 انجام می‌شود.

تبدیل نوع در محاسبات

برای درک بهتر در مورد تبدیل نوع در محاسبات ، به مثال زیر توجه کنید.

```
int a=9,b=2;
float x,y,z;
x=a/b;
y=(float) a/b;
z=9.0/2.0;
```

در مثال بالا با اینکه متغیرهای x و y و z هر سه از نوع float هستند ولی مقدار x برابر با ۴ و مقدار y و z برابر با ۴،۵ است. اگر متغیر های a و b از نوع float بودند ، مقدار x نیز برابر ۴،۵ می شد اما چون متغیر های a و b از نوع int هستند باید یک تبدیل نوع در محاسبه توسط عبارت (float) در ابتدای محاسبه انجام شود.

انواع دستورات پیش‌پردازش

یکی از امکانات زبان C فرمان‌های پیش ترجمه یا پیش پردازش است. به‌طوری‌که از عنوان آن مشخص است این فرمان‌ها قبل از شروع ترجمه‌ی برنامه و در یک مرحله‌ی مقدماتی بررسی‌شده، نتیجه آن روی متن برنامه اعمال می‌گردد .استفاده از این فرمان‌ها ازیک‌طرف باعث سهولت برنامه‌نویسی شده، از طرف دیگر باعث بالا رفتن قابلیت

اصلاح و جابه‌جایی برنامه می‌گردد. حال فرض کنید که یک ثابت در چندین جای برنامه‌ی شما ظاهر شود، استفاده از یک نام نمادین برای ثابت ایده‌ی خوبی به نظر می‌رسد. سپس می‌توان به‌جای قرار دادن ثابت در طول برنامه از نام نمادین استفاده کرد.

درصورتی‌که لازم باشد تا مقدار ثابت را تغییر دهیم، بدون استفاده از نام نمادین، باید همه جای برنامه را با دقت برای یافتن و جایگزین کردن ثابت نگاه کنیم، آیا می‌توانیم این کار را در C انجام دهیم؟!

Cقابلیت ویژه‌ای به نام پیش پردازشگر دارد که امکان تعریف و مرتبط ساختن نام‌های نمادین را با ثابت‌ها فراهم می‌آورد. پیش پردازشگر C پیش از کامپایل شدن اجرا می‌شود.

پیش پردازنده نوعی مترجم است که دستورات توسعه‌یافته‌ای از یک‌زبان را به دستورات قابل‌فهم برای کامپایلر همان زبان تبدیل می‌کند. قبل از اینکه برنامه کامپایل شود، پیش پردازنده اجرا می‌شود و دستورات را که با نماد # شروع‌شده‌اند را ترجمه می‌کند. سپس کامپایلر برنامه را کامپایل می‌کند

.دستورات پیش پردازنده در زبان سی شامل موارد زیر هستند:

دستور پیش پردازشی	توضیحات
#include	اضافه کردن فایل کتابخانه
#define	تعریف یک ماکرو
#undef	حذف یک ماکرو
#ifdef	اگر یک ماکرو تعریف شده است آنگاه
#ifndef	اگر یک ماکرو تعریف نشده است آنگاه
#if	اگر شرط برقرار بود آنگاه
#endif	انتهای بلوک if
#else	در غیر این صورت
#error	ایجاد یک خطا در روند کامپایل برنامه
#pragma	تغییر رفتا کامپایلر

نکته‌ای که در مورد پیش پردازشگر باید در نظر داشت این است که پیش پردازشگر C مبتنی برخط است. هر دستور ماکرو با یک کاراکتر خط جدید به پایان می‌رسد، نه با سمیکالن.

پیش پردازنده define

دستور defineرایج‌ترین دستور پیش پردازشی است که به آن ماکرو می گویند. این دستور هر مورد از رشته کاراکتری خاصی (که نام ماکرو هست) را با مقدار مشخص‌شده‌ای (که بدنه ماکرو هست) جایگزین می کند. نام ماکرو همانند نام یک متغیر در C است که باید با رشته تعریف‌کننده ماکرو حداقل یک فاصله داشته باشد و بهتر است جهت مشخص بودن در برنامه با حروف بزرگ نمایش داده شود. به‌عنوان‌مثال دستور #define MAX 20 موجب می‌شود تا مقدار ماکروی MAX در سرتاسر برنامه برابر با ۲۰ فرض شود؛ یعنی در هر جای برنامه از ماکروی MAX استفاده شود مثل این است که از عدد ۲۰ استفاده‌شده است.

کاربرد دیگر دستور defineدر تعریف ماکروهایی است که دارای پارامتر باشند. این مورد به‌صورت زیر استفاده می‌شود:

#define <نام ماکرو> (اسامی پارامترها) تعریف ماکرو

تعریف ماکرو مشخص می‌کند که چه عملی باید توسط ماکرو انجام گیرد. اسامی پارامترها متغیرهایی هستند که در حین اجرای ماکرو به آن منتقل می‌شوند که اگر تعداد آن‌ها بیشتر از یکی باشد، با کاما از هم جدا می شوند. از دستور undefجهت حذف ماکرویی که پیش‌تر تعریف‌شده است استفاده می‌کنیم. روش استفاده از این دستور به‌صورت زیر است:

#undef <نام ماکرو>

در اینجا نام ماکرو شناسه‌ای است که پیش‌تر توسط دستور defineتعریف‌شده است.

پیش پردازنده include

ضمیمه کردن فایل‌ها توسط دستور پیش پردازنده #include انجام می‌گیرد. این دستور به‌صورت های زیر مورد استفاده قرار می‌گیرد:

#include "نام فایل"

#include <نام فایل>

فایل‌های سرآیند به دودسته تقسیم می‌شوند:

۱. هدر فایل‌هایی که همراه کامپایلر C وجود دارند و پسوند همه‌ی آن‌ها h. است.

۲. هدر فایل‌هایی که توسط برنامه‌نویس نوشته می‌شوند.

روش اول دستور include# برای ضمیمه کردن فایل‌هایی استفاده می‌شود که توسط برنامه‌نویس نوشته‌شده‌اند و روش دوم برای ضمیمه فایل‌هایی استفاده می‌شوند که همراه کامپایلر وجود دارند.

```
#include <stdio.h>

#include "myheader.h"
```

فایل‌های سرآیند از اهمیت ویژه‌ای برخوردارند، زیرا :

۱. بسیاری از توابع مثلgetchar وputchar در فایل‌های سرآمد مربوط به سیستم، به‌صورت ماکرو تعریف‌شده‌اند.

۲. با فایل‌های سرآیندی که برنامه‌نویس می‌نویسد، علاوه بر تعریف ماکروها، می‌توان از بسیاری از تعاریف تکراری جلوگیری کرد.

دستورات پیش‌پردازش شرطی

در حالت معمولی، دستور if برای تصمیم‌گیری در نقاط مختلف به کار می‌رود .شرط‌هایی که در دستور if ذکر می‌شوند در حین اجرای برنامه ارزشیابی می‌شوند؛ یعنی اگر شرط ذکرشده در دستور if درست باشد یا نادرست، این دستور و کلیه دستورات دیگر که در بلاک if قرار دارند ترجمه می‌شوند ولی در دستورات پیش پردازنده شرطی، شرطی که در آن ذکر می‌شود در حین ترجمه ارزشیابی می‌شود. دستورات پیش پردازنده شرطی عبارت‌اند از : #if,#else,#endif, #ifdef, #ifndef

دستور if به‌صورت زیر به کار می‌رود :

#if عبارت شرطی

مجموعه ی دستورات۱

#else

مجموعه دستورات۲

www.sbargh.ir

```
#endif
```

در این دستور در صورتی که عبارت شرطی بعد از if برقرار باشد ، مجموعه دستورات ۱ و در غیر این صورت مجموعه دستورات ۲ کامپایل می شود.

برخلاف دستور if درC ، حکم‌های تحت کنترل دستور #ifdef در آکولادها محصور نمی‌گردند. به‌جای آکولادها برای پایان بخشیدن به بلوک #ifdef باید از دستور #endif استفاده شود.

```
#ifdef    DEBUG
  /* Your debugging statements here */
#endif
```

دستور #ifndef عکس دستور #ifdef عمل می‌کند. اگر ماکرویی که نام آن در جلوی #ifndef قرار دارد در یک دستور #define تعریف‌نشده باشد، مجموعه حکم‌های ذکرشده کامپایل می‌گردند، در غیر این صورت کامپایل نخواهند شد. قالب کلی استفاده شبیه #ifdef است:

```
#ifdef <نام ماکرو>
   مجموعه ی دستورات
   .
   .
   .
#endif
```

دستور #ifndef هم برای پایان به دستور #endif نیاز دارد.

```
#ifndef MESSAGE
  #define MESSAGE "You wish!"
#endif
```

دستور #error موجب جلوگیری از ادامه ترجمه‌ی برنامه‌ی توسط کامپایلر شده، به‌صورت زیر به کار می‌رود:

```
#error    پیام خطا
```

پیام خطا، جمله‌ای است که کامپایلر پس از رسیدن به این دستور، آن را به‌صورت زیر در صفحه‌نمایش ظاهر می‌کند:

error: نام فایل شماره خط

پیش پردازنده pragma

این دستور به کامپایلر این امکان را می دهد که ماکروهای مورد نظر را بدون مداخله با کامپایلرهای دیگر تولید کند. به صورت کلی از این ماکرو به صورت زیر استفاده می شود :

#pragma name

که در آن name میتواند یکی از حالت های زیر باشد :

۱. #pragma warn- : غیرفعال کردن اخطارهای صادر شده از کامپایلر
۲. #pragma warn+ : فعال کردن اخطارهای صادر شده از کامپایلر
۳. #pragma opt- : بهینه کننده کد توسط کامپایلر را غیر فعال می کند
۴. #pragma opt+ : بهینه کننده کد توسط کامپایلر را فعال می کند
۵. #pragma optsize- : بهینه کننده برنامه نسبت به سرعت
۶. #pragma optsize+ : بهینه کننده برنامه نسبت به حجم

نکته : معادل دستور optsize را میتوانید از طریق نرم افزار کدویژن در آدرس زیر تنظیم نمایید :

Project/Configuration/C Compiler/Codevision/Optimized for

۷. #pragma savereg+ : این دستور هنگامی که وقفه ای رخ دهد ، میتواند رجیسترهای R0,R1,R22,R23,R24,R25,R26,R27,R30,R31 و SREG را ذخیره نماید.
۸. #pragma savereg- : این دستور مخالف دستور قبلی است و رجیسترها را پاک می کند.
۹. #pragma regalloc+ : این دستور متغیرهای سراسری را به رجیسترها اختصاص می دهد.(معادل کلمه کلیدی register)
۱۰. #pragma regalloc- : این دستور برخلاف دستور قبلی متغیر سراسری را در حافظه SRAM تعریف می کند.

نکته : معادل دستور regalloc را میتوانید از طریق نرم افزار کدویژن در آدرس زیر تنظیم کنید :

Project/Configuration/C Compiler/Code Generation/Automatic Register Allocation

۱۱. #pragma promoteacher+ : این دستور متغیرهای char را به int تبدیل می کند.

۱۲. #pragma promoteacher- : این دستور متغیر های int را به char تبدیل می کند.

نکته : معادل دستور promoteacher را میتوانید از طریق نرم افزار کدویژن در آدرس زیر تنظیم کنید :

Project/Configuration/C Compiler/Code Generation/Promote char to int

۱۳. #pragma unchar+ : این دستور نوع داده char را به unsigned char تبدیل می کند.

۱۴. #pragma unchar- : این دستور نوع داده char را بدون تغییر و به همان صورت signed char تعریف می کند.

نکته : معادل دستور unchar را میتوانید از طریق نرم افزار کدویژن در آدرس زیر تنظیم کنید :

Project/Configuration/C Compiler/Code Generation/Char is unsigned

۱۵. #pragma library این دستور یک فایل کتابخانه ای با پسوند lib. را به برنامه پیوند میزند.

نحوه ساخت فایل های کتابخانه

تقسیم برنامه های بزرگ به واحد های کوچکتر یا اصطلاحا ماژولار کردن برنامه ، از جهات مختلفی ، بسیار سودمند است. به خوانایی برنامه کمک زیادی می کند. برنامه می تواند توسط چندین نفر نوشته شود ، تغییرات در برنامه به سهولت انجام می شود و از هر ماژول در پروژه های دیگر میتوان بهره گرفت.

هر فایل کتابخانه شامل دو فایل است :

۱. فایل هدر (header) : این فایل که با پسوند h. است حاوی الگوی توابع و پیش پردازنده ها (ماکرو) است.
۲. فایل سورس (source) : این فایل که با پسوند c. است حاوی بدنه توابعی است که الگوی آن در هدر فایل تعریف شده است.

این دو فایل در کنار هم درون پوشه پروژه اصلی قرار می گیرند. هدر فایل هایی که تنها شامل عملگرهای پیش پردازنده هستند ، نیازی به فایل سورس ندارند. در هر کامپایلر یا با هر ویرایشگر متنی (مثل Notepad) میتوان این دو فایل را ایجاد کرد و با پسوند مربوطه ذخیره کرد.

www.sbargh.ir

نحوه استفاده از کتابخانه در برنامه

برای استفاده از کتابخانه ای که خود ساخته ایم ابتدا باید هدر فایل آن را بوسیله " به برنامه اضافه کنیم. به صورت زیر :

```
#include "headerfile.h"
```

بعد از اضافه کردن هدر فایل میتوانیم از ثوابت و توابعی که درون کتابخانه تعریف کرده ایم در برنامه اصلی استفاده کنیم.

الگوی ساخت فایل هدر

برای ساخت هدر فایل با پسوند h. باید الگویی را رعایت نمود. برای مثال می خواهیم یک کتابخانه برای اتصال keypad به میکرو ایجاد کنیم. برای این کار ابتدا یک فایل با پسوند h. ساخته و سپس درون آن به صورت الگوی زیر می نویسیم :

```
#ifndef _KEYPAD_H
#define _KEYPAD_H
#include<headerfiles.h>
#define …
…
void Functions(void);
…
#endif
```

همانطور که مشاهده می کنید الگوی نوشتن هدر فایل به این صورت است که در ابتدا با استفاده از ماکروی ifndef/endif و سپس نوشتن نام هدر فایل با حروف بزرگ و دقیقا به همان الگوی مثال زده شده (یک "_" در ابتدا و یک "_" به جای نقطه) و استفاده از ماکروی #define یک ماکرو جدید با نام _KEYPAD_H تعریف

کردیم. این گونه نوشتن را محافظت از برنامه یا Header Guard گویند. هدرگارد باعث میش شود تا ماکروها و توابع تعریف شده فقط و فقط یکبار تعریف شده باشند (جلوگیری از تعریف آنها با نام یکسان) .

سپس اگر هدر فایل به کتابخانه های دیگری نیاز دارد ، آنها را با **#include** اضافه می کنیم. بعد از آن تعریف ثوابت را با استفاده از **#define** انجام می دهیم. اگر تغیر نوع در متغیر ها وجود دارد آنها را بعد از ثوابت **typedef** می کنیم. در پایان الگوی تعریف توابعی که میخواهیم آنها را در فایل سورس تشریح کنیم را باید در این قسمت بیاوریم.

الگوی ساخت فایل سورس

فایل سورس نیز دارای الگویی است که باید رعایت شود. برای ساخت فایل سورس برای **keypad.h** به صورت زیر عمل می کنیم :

```
#include "keypad.h"

void Functions(void){

بدنه تابع

}

...
```

همانطور که مشاهده می کنید ، در ابتدای هر فایل سورس ، فایل هدر **include** می شود و در خطوط بعدی تنها بدنه توابع طبق قوانین مربوط به توابع آورده می شود.

نوع داده sfrb و sfrw درکدویژن

در کامپایلر کدویژن این دو نوع داده ای برای دستیابی به رجیسترهای I/O موجود در حافظه SRAM میکرو اضافه شده است. در حقیقت با تعریف این دو نوع داده در هدر فایل میکروکنترلر (برای مثال هدر فایل mega32.h) قابلیت دسترسی راحت بیتی به رجیسترهای I/O برای کاربر فراهم شده است. بنابراین اگر فایل h. هر میکرویی را باز نمایید ، درون آن این نوع داده ای را مشاهده می کنید. نحوه تعریف آن به صورت زیر است :

```
sfrb    نام رجیستر=آدرس رجیستر;

sfrw    نام رجیستر=آدرس رجیستر;
```

در سمت راست تساوی آدرس رجیستر مورد نظر در حافظه SRAM و در سمت چپ تساوی نام دلخواهی را وارد می کنیم. مثال :

```
sfrb PORTA=0x1b;

sfrb DDRA=0x18;

...
```

همه این تعاریف مربوط به پورت ها و رجیسترهای میکروکنترلرهای AVR در هدر فایل هر یک موجود است که با include کردن آن به برنامه این رجیسترها اضافه می شود و نیازی به تعریف مجدد در برنامه نیست. بنابراین تنها آنچه که در برنامه اصلی از آن استفاده می شود ، استفاده از عملگر نقطه (dot) برای دسترسی بیتی به رجیسترهاست که به صورت زیر است :

نام رجیستر . n = مقدار;

که در آن n برای رجیسترهای تعریف شده با sfrb بین ۰ تا ۸ و برای sfrw بین ۰ تا ۱۵ است.

نکته ۱ : آدرس رجیسترهای حافظه SRAM برای هر میکروکنترلری در انتهای دیتاشیت آن آورده شده است.

نکته ۲ : تفاوت بین sfrb,sfrw در این است که از sfrb برای دسترسی بیتی به رجیسترهای ۸ بیتی و از sfrw برای دسترسی بیتی به رجیسترهای ۱۶ بیتی استفاده می شود.

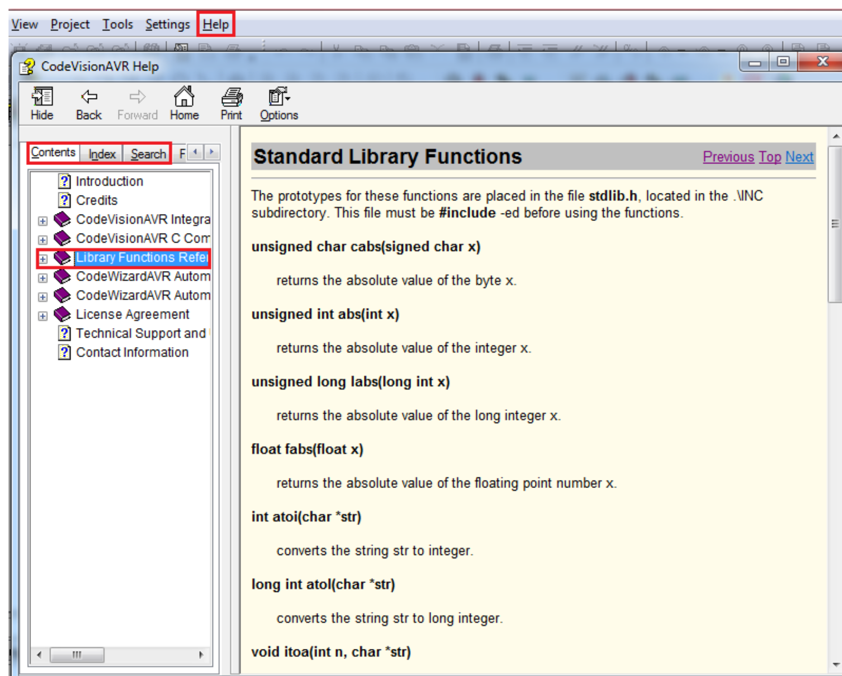
نکته ۳ : در معماری میکروکنترلرهای AVR تنها به رجیسترهایی که در آدرس 0 تا 1FH قرار دارند میتوان دسترسی بیتی داشت. بنابراین این محدودیت برای دستور sfrb و sfrw نیز برقرار خواهد بود.

توابع پرکاربرد در زبان C

برای کاربردهای مختلف و کار با واحدهای میکروکنترلر ، توابعی مورد نیاز است که در کتابخانه های از پیش تعریف شده وجود دارد. از این توابع در نمایشگرهای LCD ، کار با واحدهای USART,I2C,SPI و ... استفاده می شود. این توابع هر کدام هدر فایل هایی دارد که میتوانید برای دسترسی به آن به قسمت help نرم افزار codevision مراجعه نمایید. برای این منظور در صفحه اصلی برنامه و از منوی help به قسمت help topic بروید. در پنجره باز

www.sbargh.ir

شده میتوانید به سه صورت کتابخانه و توابع مورد نیاز خود را پیدا کنید : یکی از طریق جستجو در بخش content و زیر بخش Library Function Refference . دوم از جستجو در بخش index و سوم از طریق search نام مورد نظرتان.



توابع پرکاربرد کتابخانه stdio.h

۱. تابع getchar

این تابع بدون ورودی و دارای خروجی از نوع char می باشد . با نوشتن دستور زیر تابع منتظر می ماند تا یک کاراکتر توسط USART دریافت شود و سپس مقدار دریافت شده را به داخل یک کاراکتر از قبل تعریف شده باز می گرداند . توجه شود تا زمانی که داده از USART وارد نشده باشد برنامه منتظر می ماند و هیچ کاری انجام نمی دهد.

نام کاراکتر=getchar();

۲. تابع putchar

این تابع که دارای یک ورودی از نوع char و بدون خروجی می باشد ، کاراکتر ورودی را توسط USART ارسال می کند.

putchar(‘ کاراکتر ‘);

www.sbargh.ir

۳. توابع puts و putsf

برای ارسال یا دریافت یک رشته به صورت کامل به وسیله USART از دستورات putsf و puts استفاده می شود. تفاوت puts با putsf در این است که تابع puts رشته ی موجود در SRAM را و putsf رشته ذخیره شده در فلش را ارسال می کند.

```
putsf(" رشته ");

puts( متغیر رشته ای );
```

نکته : برای مقدار دهی به یک متغیر رشته ای از تابع sprintf استفاده می شود.

۴. تابع gets

برای دریافت رشته ها از واحد USART و ذخیره آنها روی یک متغیر رشته ای از این تابع استفاده می شود . این تابع دارای دو ورودی و بدون خروجی می باشد . ورودی اول این تابع رشته ای است که میخواهیم اطلاعات دریافت شده روی آن ذخیره شود و ورودی دوم این تابع که از نوع unsigned char است طول رشته را مشخص می کند . تا زمانی که به تعداد معین کاراکتر دریافت نشود ، کاراکترهای دریافت شده را از USRAT گرفته و در متغیر رشته ای ذخیره می کند.

```
gets( متغیر رشته ای , طول رشته );
```

۵. تابع printf

تفاوت دستور printf با puts یا putsf در این است که می توان با دستور printf متغیر ها و رشته ها را با هم و با فرمت دلخواه ارسال کرد. برای مثال میخواهیم دمای اتاق را ارسال کنیم ، به صورت زیر میتوان این کار را انجام داد . با نوشتن کد زیر تمامی کاراکترهای موجود در عبارت " Temp=%d " به سرعت و پشت سر هم از طریق واحد یوزارت ارسال می شود و به جای d% در عبارت فوق دمای اتاق که در متغیر i قرار دارد ، ارسال می شود.

```
printf("Temp=%d",i);
```

جدول زیر فرمت متغیرهای کاراکتری قابل ارسال توسط تابع printf را نشان می دهد.

کاراکتر	نوع اطلاعات ارسالی
%c	یک تک کاراکتر
%d	عدد صحیح علامت دار در مبنای ۱۰
%i	عدد صحیح علامت دار در مبنای ۱۰
%e	نمایش عدد ممیز شناور به صورت علمی
%E	نمایش عدد ممیز شناور به صورت علمی
%f	عدد اعشاری
%s	عبارت رشته ای واقع در حافظهSRAM
%u	عدد صحیح بدون علامت در مبنای ۱۰
%X	به فرم هگزا دسیمال با حروف بزرگ
%x	به فرم هگزا دسیمال با حروف کوچک
%p	عبارت رشته ای واقع در حافظهFLASH
%%	نمایش علامت ٪

کاراکتر های کنترلی: این کاراکتر ها برای کنترل صفحه نمایش میباشند و به شکل جدول زیر میباشند:

کاراکتر کنترلی	کاری که انجام میشود
\n	به خط بعد میرود
\t	به اندازه ۸ فاصله به جلو میرود(مانند کلید تب)
\f	به صفحه بعد می رود
\a	بوق سیستم را به صدا در می آورد
\\	کاراکتر \ را چاپ میکند
\”	کاراکتر “ را چاپ میکند
\’	کاراکتر ‘ را چاپ میکند
\v	به ۸ خط بعد میرود

\r	کلید را مشخص میکند
\?	علامت ? را چاپ میکند
\:	علامت : را چاپ میکند

تعیین طول (width) و دقت (precision) خروجی در تابع printf

تابع printf این قابلیت را دارد که طول داده ارسالی و دقت آن را تعیین نماید . طول و دقت بعد از کاراکتر ٪ و قبل از حروف نشان دهنده فرمت به صورت دقت.طول نوشته می شود . برای مثال میخواهیم دقت یک عدد اعشاری را تا ۴ رقم اعشار و طول آن را تا ۷ رقم در نظر بگیریم باید آن را در تابع printf به صورت زیر بنویسیم:

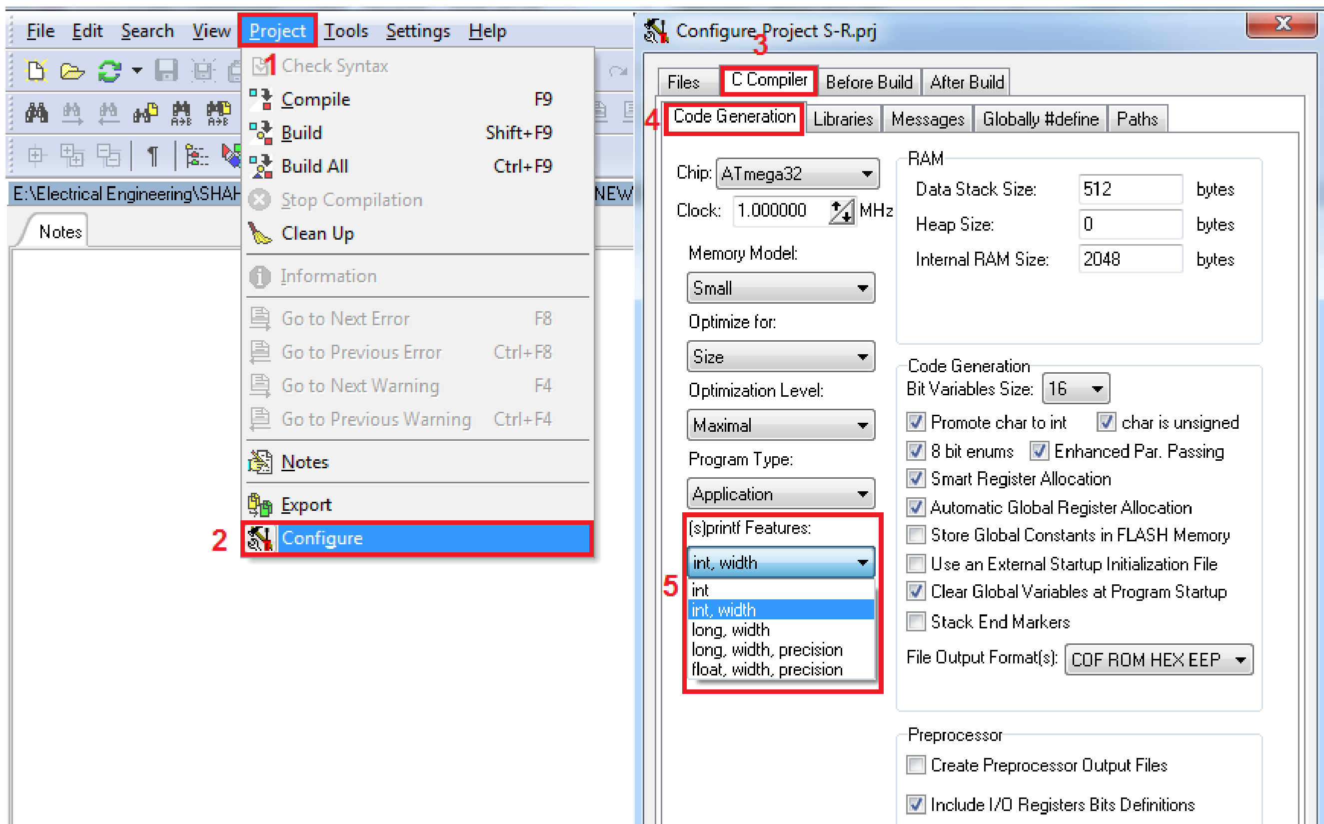
```
printf("A=%7.4f",i);
```

در مثال فوق بعد از =A به اندازه ۷ کاراکتر قرار میگیرد که حداکثر ۴ تا از آن برای اعشار و ۳ تا از آن برای قسمت صحیح عدد است . در صورتی که طول متغیر i کمتر از ۷ کاراکتر را اشغال کند به همان تعداد کاراکتر خالی سمت چپ عدد قرار می گیرد.

نکته : برای ارسال عددی از نوع Long از کاراکتر ویژه 'l' استفاده می شود . این کاراکتر ویژه را میتوان بعد از کاراکتر ٪ و قبل از کاراکترهای فرمت c ، d، u و x به کار برد . مثال:

```
printf("A=%lu",A);
```

نکته مهم : با توجه به اینکه تابع printf حجم زیادی از حافظه برنامه را به خود اختصاص می دهد ، در نرم افزار CodeVision برای استفاده بهینه از این تابع کاربر می تواند تنظیمات نوع عملکرد تابع printf را بر حسب نیاز تغییر دهد . برای این کار از منوی Project گزینه ی Configure Project را انتخاب کرده و در سربرگ C Compilerبه زیر برگ Code Generation بروید . همانطور که در شکل زیر نیز مشاهده می کنید در قسمت printf featureمی توان طول ، دقت و نوع تابع printfیا sprintf را در صورت وجود معین کرد.



۶. تابع scanf

می تواند رشته یا متغیر را از ورودی با یک فرمت مشخص دریافت و در یک آرایه ذخیره کند. مثال:

```
scanf("Temp=%d",A);
```

در مثال فوق مقدار مورد نظر را به صورت intدریافت کرده و به صورت فرمت مشخص شده در متغیر A ذخیره می کند. برای استفاده بهینه از این تابع نیز در نرم افزار CodeVision در همان قسمت Code Generation بخش scanf featureمیتوان نوع و طول ورودی را تنظیم کرد.

۷. تابع sprintf

این تابع همانند printf عمل می کند با این تفاوت که خروجی آن به جای ارسال توسط واحد USART در یک آرایه که در آرگومان اول تابع مشخص می شود ، ذخیره می شود . مثال:

```
int data=10;\nchar s[10];

sprintf(s,"Tha data is :%d",data);
```

۸. **تابع sscanf**

این تابع همانند scanf عمل می کند با این تفاوت که ورودی آن به جای دریافت از واحد USART از یک آرایه که در آرگومان اول تابع مشخص می شود ، گرفته می شود . مثال:

```
char a[10];\nsscanf(s,"%s",a);
```

پس از اجرای دستور فوق محتویات آرایه s به آرایه a منتقل می شود.

توابع پرکاربرد کتابخانه string.h برای کار با رشته ها

در صورتی که با رشته ها و آرایه های رشته ای کار می کنید ، میتوانید با اضافه کردن هدر فایل string.h از توابع مفید این کتابخانه در برنامه استفاده نمایید . در پروژه هایی مانند راه اندازی ماژول های GSM ، GPSو Bluetooth به برخی از آنها نیاز خواهید داشت.

۱. **تابع strcmp**

تابع strcmp کاراکترهای دو رشته را باهم مقایسه کرده و یک عدد صحیح نسبت به میزان تفاوت دو عدد با هم به خروجی تابع برگردانده می شود.

```
strcmp(str1,str2);
```

- اگر str1<str2 باشدمقدار برگردانده شده عددی کوچکتر از صفر خواهد بود.
- اگر str1=str2 باشد مقدار برگردانده شده برابر صفر خواهد بود.
- و اگر str1>str2 باشد مقدار برگردانده شده عددی بزرگتر از صفر خواهد بود.

۲. **تابع strcpy**

به علت اینکه مقدار دادن به متغیر رشته ای بطور مستقیم امکان پذیر نیست ، از طریق این تابع رشته ای در رشته ی دیگر قرار می گیرد . مثال:

```
strcpy(name,”ALI”);
```

۳. **تابع strncpy**

تابع strncpy تعداد مشخصی از کاراکتر های یک رشته را در رشته ی دیگر کپی می کند. مثال:

```
strncpy(str1,str2,n);
```

در مثال فوق str1 رشته ای است که به تعداد n کاراکتر از کاراکترهای str2 در آن کپی می شود.

۴. **تابع strlwr**

رشته ای را به عنوان ورودی پذیرفته و کلیه ی حروف بزرگ آن را به کوچک تبدیل می کند.

۵. **تابع struper**

رشته ای را به عنوان ورودی پذیرفته و کلیه ی حروف کوچک آن را به بزرگ تبدیل می کند.

۶. **تابع strlen**

از این تابع برای تعیین طول یک رشته مورد استفاده قرار می گیرد.

۷. **تابع strrev**

این تابع کاراکترهای یک رشته را معکوس می کند یعنی کاراکتر ابتدایی را به انتهای آن رشته منتقل و برای کاراکترهای بعدی نیز عمل معکوس کردن را انجام می دهد.

توابع ریاضی پرکاربرد

برای محاسبات ریاضی میتوانید کتابخانه math.h را به پروژه خود اضافه کنید و از توابع موجود در آن استفاده نمایید.

۱. تابع abs : قدر مطلق عدد ورودی را بر می گرداند.

```
int abs(int x);
```

۲. تابع sqrt : جذر عدد مثبت ورودی را بر می گرداند.

```
double sqrt(double x);
```

۳. تابع pow : ورودی x را به توان ورودی y کرده و بر می گرداند.

```
double pow(double x,double y);
```

۴. تابع sin : مقدار سینوس زاویه ورودی را بر حسب رادیان بر می گرداند.

```
double sin(double x);
```

۵. تابع cos : مقدار کسینوس زاویه ورودی را بر حسب رادیان بر می گرداند.

```
double cos(double x);
```

۶. تابع tan : مقدار تانژانت زاویه ورودی را بر حسب رادیان بر می گرداند.

```
double tan(double x);
```

۷. تابع log10 : مقدار لگاریتم در مبنای ۱۰ عدد ورودی را بر می گرداند.

```
double log10(double x);
```

توابع کاراکتری پرکاربرد

این توابع در فایل ctype.h قرار دارند و برای کار با کاراکترها مورد استفاده هستند.

۱. تابع isalnum : این تابع کاراکتری را در ورودی دریافت کرده و در صورتی که جزو حروف a تا z (A تا Z) و یا ۰ تا ۹ نباشد، مقدار صفر را به خروجی بر می گرداند. در غیر این صورت مقدار بازگشتی غیر صفر خواهد بود.

```
int isalnum(int ch);
```

۲. تابع isdigit : تابع بالا کاراکتری را در ورودی دریافت کرده و اگر مقدار آن یکی از ۰ تا ۹ نباشد ، صفر را بر می گرداند و در غیر این صورت مقدار بازگشتی غیر صفر خواهد بود.

```
int isdigit(int ch);
```

۳. تابع ispunct : این تابع کاراکتری را در ورودی دریافت کرده و اگر جزو حروف ویرایشی مثل کاما ، نقطه و غیره نباشد ، مقدار صفر را به خروجی برمی گرداند. در غیر این صورت مقدار بازگشتی غیر صفر خواهد بود.

```
int ispunct(int ch);
```

۴. تابع touper : این تابع کاراکتری را در ورودی دریافت کرده و آن را به حروف بزرگ انگلیسی تبدیل می کند.

```
int touper(int ch);
```

۵. تابع tolower : این تابع کاراکتری را در ورودی دریافت کرده و آن را به حروف کوچک انگلیسی تبدیل می کند.

```
int tolower (int ch);
```

اشاره گرها

زمانی که یک متغیر تعریف می شود ، بخشی از حافظه را اشغال می کند. بسته به نوع متغیر این بخش میتواند یک یا چند بیت یا بایت باشد. اشاره گر خود نیز یک متغیر است که به جای ذخیره کردن داده آدرس محل قرارگیری متغیرهای دیگر را در خود ذخیره می کند. یعنی یک اشاره گر به محل ذخیره یک متغیر در حافظه اشاره می کند. هر اشاره گر می تواند آدرس متغیر هم نوع خود را در خود نگه دارد. برای مثال برای نگه داری آدرس یک متغیر int نیاز به تعریف یک اشاره گر از نوع نیاز به تعریف یک اشاره گر از نوع int می باشد. قالب تعریف یک اشاره گر را در زیر مشاهده می کنید :

نوع اشاره گر * نام اشاره گر

```
int *x;
char *y;
```

همانطور که مشاهده می شود تنها تفاوت یک اشاره گر با متغیر در * قبل از نام اشاره گر است. برای نگه داری دائمی یک اشاره گر ، محل ذخیره یک اشاره گر میتواند توسط یکی از کلمات flash,eeprom تعیین شود. در صورت تعیین نکردن محل حافظه ، اشاره گر به صورت پیش فرض در حافظه SRAM ذخیره خواهد شد.

مقدار دهی به اشاره گر

برای اینکه آدرس محل یک متغیر را در یک اشاره گر ذخیره کنیم ، از عملگر & استفاده می کنیم. مثال :

```
int *x,y;
y=142;
x=&y;
```

در این مثال در ابتدا یک اشاره گر و یک متغیر هر دو از نوع int تعریف شده است. به متغیر y مقداری نسبت داده شده است. برای اینکه آدرس محل ذخیره متغیر y در حافظه را داشته باشیم ، از عملگر & استفاده می کنیم.

دسترسی به محتوای یک اشاره گر

برای اینکه به محتوای یک اشاره گر که به محلی از حافظه اشاره می کند را داشته باشیم ، از عملگر * استفاده می کنیم. مثال :

```
int *x,y,z;
y=142;
x=&y;
z=*x;
```

یک متغیر z به برنامه اضافه کردیم و در آن محتوای x را نسبت دادیم. بنابراین z=142 می شود.

عملیات روی اشاره گرها

عملیات جمع و تفریق را می توان روی متغیرهای اشاره گر انجام داد اما ضرب و تقسیم را روی یک اشاره گر نمی توان استفاده کرد. نکته مهمی که باید به آن توجه کرد این است که چون اشاره گر آدرسی در حافظه است وقتی

عملیاتی که روی آن انجام می گیرد رفتار متفاوتی دارد. برای مثال عمل جمع اشاره گر را به تعداد بایت های نوع داده آن حرکت می دهد .

مثال : چون a اشاره گری به یک عدد int است و نوع int ۲ بایت دارد با عمل افزایش ۲ واحد به a اضافه می شود. یعنی به ۲ بایت بعدی حافظه اشاره می کند.

```
int a=10;
int *p;
p=&a;
p=p+2;
```

ارتباط اشاره‌گر با آرایه و رشته

در زبان برنامه‌نویسیC ، بین آرایه‌ها با رشته‌ها و اشاره‌گرها، ارتباط نزدیکی وجود دارد. اشاره‌گرها حاوی آدرس هستند و اسم هر آرایه یا رشته نیز یک آدرس است. اسم آرایه، آدرس اولین عنصر آرایه را مشخص می‌کند؛ به‌عبارت‌دیگر، اسم هر آرایه، آدرس اولین محلی را که عناصر آرایه ازآنجا به بعد در حافظه ذخیره می‌شوند، نگهداری می‌کند؛ بنابراین اسم هر آرایه، یک اشاره‌گر است.

به مثال زیر توجه کنید:

```
int table [5];

int *p;
```

همان‌طور که در مثال صفحه قبل مشاهده می‌شود، یک آرایه با ۵ عنصر به نام tableو اشاره‌گر p هر دو از نوع int معرفی‌شده‌اند. اگر اولین عنصر آرایه tableدر محل ۱۰۰۰ حافظه واقع‌شده باشد، نام **آرایه** به محل ۱۰۰۰ آرایه اشاره خواهد کرد.

```
p=table;
```

چون هر دو متغیر table و p از جنس اشاره گر هستند ، یدون هیچ عملگری آنها را میتوان برابر هم قرار داد. در نتیجه میتوان گفت موارد زیر با یکدیگر معادل هستند :

*(p+1) معادل table[1]

p[2]   معادل   *(table+1)

*p   معادل   *table

استفاده از اشاره گرها در توابع

توابعی که قبلا با آن آشنا شدیم، تنها یک مقدار را به خروجی باز می گرداندند. با استفاده از اشاره گرها در ورودی توابع ، میتوان توابعی ساخت که بیش از یک خروجی داشته باشند. در این روش که به آن فراخوانی تابع با ارجاع گفته می شود، به جای متغیرهای ورودی در هنگام تعریف تابع ، اشاره گر قرار می گیرد و در هنگام فراخوانی تابع ، به جای متغیرهای ورودی آدرس آنها قرار می گیرد. بنابراین اگر تابعی می خواهیم که چند پارامتر خروجی دارد، می توانیم پارامترهای خروجی را در لیست پارامترهای ورودی تابع و به صورت اشاره گر تعریف کنیم تا تابع آنها را پر کرده و تحویل ما دهد. به مثال زیر توجه کنید :

```
void fx(int *a,int *b,int *c);
void main(void){
int x=10,y=20,z;
fx(&x,&y,&z);
}
void fx(int *a,int *b,int *c){
int i,j,k;
i=*a;
j=*b;
i=i/2;
j=j/2;
*a=I;
*b=j;
*c=i+j;
}
```

در این مثال محتوای ورودی به متغیر های I,j ریخته می شود و سپس بر روی آن عملیات مورد نظر صورت می گیرد. در این تابع هر دو متغیر I,j تقسیم بر دو شده است. سپس مقدار جدید i به جای محتوای آدرس a قرار می گیرد. همچنین مقدار جدید متغیر j به جای محتوای آدرس اشاره گر b و جمع I,j نیز در محتوای آدرسی قرار می گیرد که اشاره گر c به آن اشاره می کند. در نتیجه در پایان برنامه y=10 x=5 و z=15 می شود. (تابعی با ۳ ورودی و ۳ خروجی)

ساختار

همان‌طور که تا اینجا آموختیم، آرایه‌ها می‌توانند برای جمع‌آوری گروه‌هایی از متغیرهایی با نوع مشابه مورداستفاده قرار گیرند؛ بنابراین نمی‌توان به‌عنوان‌مثال آرایه‌ای تعریف کرد که شامل پنج‌خانه از نوع صحیح و پنج‌خانه از نوع اعشاری باشد. از طرفی هم در کاربردهای مختلف برنامه‌نویسی نیاز به تعریف کردن عناصر مختلف در کنار هم و منسوب کردن یک نام به همه‌ی آن‌ها داریم تا بتوانیم مجموعه‌ی آن‌ها را به‌صورت یکجا مورد پردازش‌هایی مانند بازنویسی آن‌ها به‌صورت یکجا، ارسال کل آن‌ها به یک تابع و یا آماده‌سازی و برگرداندن همه‌ی آن‌ها به‌عنوان نتیجه یک تابع قرار دهیم.

فرض کنید داده‌های مربوط به یک دانشجو مثل شماره دانشجویی، نام خانوادگی، نام، جنسیت، تعداد واحد گذرانده شده و معدل کل را که دارای نوع‌های متفاوتی هستند را بخواهیم تحت یک نام تعریف کنیم.

یا به‌عنوان‌مثال دیگر، اگر بخواهیم اطلاعات مربوط به کارکنان شرکتی را که شامل نام کارمند (از نوع کاراکتری)، شماره کارمندی (از نوع عدد صحیح)، حقوق)از نوع عدد صحیح) و ... است، تحت یک نام ذخیره کنیم، در این صورت متغیر معمولی و آرایه پاسخگوی نیاز ما نیستند. اکنون می‌خواهیم بدانیم که چگونه قطعات داده‌هایی را که نوع یکسان ندارند، (مانند مثال فوق) جمع‌آوری کنیم.

پاسخ این است که می‌توانیم متغیرهایی با نوع‌های مختلف را با نوع داده‌ای به نام ساختار گروه‌بندی کنیم. بنابراین می‌توان گفت که ساختار در زبانC ، نامی برای مجموعه‌ای از متغیرهاست که این متغیرها می‌توانند هم نوع نباشند، یعنی می‌تواند ساختاری از انواع مختلف داده‌ها ازجمله float, int, char, unsigned char و ... را درقالب یک نام در خود داشته باشد و کاربر در هر زمان به آن‌ها دسترسی داشته باشد.

قالب تعریف ساختار را در زیر مشاهده می‌کنیم:

```
struct [structure tag]{
   member definition
   member definition
   ...
   member definition
}[one or more structure variables];
```

نام ساختار یا (structure tag) از قانون نام‌گذاری برای متغیرها تبعیت می‌کند. عضوهای ساختار یا(member) ، متغیرهایی هستند که قسمتی از ساختار می‌باشند و همانند یک متغیر معمولی یا آرایه، باید اسم و نوع هرکدام مشخص باشد. لیست نام‌ها یا (structure variables) هم متغیرهایی هستند که قرار است ساختمان این ساختار را داشته باشند. برای استفاده از عناصر ساختار معرفی‌شده باید متغیرهایی از نوع ساختار پس‌ازآن معرفی شود. دو روش برای این کار وجود دارد.

در زیرقالب **روش اول** را مشاهده می‌کنید:

```
struct نام ساختار{
  عناصر ساختار
}نام متغیرها;
```

به مثال زیر توجه کنید:

```
struct student_record{
long  student_number;   /*شماره دانشجویی*/
char  first_name[21];   /*نام دانشجو*/
char  last_name[31];    /*نام خانوادگی*/
```

```
char  gender_code;      /*کد جنسیت*/

float average;          /*معدل کل*/

int   passed_units  ;   /*واحدهای گذرانده شده*/

}student1;              /* تعریف متغیری از نوع ساختمان این ساختار*/
```

در این تعریف student_record نام الگوی تعریف‌شده برای این ساختار است که می‌تواند نوشته نشود و student1 نام متغیری است با ساختمان این ساختار که دارای شش عضو است. درصورتی‌که بعد از خاتمه‌ی تعریف ساختار (بعد از {) نامی نوشته نشود، فقط یک الگو تعریف‌شده است و چون متغیری با ساختمان این ساختار تعریف‌نشده، حافظه‌ای اشغال نخواهد شد. در این حالت می‌توان در ادامه‌ی برنامه از کلمه‌ی **struct** و نام ساختار (در اینجا**student_record**) برای تعریف ساختارهای موردنیاز استفاده کرد.

در زیرقالب **روش دوم** را می‌بینید که پس از معرفی ساختار صورت می‌گیرد:

```
Struct <نام ساختار> <نام متغیرها>;

Struct s_type  p1,p2;
```

در بالا متغیرهایی بانام **p1,p2** از نوع ساختار **s_type** معرفی‌شده‌اند. هرکدام از این‌ها حاوی کل ساختار معرفی‌شده، می‌باشند. درروش دوم در زمان معرفی ساختار، متغیرهایی از نوع ساختار در انتهای آن معرفی می‌شوند.

نکته ۱ : در کامپایلر کدویژن با استفاده از کلمات کلیدی **eeprom** و **flash** این قابلیت وجود دارد که ساختارها را در ناحیه SRAM ، FLASH یا EEPROM تعریف نمود. در صورتی که نام محل ذخیره سازی ذکر نشود ، کامپایلر به طور پیش فرض حافظه SRAM را انتخاب میکند.

نکته ۲ : ساختارهایی که در حافظه **flash** تعریف می شوند ، از نوع ساختار ثابت می باشند و نمیتوان در عناصر آن نوشت و فقط میتوان آنها را خواند.

دسترسی به عناصر ساختار

قالب دسترسی به عناصر ساختار با استفاده از نقطه (dot) و به صورت زیر است:

نام عناصر موردنظر در ساختار. نام متغیر از نوع ساختار

ابتدا نام ساختار حاوی آن عضو و سپس نام عضو بیان‌شده و این نام‌ها با عملگر عضو ساختار که علامت آن نقطه است به یکدیگر مرتبط می‌شوند. این عملگر که دارای بالاترین تقدم عملیات بوده ترتیب اجرایش از چپ به راست است. توجه شود که به‌صورت قراردادی در دو طرف عملگر نقطه فاصله خالی گذاشته نمی‌شود .اگر عناصر از نوع آرایه باشند، ذکر اندیس آرایه جهت دستیابی به آن عنصر ضروری است.

مثال :ساختاری را تعریف کنید که اطلاعات یک دانشجو را در خود ذخیره نماید.

```
Struct student{
   Long int id;
   Char name[20];
   Float average;
   Int age;
}s;
```

مثال: مقداردهی اولیه به عناصر متغیر ساختاریs

```
s.id = 860133710;
s.name = "ali";
s.average = 17.64;
s.age = 18;
```


www.sbargh.ir

تخصیص ساختارها

در مقداردهی اولیه می‌توان مجموعه‌ای از مقادیر را به یک ساختار نسبت داد، ولی انجام چنین کاری در متن برنامه به‌صورت دستور اجرایی امکان‌پذیر نیست .تنها عمل تخصیص که در مورد رکوردها در زبان C تعریف‌شده است، تخصیص یک ساختار به ساختار دیگری با ساختمان دقیقا یکسان (هر دو ساختار از طریق یک struct تعریف‌شده باشند) هست که در این صورت محتویات هر فیلد از ساختار مبدأ به فیلد متناظر از ساختار مقصد منتقل می‌شود. ساختار مبدأ می‌تواند متغیری در برنامه یا خروجی یک تابع باشد.

اشاره‌گرها و ساختارها

در زبان C تعریف اشاره‌گر از نوع ساختار، همانند تعریف سایر انواع اشاره‌گرها امکان‌پذیر است. همان‌طور که در فراخوانی تابع می‌توانید اشاره‌گری را ارسال کنید که به آرایه ارجاع می‌دهد، می‌توانید اشاره‌گری که به ساختار اشاره می‌کند را نیز ارسال کنید.

به‌هرحال برخلاف ارسال ساختار به تابع که نسخه‌ی کاملی از ساختار را به تابع می‌فرستد، ارسال اشاره‌گر به ساختار فقط آدرس ساختار را به تابع می‌فرستد. سپس تابع می‌تواند جهت دستیابی مستقیم به اعضای ساختار از آدرس استفاده می‌کند و از سرریزی تکرار ساختار پرهیز نماید، بنابراین این روش جهت ارسال اشاره‌گر به ساختار کارآمدتر است، نسبت به اینکه خود ساختار را به تابع ارسال کنید.

اشاره‌گر ساختار به دو منظور استفاده می‌شود:

- امکان فراخوانی به روش ارجاع را در توابعی که دارای آرگومان از نوع ساختار هستند، فراهم می‌کند.
- برای ایجاد فهرست‌های پیوندی و سایر ساختار داده‌هایی که با تخصیص حافظه پویا سروکار دارند، به کار می‌رود.

وقتی‌که ساختارها از طریق فراخوانی به روش ارجاع به توابع منتقل می‌شوند، سرعت انجام عملیات بر روی آن‌ها بیشتر می‌گردد. لذا در حین فراخوانی توابع، بهتر است به‌جای ساختار، آدرس آن را منتقل نمود. عملگر & برای مشخص کردن آدرس ساختار مورداستفاده قرار می‌گیرد.

تعریف اشاره‌گرهای ساختار مانند تعریف متغیرهای ساختار است، با این تفاوت که قبل از اسم متغیر، علامت * قرار می‌گیرد.

مثال زیر را در نظر بگیرید، در این دستورات person یک متغیر ساختار و p یک اشاره‌گر ساختار تعریف‌شده است.

```
Struct bal{
   Float balance;
   Char name[80];
}person, *p;
```

اکنون دستور زیر را در نظر بگیرید

```
p = &person;
```

با این دستور، آدرس متغیر ساختار person در اشاره‌گر p قرار می‌گیرد. برای دسترسی به محتویات عناصر ساختار از طریق اشاره‌گر، باید اشاره‌گر را در داخل پرانتز محصور کنید. به‌عنوان‌مثال دستور زیر موجب دسترسی به عنصر balance از ساختار person می‌شود. علت قرار دادن متغیر اشاره‌گر در پرانتز، این است که تقدم عملگر نقطه از * بالاتر است.

```
(*p).balance;
```

به‌طورکلی برای دسترسی به عناصر ساختاری که یک اشاره‌گر به آن اشاره می‌کند به دو روش می‌توان عمل کرد:

- ذکر نام اشاره‌گر در داخل پرانتز و سپس نام عناصر موردنظر که با نقطه از هم جدا می‌شوند. (مثل دسترسی به عنصر balance از ساختار person توسط اشاره‌گر p)
- استفاده از عملگر ->که روش مناسب‌تری است. اگر بخواهیم با استفاده از عملگر-> به عنصر balance از ساختار person دسترسی داشته باشیم باید به طریق زیر عمل کنیم (علامت -> متشکل از علامت منها و علامت بزرگ‌تر است).

```
p -> balance;
```

در این‌جا بیان این نکته ضروری است که آرایه‌ها، اشاره‌گرها و ساختارها دارای ارتباط نزدیکی باهم هستند. در ضمن عمگرهای مربوط به آن‌ها شامل زیر نویس یا []، عضور رکورد یا نقطه و دستیابی غیرمستقیم به عضو رکورد یا -> همگی دارای بالاترین تقدم هستند .عملگرهای دستیابی غیرمستقیم یا * و استخراج آدرس یا &هم دارای تقدم دوم می‌باشند. حال اگر این عبارت‌ها همراه همدیگر در عبارتی ظاهر شوند باید کمی با دقت کد مورد نظر نوشته شود. به مثال زیر دقت کنید:

```
struct point{
        float x;
        float y;
};
struct line{
        struct point strat;
        struct point end;
        char *name;
} a = {1, 1, 10, 20, "ab"};
struct line *pa, *pm,
        m[] = {{2, 3, 4, 5, "cd"},{4,6,8,1, "mn"},{8,5,4,2, "xy"}};
pa = &a;
pm = &m[1];
```

با توجه به تعاریف بیان شده و اینکه ترتیب اجرای دو عملگر عضو ساختار و دستیابی غیرمستقیم از چپ به راست است، عبارت‌های زیر معادل هستند:

```
a.start.x
pa->start.x
(a.start).x
(pa->start).x
(*pa).start.x
```

دو عبارت زیر نیز یک معنی می‌دهند:

```
m[1].end.y
pm->end.y
```

با این توضیح که در اولی دستیابی از طریق اندیس خانه‌ی یکم آرایه‌ی m که یک ساختار است به عضو end و سپس به عضو y انجام شده است، ولی در دومی دستیابی به عضو end از طریق آدرس خانه‌ی یکم آرایه‌ی m که در متغیر pmقرار دارد انجام شده است.

آرایه‌ای از ساختارها

در C، نام آرایه را می‌توانید در مقابل نام ساختار قرار دهید تا آرایه‌ای از ساختارها را اعلام کنید. یکی از بیشترین موارد کاربرد ساختارها، استفاده از آن‌ها به‌عنوان عناصری از آرایه است. برای تعریف آرایه‌ای از ساختارها، ابتدا نوع ساختار را تعریف کرده سپس همانند متغیرهای معمولی، آرایه‌ای از آن نوع را تعریف می‌کنیم. برای نمونه، با فرض داشتن ساختاری به نام x حکم زیر راداریم:

```
Struct x array_of_structure[8];
```

آرایه‌ای به نام array _of_structure از Struct x اعلام کرده است. این آرایه ۸ عنصر دارد، هر عنصر آن نمونه یکتایی از Struct x است .مثال زیر را در نظر بگیرید:

```
Struct student{
   Char name[21];
   int stno;
   Float ave;
};
Struct student st[100];
```

در این دستورات، آرایه ۱۰۰ عنصری st طوری تعریف می‌شود که هر یک از عناصر آن، از نوع ساختار student است .

یونیونها

ساختمان یونیون کاملا مشابه با ساختار است با این تفاوت که به جای کلمه کلیدی **struct** از کلمه کلیدی **union** استفاده می شود. اما در یونیون محل مشخصی از حافظه ، بین دو یا چند متغیر به صورت اشتراکی مورد استفاده قرار می گیرد به طوری که متغیرها همزمان نمیتوانند از این محل در حافظه استفاده کند ، بلکه هر متغیر در زمان های متفاوتی می تواند این محل را مورد استفاده قرار دهد. بنابراین فضایی که یک یونیون در حافظه اشغال می کند مانند فضایی که یک ساختمان اشغال می کند نیست. بلکه یونیون بیشترین طول متغیر درون خود را (به لحاظ طول بیت) به عنوان طول خود در نظر می گیرد و این فضا را بین بقیه متغیرهای درون یونیون به اشتراک می گذارد. یونیون نیز همانند ساختمان دو روش تعریف دارد. نحوه تعریف یونیون به صورت زیر است :

union نام یونیون

{

عناصر یونیون

};

مثال :

```
union u_type
{
char l;
int x;
float y;
}
```

در این یونیون تعریف شده چون بزرگترین نوع float است که ۴ بایت حافظه را اشغال می کند ، بنابراین کل یونیون ۳۲ بیت از حافظه را اشغال میکند.

روش اول معرفی یونیون : بعد از تعریف

union نام یونیون نام متغیرها ;

مثال :

```
union u_type i1,i2;
```

روش دوم معرفی یونیون : در حین تعریف

مثال :

```
union u_type
{
char I;
int x;
float y;
}i1,i2;
```

دسترسی به عناصر یونیون : با استفاده از نقطه صورت می گیرد. مثال :

```
i1.i=33;
i2.x=2048;
i1.y=6.28;
i2.i=254;
```

نکته : هنگامی که از نوع های داده ای کوچکتر از ۳۲ بیت استفاده شود بقیه بیت ها بدون استفاده و ۰ هستند.

داده شمارشی

این نوع داده قابلیت نامگذاری یک مجموعه را می دهد. برای مثال میتوان روزهای هفته را درون یک داده شمارشی قرار داد به طوری که روز اول هفته عدد ۰ و روز آخر هفته عدد ۶ تخصیص می یابد. قالب معرفی نوع داده شمارشی را در زیر مشاهده می کنید :

```
enum نام نوع شمارشی
{
عنصر اول;
عنصر دوم;
…
عنصر آخر;
};
```

مثال :

```
enum color
{
red;
green;
blue;
yellow;
};
```

روش اول معرفی نوع شمارشی : بعد از تعریف

```
enum نام نوع شمارشی   نام متغیرها ;
enum color c1,c2;
```


www.sbargh.ir

روش دوم معرفی نوع شمارشی : در حین تعریف

مثال :

```
enum color
{
red;
green;
blue;
yellow;
}c1,c2;
```

نحوه مقدار دهی به نوع شمارشی : مثال :

```
c1=red;
c2=blue;
```

تغییر نام انواع داده ها با دستور typedef

توسط این دستور در زبان c میتوان نام داده هایی همچون int,char,float,... را به هر نام دلخواه دیگری تغییر داد. این دستور دو مزیت دارد :

1. موجب می شود تا برای نوع داده هایی که نام طولانی دارند ، نام کوتاه تری انتخاب کرد.
2. اگر برنامه به کامپایلر دیگری منتقل شود و کامپایلر جدید و قبلی از نظر نوع و طول داده ها مطابقت نداشته باشد، برای حل مشکل کافیست فقط در دستور typedef تغییراتی ایجاد شود.

قالب استفاده از این دستور را در زیر مشاهده می کنید :

typedef نام موجود نام جدید;

در قالب بالا نام موجود یکی از نوع های معتبر در زبان سی است و نام جدید نامی دلخواه برای آن می باشد. مثال :

```
typedef char str;
```

بعد از دستور بالا میتوان به جای char از str برای تعریف داده های جدید استفاده کرد. مثال :

```
str x,y;
```

لزوم برنامه نویسی به سبک ماژولار

در طراحی و توسعه ی یک پروژه لازم است به موارد زیر توجه شود:

تولید کننده ای که بخواهد برای هر سیستم جدید از ابتدا شروع به طراحی برنامه کند نمی تواند برای مدت طولانی توانایی رقابت در بازار را داشته باشد. زبان برنامه نویسی که استفاده می شود باید توانایی ایجاد کتابخانه های انعطلاف پذیر را داشته باشد، تا برنامه نویس بتواند از کتابخانه هایی که آزمایش (test)، اشکال زدایی (debug) و تایید (verify)شده اند در پروژه های آینده استفاده کند. همچنین لازم است امکان انطباق کتابخانه با میکروکنترلرهای جدید وجود داشته باشد.

کارکنان یک مجموعه تغییر می کنند و حتی اگر ثابت باشند، حافظه ی انسان بازه ی زمانی محدودی را به خاطر می آورد. هر سیستمی نیاز به ارتقا و به روز رسانی دارد و اگر برنامه ی آن به شیوه ی درستی نوشته و مستند سازی نشده باشد، درک و تغییر آن مشکل می شود. بنابراین برنامه ی خوب، برنامه ای است که امکان درک و تغییر آن در هر زمان وجود داشته باشد (نه فقط به وسیله ی طراح اولیه، بلکه به وسیله افراد دیگر).

علاوه بر موارد فوق لازم است به این نکته توجه شود که در اجرای پروژه های بزرگ و پیچیده اگر از سبک برنامه نویسی مناسبی استفاده نشود، نگهداری و اشکال زدایی برنامه بسیار مشکل می شود، هزینه ها افزایش می یابد و امکان موفقیت طرح به حداقل می رسد. همچنین گاهی لازم است پروژه های بزرگ به اجزای کوچکتری خرد شوند و هر بخش را فرد و یا تیم مستقلی پیاده سازی کند. با استفاده از سبک برنامه نویسی ماژولار، می توانیم کتابخانه هایی ایجاد کنیم که به سادگی قابل تغییر و قابل استفاده در پروژه های دیگر هستند.

مفهوم ماژول نرم افزاری

ماژول به معنای مولفه و یا جزئی از برنامه است که خود شامل تعدادی تابع مرتبط با یکدیگر است. معمولا مجموع توابعی که داخل یک ماژول قرار می گیرند عملکرد خاصی را پیاده سازی می کنند.

برخی از ماژول ها ارتباط جانبی داخلی یا خارجی ندارند و معمولا فرآیندی را بر روی داده انجام می دهند. در این نوع ماژول ها، بخشی از برنامه به عنوان مشتری، سرویسی را از ماژول درخواست می کند و ماژول آن درخواست را پاسخ می دهد.

برخی از ماژول ها نیز ارتباط برنامه با یک سخت افزار داخلی (مانند پورت های سریال، مبدل آنالوگ به دیجیتال و ...) یا خارجی (مانند نمایشگرLCD ، صفحه کلید، سنسور و ...) را برقرار می کنند. ماژول شامل دو بخش پیاده سازی و واسط (Interface) است. در بخش پیاده سازی، بدنه ی توابع قرار می گیرند و در بخش دوم نحوه ی استفاده از توابع، در اختیار کاربر (مشتری سرویس) قرار می گیرد.

یکی از دلایل اساسی استفاده از ماژول این است که بتوان بخش های لازم از یک موضوع را مشخص و بخش های غیر ضروری را پنهان کرد. بنابراین بخش واسط، شامل اطلاعاتی است که برای استفاده از آن موضوع مورد نیاز است و بخش پیاده سازی، شامل چگونگی انجام آن موضوع است. به پنهان سازی جزئیات غیر ضروری، Abstractionگفته می شود. از این مفهوم در طراحی بسیاری از وسایل استفاده می شود. به عنوان مثال زمانی که یک خودرو را می رانیم لزوما نیاز نیست که از نحوه ی عملکرد موتور و سایر اجزا آن آگاهی داشته باشیم، بلکه کافی است که واسط استفاده موتور(شامل پدال های کلاچ، ترمز و گاز) در اختیارمان باشد و نحوه ی به کارگیری این واسط را بدانیم. در برنامه نویسی نیز، Abstractionمفهوم بسیار ارزشمندی است که در زبان های شی گرا (مانند java و C++) به شکل عالی و در زبان های ماژولار به شکل ابتدایی از آن پشتیبانی شده است.

متاسفانه زبان C از شی گرایی و برنامه نویسی ماژولار پشتیبانی نمی کند، اما با استفاده از فایل های سورس (source)و هدر (header) می توان به تا حد قابل قبولی به شیوه ی ماژولار برنامه نویسی کرد.

برنامه نویسی به روش ماشین حالت

یکی از روش های برنامه نویسی پیشرفته ، استفاده از ساختار ماشین حالت یا **State Machine** در برنامه نویسی می باشد. این سبک از برنامه نویسی دارای کمترین هنگ و خطا و بیشترین انعطاف پذیری است. در این روش از حلقه **switch** برای تشریح حالت های مختلف و از نوع شمارشی **enum** برای تدوین انواع حالت ها استفاده خواهیم کرد. بنابراین ساختار کلی این روش به صورت زیر است :

```
…
enum { s0,s1,s2,….,sn}state;
void main {
…
while(1){
کاری که هر بار باید انجام شود و حالت های مختلف بر اساس آن شکل می گیرد
switch(state)
{
case s0:
کارهایی که در حالت اول باید صورت گیرد
if( شرط رفتن به حالت بعد ) state=حالت بعدی


case s1:
کارهایی که در حالت دوم باید صورت گیرد
if( شرط رفتن به حالت بعد ) state=حالت بعدی
.
.
```

.

case sn:

کارهایی که در حالت آخر باید صورت گیرد

if(شرط رفتن به حالت بعد) state=حالت بعدی

default:

کارهایی که در صورت برقرار نبودن هیچ یک از حالت های فوق انجام می شود

}

}

}

پایان

منابع :

برنامه نویسی به زبان C ، عین الله جعفر نژاد قمی
مرجع کامل میکروکنترلرهای AVR ، پرتوی فر مظاهریان بیانلو
آموزش کاربردی میکروکنترلرهای AVR ، شجاع داودی
Atmega32 Datasheet, Atmel corporation