کتاب PDF

میکرو کنترلرهای AVR با پروژه های ۱۰۰٪ عملی

تألیف جابر الوندی

منبع درس ریزپردازنده۱ دانشگاه پیام نور

فایل PDF موجود نسخه RF، در بین سایر فایل های PDF ریزپردازنده موجود در سایت های دانلود کتاب، کامل ترین فایل PDF محسوب میشود. چرا که در سایر کتاب های PDF مشابه، علاوه بر ناقص بودن صفحات کتاب، نظم و ترتیب خاصی بین صفحات وجود ندارد که این نقص، باعث ایجاد مشکل و سردرگمی برای دانشجویان در هنگام مطالعه کتاب PDF می شود.

از همین رو ویرایش RF کتاب پی دی اف ریزپردازنده۱ را که نواقص فایل های قبلی را برطرف کرده، برای استفاده دانشجویان عزیزی که به کتاب اصلی این درس دسترسی ندارند، آماده کرده ایم.

Edited by Aref.b *4123ph@gmail.com*

فصل ۲

اصول طراحی و آموزش زبان C

اهداف

۱. آشنایی با اصول طراحی و ترسیم الگوریتم
۲. آموزش دستورات برنامه‌نویسی C مخصوص میکروکنترلر
۳. معرفی توابع کتابخانه‌ای و کاربرد آنها در برنامه‌نویسی

۲-۱ اصول طراحی

در یک طراحی الکترونیکی می‌بایست اصول و قوانین آن رعایت شود. هر شخص طراح می‌بایست دارای سه خصوصیت باشد:

۱. دایره اطلاعاتی (علم روز و ابزارات جدید را خوب بشناسد و خود را بتواند Update کند)
۲. شناخت سخت‌افزار (طراحی PCB ، شناخت سنسور و میکروکنترلر و شناخت PLC و ...)
۳. آگاهی کامل از یک زبان برنامه‌نویسی (زبان C ، پاسکال، بیسیک، اسمبلی، VHDL و ...)

هر طراحی سه مرحله دارد که یک به یک به این مراحل می‌پردازیم.

۱. فهمیدن صورت مسئله اولین گام برای یک طراحی است

برای اینکه یک شخص طراح بتواند موفق باشد می‌بایست صورت مسئله را درک کند. گاهی اوقات پیشنهاد طراحی با جمله‌ای بسیار کوتاه و یا بسیار بلند داده می‌شود اما یک طراح باید در درجه اول خوب بفهمد که متقاضی طرح چه سیستمی مورد نظرش می‌باشد و در درجه دوم باید معیارهایی که در صورت مسئله وجود ندارد را در نظر بگیرد.

مثال ۲-۱ : سیستمی را طراحی کنید که دمای یک دستگاه صنعتی را اندازه بگیرد و هر موقع که از دمای تنظیم شده، درجه حرارت بالاتر رفت دستگاه را با خاموش کردن آن محافظت نماید.

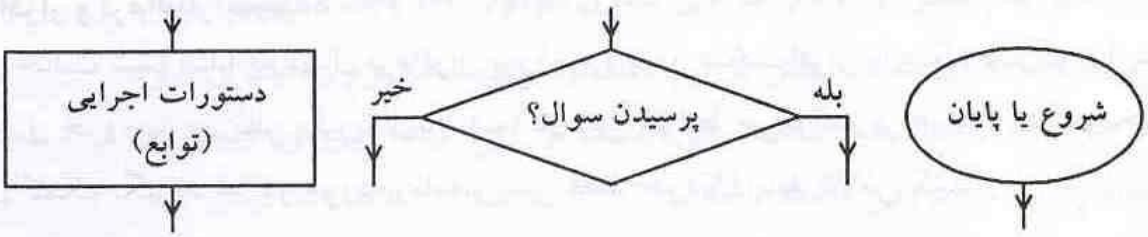
توضیح مثال ۲-۱ مطرح شده : در این مثال طراحی، شما می‌بایست، در ظاهر سیستمی را طراحی کنید که بتواند دما را اندازه بگیرد و قابلیت تنظیم را نیز داشته باشد. اما اگر به باطن صورت مسئله توجه کنیم در می‌یابیم که این دستگاه در صنعت می‌خواهد استفاده شود پس قضیه نویز نباید فراموش شود طراحی باید طوری باشد که در قبال نویز مصونیت داشته باشد. نکته دیگر اینکه دمای محیط اطراف سنسور موجب تغییرات لحظه‌ای سیستم می‌شود یعنی اگر به طور مثال دمای تنظیم شده 40 درجه سانتی‌گراد باشد و بالای این دما دستگاه توسط سیستم طراحی شده خاموش شود و دما برای لحظه‌ای به 39 درجه سانتی‌گراد برسد دستگاه دوباره روشن می‌شود پس این یعنی به نوسان افتادن ناخواسته سیستم و این مشکل بزرگی است زیرا در این حالت نه تنها دستگاه محافظت نمی‌شود بلکه با روشن و خاموش شدن مکرر، ممکن است دستگاه آسیب ببیند. برای حل این مشکل باید یک محدوده مُرده تعریف کنیم که بالای عدد تنظیم شده دستگاه را خاموش کند و در دمایی با اختلاف قابل تنظیم دستگاه را روشن کند. به طور مثال بالای 40 درجه دستگاه را خاموش کند و در دمای زیر 36 درجه سانتی گراد دستگاه را روشن کند. ضمناً در این صورت مسئله عنوان نکرده است که ماکزیمم دما چقدر است که ما آن را در انتخاب سنسور دمای سیستم معیار قرار دهیم. همچنین فاصله قرار گرفتن سنسور از محل مورد نظر تا سیستم نادیده گرفته شده است.

مثال ۲-۲ : سیستمی برای راه‌اندازی یک موتور سه فاز طراحی کنید که اول بصورت ستاره شروع بکار کند و بعد از مدتی به حالت مثلث برود.

توضیح : در این طراحی نیز، شما باید علاوه بر قابلیت تغییر زمان بین راه‌اندازی ستاره به مثلث، برای این سیستم کلیدهای Start و Stop در نظر گرفته و ضمناً موقعیکه می‌خواهد از حالت ستاره به مثلث تغییر حالت بدهد بهتر است یک زمان تأخیری کوتاهی قرار داده شود.

۲. طراحی الگوریتم برنامه (فلوچارت)

دومین مرحله از اصول طراحی، الگوریتم برنامه نرم‌افزاری پروژه می‌باشد. الگوریتم‌ها در طراحی و عیب‌یابی کارایی ویژه‌ای دارند و برای تمام برنامه نویسان یک اصل مهم می‌باشد و حُسن آن در این است که کاربر بر اساس آن می‌تواند، برنامه بنویسد. مزیت دیگر آن این است که کاربر بعد از یک مدتی اگر به الگوریتم برنامه نگاه کند سریعاً متوجه عملکرد برنامه می‌شود. در این مرحله طراح می‌بایست فارغ از اینکه با چه میکروکنترلری کار می‌کند و یا به چه زبان برنامه‌نویسی، مسلط است باید برنامه را بصورت پرسش و پاسخ که همان طراحی الگوریتم (فلوچارت) است در بیاورد و مطابق آن مراحل بعدی را طی کند. البته زمانی که شما یک برنامه‌نویس حرفه‌ای شوید شاید الگوریتم یک برنامه را در ذهن خود تجسم کنید و نیازی به ترسیم آن نداشته باشید اما قدری مشکل به نظر می‌رسد. پروژه‌های این کتاب خیلی پیچیده نبوده و نیازی به ترسیم الگوریتم نداشته‌اند.



شکل ۲-۱ علائم استاندارد طراحی الگوریتم

☑ علامت بیضی مشخص کننده شروع و پایان برنامه می‌باشد.
☑ علامت لوزی برای مطرح کردن سؤال است و دارای خروجی بله یا خیر می‌باشد.
☑ علامت مستطیل نشانگر دستورات اجرایی که همان زیر برنامه‌های زبان برنامه‌نویسی است.

در طراحی فلوچارت باید نکات زیر رعایت شود:

۱. نخست اینکه باید در متن نوشتاری داخل بلوک‌ها خلاصه‌نویسی رعایت شود به طوری که آن جمله کوتاه بتواند مفهوم عملکرد لازم را برساند.
۲. بلوک‌های مرتبط با هم در کنار یکدیگر قرار گیرند تا از پیچیدگی فلوچارت جلوگیری شود.
۳. برای برنامه‌های بزرگ بهتر است اول قسمتی از برنامه را طراحی کرده و سپس قسمت‌های دیگر را بعد از موفقیت در قسمت قبل دنبال کنید.

طراحی الگوریتم یک پروژه دارای ویژگی‌هایی برای یک طراح است نمونه بارز آن فهمیدن سخت‌افزار و نرم‌افزار می‌باشد چرا که از طریق آن می‌توان میکروکنترلر یا PLC خود را انتخاب کرد. به طور مثال در یک الگوریتم شما احتیاج به سه تایمر و هفت وقفه خارجی دارید و حدس می‌زنید که ظرفیتی حدود 40 کیلو بایت مورد نظر است پس می‌بایست دنبال تراشه‌ای با این خصوصیات بریم. بهترین انتخاب در چنین شرایطی ATmega128/64 می‌باشد ولی در شرایطی که نیاز به وقفه‌های زیاد و ظرفیت بالایی نیست تراشه‌ای که اقتصادی‌تر باشد را انتخاب می‌کنیم. ضمناً طراحی الگوریتم به ما کمک می‌کند که برنامه نرم‌افزاری را با چه زبانی بنویسیم.

۳. نوشتن برنامه نرم‌افزاری و طراحی سخت‌افزار پروژه

در این مرحله کاربر می‌بایست انتخاب کند که اول سخت‌افزار را طراحی کند یا اینکه برنامه نرم‌افزاری را بنویسد برای این مرحله از کار، سه حالت وجود دارد:

۱. سخت‌افزار پیچیده ولی نرم‌افزار آسان

در چنین وضعیتی بهتر است که اول سخت‌افزار را طراحی کنید زیرا اگر نتوانید از عهده سخت‌افزار آن برآیید حتماً نوشتن نرم‌افزار کمکی نخواهد کرد.

۲. سخت‌افزار آسان ولی نرم‌افزار پیچیده

در این حالت نیز راه پیچیده را انتخاب می‌کنیم چرا که اگر نتوانیم نرم‌افزار را بنویسیم طراحی سخت‌افزار راهی اشتباه است.

۳. سخت‌افزار و نرم‌افزار پیچیده

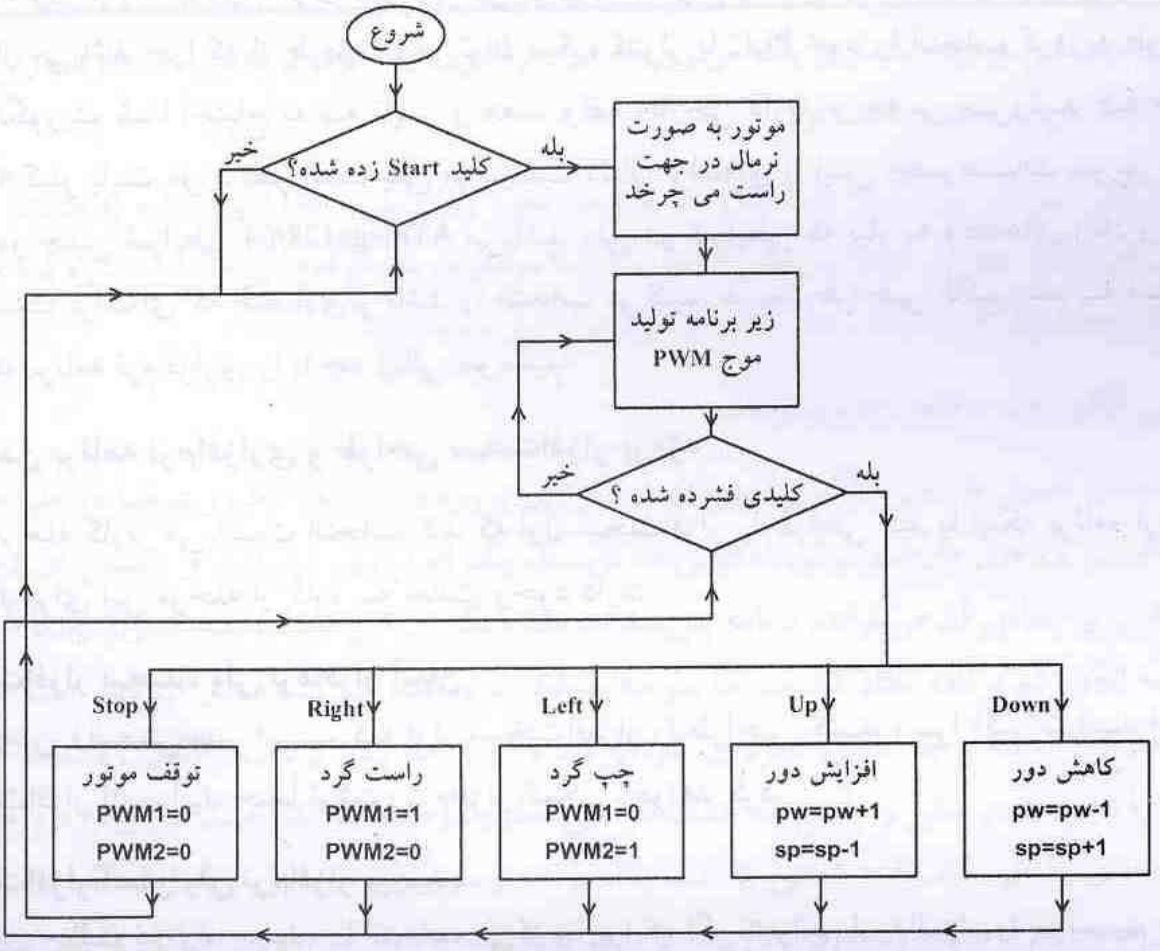
در این حالت شما ابتدا به دنبال نرم‌افزار پروژه بروید و سخت‌افزار را یک بلوک با یک سری ورودی و خروجی در نظر بگیرید زیرا شما در این شرایط می‌توانید برای طراحی سخت‌افزار از دیگران کمک بگیرید اما در مورد برنامه‌نویسی فقط خودتان باید تلاش کنید.

خصوصیاتی که سخت‌افزار پروژه باید داشته باشد:

۱. تا حد امکان سخت‌افزار ساده و کم حجم باشد.
۲. در انتخاب ورودی و خروجی‌های میکروکنترلر بهترین حالت را انتخاب کنیم.
۳. PCB پروژه را خوب طراحی کنیم.
۴. مدار را تا می‌توانیم از نویز پذیری مصون کنیم (این کار را با گذاشتن فیلترها و خازن‌های حذف نویز و عایق بندی با فلز متصل به زمین و سیم‌های شیلد دار و . . . انجام می‌دهیم)

خصوصیاتی که نرم‌افزار پروژه باید داشته باشد:

۱. تا آنجایی که ممکن است کم ظرفیت باشد.
۲. توابع و دستورات اضافی در آن وجود نداشته باشد.
۳. برنامه را به زبانی که تسلط دارید بنویسید.
۴. قبل از نوشتن نرم‌افزار ورودی و خروجی‌ها را تعریف کنید.



شکل ۲-۲ الگوریتم ساده راه‌اندازی موتور DC

نکاتی که از الگوریتم شکل ۲-۲ در طراحی کنترل موتور DC باید رعایت شود:

۱. برنامه باید دارای مقادیر پیش فرض باشد و موتور ابتدا نرم راه‌اندازی شود.
۲. به جز کلید Start بقیه کلیدها می‌بایست از نوع وقفه بیرونی میکروکنترلر استفاده شوند تا عملکرد بسیار سریعی داشته باشند.
۳. ماکزیمم و مینیمم درصد PWM باید مشخص شود.

مقدمه‌ای بر زبان C

زبان C در سال ۱۹۷۲ توسط دنیس ریچی طراحی شد و علت نام‌گذاری C، این است که بعد از زبان B (BCPL) طراحی شده است. زبان‌های برنامه‌نویسی به لحاظ ساختاری به سه دسته تقسیم می‌شوند **زبان‌های سطح بالا** (زبان‌هایی که به زبان محاوره‌ای انسان نزدیکتر هستند مانند: پاسکال، بیسیک)، **زبان‌های میانی** (زبان‌هایی که مستقیماً به حافظه دستیابی دارند و از طرفی قابلیت انعطاف پذیری همچون زبان‌های سطح بالا را بر خوردار هستند)، **زبان‌های سطح پایین** (زبان‌هایی که با سخت‌افزار کار دارند مانند: اسمبلی)، زبان C از آنجا که ارتباط نزدیکی با زبان اسمبلی دارد جزء زبان‌های میانی است اما با توجه به کارایی بالای این زبان در برنامه‌نویسی میکروکنترلرها، آن را می‌توان یک زبان قدرتمند و سطح بالا دانست. این زبان قابلیت انعطاف‌پذیری را دارد و تمام ضعف‌های زبان اسمبلی را پوشش می‌دهد و با وجود اینکه تعداد کلمات کلیدی در این زبان بسیار کمتر از زبان بیسیک است اما نسبت به بیسیک بسیار قدرتمندتر است. شرکت‌های سازنده میکروکنترلر بعد از طراحی تراشه، کامپایلر (مفسر) آن را به بازار عرضه می‌کنند معمولاً برنامه‌نویسی استاندارد میکروکنترلرها به زبان‌های اسمبلی، C، بیسیک و پاسکال وجود دارند که ارتباط بسیار نزدیکی با معادل این زبان‌ها در کامپیوتر دارند. به عنوان مثال زبان C مبتنی بر میکروکنترلر تمام قوانین زبان C سیستم را می‌پذیرد و مواردی به آن اضافه شده است. شما برای یادگیری زبان C میکروکنترلر می‌بایست زبان C سیستم را فرا بگیرید اما فراموش نکنید که برخی از توابع و فایل‌های الحاقی (سر آمد) فقط برای زبان C کامپیوتر کارایی دارند. در این فصل شما زبان C مخصوص میکروکنترلر را یاد می‌گیرید. یکی از مزایای این زبان آن است که فرقی در برنامه‌نویسی بین میکروکنترلرها وجود ندارد و این فقط قابلیت‌های کامپایلرها و رجیسترهای تراشه‌هاست که برنامه‌نویسی آنان را از هم متمایز ساخته است.

۲-۲ آموزش زبان C

در زبان C کاربر می‌بایست دستورات، توابع و سخت‌افزار تراشه‌ای که با آن کار می‌کند را بشناسد. موقعیکه ما یک برنامه را به زبان C می‌نویسیم بعد از کامپایل شدن به فایل‌های متعددی از جمله فایل هگزاد و اسمبلی تبدیل می‌شود که در فصل سوم توضیح خواهیم داد.

مفاهیم اولیه زبان C

- در زبان C بین حروف کوچک و بزرگ تفاوت وجود دارد. در این زبان دستورات و کلمات کلیدی با حروف کوچک نوشته می‌شوند به عنوان مثال : کلمه کلیدی **void** با کلمه VOID فرق

دارد و اگر دستورات و کلمات کلیدی را با حروف بزرگ بنویسیم کامپایلر پیغام خطا می‌دهد.

- هر دستور به ; ختم می‌شود که نشان دهنده آن است که خط دستوری پایان یافته است.
- حداکثر طول یک دستور، ۲۵۵ کاراکتر است.
- هر دستور را می‌توان در یک سطر یا چند سطر نوشت.
- در هر سطر می‌توان چند دستور نوشت.
- توضیحات در یک سطر می‌توانند بعد از // قرار بگیرند و همچنین توضیحات گروهی با /* شروع می‌شوند و به */ ختم می‌شوند.

ساختار کلی زبان C

```
#include <نام تراشه.h>                    //معرفی فایل‌های سرآمد یا الحاقی
#include <نام کتابخانه.h>
#define   out1   PORTC.0                  //معرفی شناسه‌ها
.       .
.       .
.       .
unsigned  char   x,y;                     //معرفی متغیرهای ۸ بیتی کلی (همگانی)
unsigned  int    N,count;                 //معرفی متغیرهای ۱۶ بیتی کلی (همگانی)
.       .
.       .
.       .
void function1(){                         //توابع فرعی
unsigned char k,j;                        //معرفی متغیرهای ۸ بیتی محلی
دستورالعمل‌ها;
}
.       .
.       .
.       .
void main(){                              //تابع اصلی برنامه
دستورالعمل‌ها;
function1();                              //فراخوانی توابع فرعی
While(1){                                 //حلقه بی نهایت برنامه
 };
}
```

پیش پردازنده‌ها

فایل‌های سرآمد توسط دستورات پیش پردازنده در ابتدای برنامه معرفی (گنجانده) می‌شوند. در زبان C می‌بایست میکروکنترلری که استفاده می‌کنید را معرفی نمائید و بعد از آن کتابخانه‌هایی را که به آنها در برنامه‌نویسی نیاز دارید، باید معرفی کنید. توجه داشته باشید که کتابخانه‌ها برای میکروکنترلرها و کامپایلرهای مختلف، متفاوت می‌باشند.

در واقع فایل‌های سرآمد، فایل‌های از قبل نوشته شده‌ای می‌باشند که در پوشه INC در مسیر نصب نرم‌افزار قرار دارند. حداکثر فایل‌های سرآمد در برنامه، بستگی به نوع کامپایلر دارد. مثلاً نرم‌افزار CodeVisionAVR، ۱۶ فایل سرآمد در یک برنامه را می‌پذیرد.

فایل‌های سرآمد را با دستور پیش پردازنده #include در بین دو علامت < > قرار می‌دهیم و پسوند اینگونه فایل‌ها *.h می‌باشد. بعضی از کامپایلرها اجازه ساخت کتابخانه و فایل الحاقی را می‌دهند که در فصل سوم در این مورد بحث خواهیم کرد.

تعریف شناسه‌ها یا ثوابت

شناسه (ماکرو) برای تعریف بر چسب‌هایی معادل یک نام رشته‌ای یا هر مقداری می‌باشند و ضمناً می‌توانند برای تعریف ماکروهای شبه تابع استفاده شوند که در واقع شبیه دستور EQU در زبان اسمبلی عمل می‌کند. برای درک بیشتر به مثال‌های زیر توجه فرمائید.

مثال ۲-۳:

```
#define  name     "Micro"
#define  number   125.456
#define  out1     PORTA.5
```

در این تعاریف، کلمه name با مقدار رشته‌ای "Micro" برابر است و number نیز با عدد 125.456 برابر است. فراموش نکنید که شما ثوابت را تعریف می‌کنید بنابراین مقدارهای number یا name نمی‌توانند در برنامه تغییر کنند. کلمه out1 معادل PORTA.5 می‌باشد و می‌تواند صفر یا یک شود.

مثال ۲-۴:

```
#define  sum(x,y)  x+y
i=sum(4,6);
```

در این تعریف ماکرو، عمل جمع انجام می‌شود. بنابراین اعداد ۴ و ۶ با هم جمع می‌شوند و حاصل آن یعنی عدد ۱۰ در متغیر از قبل تعریف شده، یعنی متغیر i قرار می‌گیرد.

متغیرها

متغیر نامی است که ما برای حافظه موقت یا دائمی میکروکنترلر تعریف می‌کنیم. متغیر یا داده می‌تواند در توابع دارای مقدار باشد و اعمال منطقی و ریاضیاتی بر روی آن در قالب یک تابع انجام پذیرد. در واقع یک کاربر با متغیر کار دارد و نیازی نیست که ثبات‌ها و آدرس مکانی حافظه را به خاطر بسپارد هر چند می‌توان نحوه ذخیره‌سازی یک متغیر را تعیین کرد. برای نام‌گذاری متغیرها از حروف کوچک و بزرگ لاتین (a...z or A...z) و همچنین از علائمی نظیر آندر لاین (_) استفاده می‌شود. توجه داشته باشید که نمی‌توانید از کلمات کلیدی و دستورات به عنوان نام متغیر استفاده کنید و همچنین نباید متغیر با عدد شروع شود ولی می‌تواند به عدد ختم شود. متغیرها در زبان C می‌توانند ۸ بیتی، ۱۶ بیتی، ۳۲ بیتی و اعداد اعشاری باشند. این که متغیر را از چه نوع داده‌ای تعریف کنیم بستگی به مقداری است که می‌خواهیم در آن قرار دهیم. در حافظه مقدار متغیرها آورده شده است.

توجه

هر چقدر داده‌ها را شما از نوع ۸ بیتی تعریف کنید و کمتر از متغیرهای کلی استفاده کنید حجم کد برنامه کاهش می‌یابد.

انواع داده‌ها در زبان C

نوع متغیر	اندازه بر حسب بیت	محدوده تغییرات
bit	1	0 or 1
char یا signed char	8	-128 to 127
signed char	8	0 to 255
int یا signed int	16	-32768 to 32767
signed int	16	0 to 65535
signed short int	16	-32768 to 32767
unsigned short int	16	0 to 65535
long int	32	-2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
float	32	±1.175e-38 to ±3.402e38
double	32	±1.175e-38 to ±3.402e38

جدول ۲-۱ انواع داده‌های استاندارد در زبان C

نحوه‌ی تعریف متغیر

متغیرها در زبان C دو نوع اند یکی کلی و دیگری محلی، که نحوه‌ی تعریف آنها مشابه می‌باشد و فقط محل تعریف آنها در برنامه‌نویسی متفاوت است.

انواع متغیر در زبان C

۱. **متغیرهای کلی** : این متغیرها بعد از معرفی پیش پردازنده‌ها و شناسه‌ها در ابتدای برنامه و خارج از بدنه‌های توابع می‌آیند و مقدار آنها در تمامی توابع قابل دسترسی می‌باشد و مقدار خود را در تمامی توابع حفظ می‌کنند.

۲. **متغیرهای محلی** : این متغیرها در داخل بدنه توابع تعریف می‌شوند و مقدار آنها با خارج شدن از بدنه توابع از بین می‌رود یعنی صفر می‌شوند.

معرفی متغیر به شکل زیر است:

نوع داده نام متغیر ;

مثال ۲-۵ : دو متغیر ۸ بیتی بی علامت X و Z تعریف کنید و به متغیر X مقدار اولیه 12 بدهید و یک

متغیر دیگر با نام k از همین نوع در آدرس ۲۰۰ هگزاد SRAM و همچنین یک متغیر از نوع اعشاری با نام fv تعریف کنید.

```
unsigned  char x=12,z;
unsigned char k@0x200;
float   fv=0.0;
```

همان طور که ملاحظه می‌کنید متغیرهای x و z چون از یک نوع داده بودند در یک خط تعریف شدند و با علامت کاما (,) از هم جدا شدند. این کار برای تعریف سریعتر متغیر در یک خط به کاربر کمک می‌کند. می‌توانستیم متغیرهای x و z را به شکل زیر نیز تعریف کنیم و مقدار x را در ابتدا ندهیم.

```
unsigned  char x ;
unsigned  char z ;
x=12;
```

فرمت یا مبنای متغیرها

⇦ ( ) : اگر شما قبل از عدد علامتی قرار ندهید به معنای دسیمال است.

```
unsigned char X=15;   // به معنی مقدار دادن ۱۵ دسیمال می‌باشد
unsigned char X=15U;  // قرار دادن حرف U بعد از دسیمال اختیاری است
float X=3.14F;        // قرار دادن حرف F بعد از یک داده اعشاری اختیاری است
```

⇦ (0x) : اگر قبل از یک عدد بیاید به معنی هگزاد است.

```
unsigned char X=0x12;  //  به معنی قرار دادن مقدار ۱۲ هگزاد می‌باشد
```

⇦ (0b) : اگر قبل از عدد بیاید به معنی باینری است.

```
unsigned char X=0b11001111; // به معنی قرار دادن مقدار بر حسب باینری می‌باشد
```

⇦ (" ") : اگر کلمه‌ای در بین دو علامت دابل کوتیشن قرار گیرد یعنی مقدار رشته‌ای است.

```
char X[]="alvandi";
```

⇦ (' ') : اگر یک کاراکتر در بین دو علامت کوتیشن قرار گیرد مقدار بر حسب اسکی است.

```
unsigned char X='s';
```

کلاس ذخیره‌سازی متغیرها

در معرفی یک متغیر می‌توان نحوه‌ی ذخیره‌سازی آن را تعریف کرد. برای تعیین کلاس ذخیره‌سازی می‌بایست قبل از تعریف نوع داده متغیر، کلاس آن را تعیین کنیم. فرم کلی:

< کلاس ذخیره‌سازی حافظه > < نوع داده > نام متغیر ;

انواع کلاس ذخیره‌سازی :

auto : متغیرهایی که در داخل بدنه یک تابع تعریف می‌شوند، متغیرهای محلی نامیده می‌شوند. این متغیرها با فراخوانی یک تابع، در درون آن شکل می‌گیرند و با برگشتن از تابع از بین می‌روند به این نوع کلاس ذخیره‌سازی، کلاس حافظه اتوماتیک گفته می‌شود. این حالت برای متغیرهای محلی، پیش

فرض کامپایلر می‌باشد و نیاز نیست که قبل از نوع داده کلمه auto بیاید به عبارت دیگر، متغیرهای تعریف شده در یک تابع از نوع اتوماتیک هستند.

static : متغیرهای محلی که از نوع ثابت تعریف می‌شوند با خارج شدن از یک تابع از بین نمی‌روند و فقط در بدنه همان تابعی که تعریف شده‌اند قابل دسترسی هستند. متغیرهای کلی در واقع متغیرهای ثابتی هستند که در تمامی توابع قابل دسترسی‌اند و محتوای آنها از بین نمی‌رود.

extern : این نوع کلاس برای متغیرهایی است که در یک فایل دیگر (مثل فایل سر آمد) معرفی و مقداردهی اولیه شده‌است و در فایل جاری برنامه، می‌توان از آنها استفاده کرد.

register : این نوع کلاس، نحوه‌ی ذخیره شدن یک متغیر را در یکی از رجیسترهای میکروکنترلر تعیین می‌کند و فقط در قالب متغیرهای محلی کاربرد دارند.

عملگرها

در زبان C به علائم منطقی و ریاضیاتی، عملگر گفته می‌شود. عملگرها را نمی‌توان به عنوان دستور نامید زیرا آنها فقط یک عمل خاصی را انجام می‌دهند.

تقدم عملگر	عملگر	مفهوم	مثال
اولویت اول	++	افزایش به اندازه یک واحد	X++
	--	کاهش به اندازه یک واحد	x--
اولویت دوم	-	علامت منفی	-x
اولویت سوم	*	ضرب	x * y
	/	تقسیم (خارج قسمت)	X/10
	%	تقسیم (باقی مانده)	X%10
اولویت چهارم	-	تفریق	x-y
	+	جمع	X+y

جدول ۲-۲ عملگرهای محاسباتی

مثال ۲-۶ : مقدار متغیر a=14 را بعد از یک واحد افزایش در متغیر b قرار دهید؟

```
unsigned  char a=14,b;          (قسمت اول)
b=a++;    //b=14
unsigned  char a=14,b;          (قسمت دوم)
b=++a;    //b=15
```

در مثال ۲-۶ قسمت اول اشتباه می‌باشد. چون عملگر ++ در اولی بعد از متغیر a آمده است بنابراین ابتدا مقدار قبلی a در متغیر b قرار می‌گیرد و بعد یک واحد به متغیر a اضافه می‌شود.

مثال ۲-۷ : چه مقداری در متغیر z بعد از انجام عبارت زیر قرار می‌گیرد؟

```
unsigned int   x=7,y=6,z;
z=x+y*6/2;
```

برای مثال ۲-۷ ممکن است شما جواب‌های متفاوتی همچون اعداد 39، 21.5 و 25 برای این مسئله داشته باشید اما جواب صحیح عدد 25 می‌باشد چرا که تقدم عملگر نشان‌دهنده حاصل عبارت خواهد بود. در این عبارت ابتدا y در عدد 6 ضرب شده و حاصل آن بر عدد 2 تقسیم شده و نتیجه با مقدار متغیر x جمع می‌شود.

نکته : هرگاه از دو عملگر با تقدم یکسان استفاده شود اولویت با عملگری است که اول بیاید. در این مثال چون عملگر ضرب، زودتر از عملگر تقسیم آمده است ابتدا عمل ضرب انجام می‌گیرد.

مثال ۲-۸ : چه مقداری در متغیر z بعد از انجام عبارت زیر قرار می‌گیرد؟

```
unsigned int   x=7,y=6,z;
z=(x+y)*(6/2);
```

در این مثال حاصل عبارت ۳۹ خواهد بود.

نکته : هرگاه یک عبارت در بین علامت پرانتز( ) قرار گیرد، اولویت بالاتری خواهد داشت.

مثال	مفهوم	عملگر	تقدم عملگر
a>b	بزرگتر	>	اولویت اول
a>=b	بزرگتر یا مساوی	>=	
a<b	کوچکتر	<	
a<=b	کوچکتر یا مساوی	<=	
a==b	متساوی	==	اولویت دوم
a!=b	نامساوی	!=	

جدول ۲-۳ عملگرهای مقایسه‌ای (رابطه‌ای)

مثال	مفهوم	عملگر	تقدم عملگر
!x	نقیض (not)	!	اولویت اول
a && b	وَ (and)	&&	اولویت دوم
a || b	یا (or)	||	اولویت سوم

جدول ۲-۴ عملگرهای منطقی

عملگر	مفهوم	مثال	معادل
=	مساوی یا mov	x=12	
*=	انتساب ضرب	x*=y	x=x*y
/=	انتساب تقسیم	x/=10	x=x/10
%=	انتساب باقی مانده تقسیم	x%=10	x=x%10
+=	انتساب جمع	x+=y	x=x+y
-=	انتساب تفریق	x-=y	x=x-y
&=	انتساب AND بیتی	x&=z	x=x&z
^=	انتساب XOR بیتی	x^=z	x=x^z
|=	انتساب OR بیتی	x|=z	x=x|z
<<=	انتساب شیفت به چپ	x<<=5	x=x<<5
>>=	انتساب شیفت به راست	x>>=3	x=x>>3

جدول ۲-۵ عملگرهای بیتی و منطقی

تقدم عملگر	عملگر	مفهوم	مثال	نتیجه
اولویت اول	~	مکمل ۱	~ (0x66)	0x9A
اولویت دوم	>>	شیفت به راست	0b11001011 >> 4	0b00001100
	<<	شیفت به چپ	0b11001011 << 4	0b10110000
اولویت سوم	&	AND بیتی	0x66 & 0xff	0x66
اولویت چهارم	^	XOR بیتی	0x12 ^ 0x12	0
اولویت پنجم	|	OR بیتی	0x05 | 0x30	0x35

جدول ۲-۶ عملگرهای انتسابی یا ترکیبی

عملگر شرطی ؟

فرم کلی استفاده از این عملگر بصورت زیر است :

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

در این فرم دستوری عبارت شرطی اول مورد ارزیابی قرار می‌گیرد که اگر نتیجه آن درست باشد، عبارت دوم در متغیر قرار می‌گیرد و در غیر اینصورت عبارت سوم در متغیر قرار خواهد گرفت.

مثال ۲-۹:

```
y=(x > z) ? x : z
```

در مثال ۲-۹ مقدار x و z با هم مقایسه می‌شود که اگر x بزرگتر باشد عبارت دوم یعنی x در متغیر y قرار می‌گیرد و گرنه مقدار z در متغیر y قرار خواهد گرفت.

عملگرهای & و *

با استفاده از عملگر & می‌توانیم به آدرس یک متغیر و با استفاده عملگر * می‌توانیم به محتوای آدرس یک متغیر دسترسی داشته باشیم.

مثال ۲-۱۰ :

```
unsigned char y@0x400;  //معرفی متغیر در مکان ۴۰۰ هگزاد در SRAM
unsigned char *x,i;     //معرفی متغیر اشاره‌گر x و متغیر دلخواه i
y=15;                   //مقداردهی متغیر با عدد ۱۵ بر حسب دسیمال
x=&y;                   //متغیر x برابر آدرس متغیر y یعنی ۴۰۰ هگزاد می‌شود
i=*x;                   //محتوای مکانی که x به آن اشاره می‌کند را داخل i کپی می‌کند
```

در مثال ۲-۱۰ آدرس متغیر y، 400 هگزاد و مقدار آن 15 دسیمال می‌باشد و در سطر چهارم مقدار x برابر آدرس y می‌شود و در سطر پنجم، x به مکان 400 هگزاد از SRAM داخلی اشاره می‌کند و محتوای آن مکان یعنی 15 دسیمال را به متغیر i کپی می‌کند.

توجه

اگر از اشاره‌گر * استفاده می‌کنید باید توجه داشته باشید که نوع داده اشاره‌گر، باید با نوع داده آرایه یا رشته مورد نظر، یکی باشد. این را نیز به یاد داشته باشید که عملگرهای & و * را با عملگرهای مشابه انتسابی اشتباه نگیرید.

عملگر کاما (,)

این عملگر برای جداسازی چند متغیر، عبارت و عمل دستوری می‌باشد.

مثال ۲-۱۱ :

```
unsigned char a,b;
b=(a=10,a/2);
```

عملگر کاما در سطر اول، برای جداسازی دو متغیر در یک سطر است و در سطر دوم برای جداسازی دو عبارت به کار رفته است که حاصل متغیر b برابر 5 خواهد بود.

عملگر sizeof ( )

این عملگر طول یک متغیر یا نوع داده را بر حسب بایت بر می‌گرداند به طور مثال داریم:

```
unsigned int y,x;
x=sizeof(y);        //طول نوع داده int بر حسب بایت ۲ می‌باشد پس مقدار x=2 می‌شود
```

- **عملگر ( )** : پرانتزها عملگرهایی هستند که تقدم عملگرهای داخل خود را بالا می‌برند و در تعریف توابع نیز بکار می‌روند. (جدول ۲-۷)

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

جدول ۲-۷ تقدم عملگرها در حالت کلی

دستورات زبان C

دستورات زبان C در قالب بدنه یک تابع می‌آیند و یک عمل خاصی را بر روی متغیرهای کلی و محلی انجام می‌دهند. هر دستور در زبان C با علامت { باز می‌شود و با علامت } بسته می‌شود.

۱. دستور شرطی if – else :

از این دستور برای ارزیابی یک شرط کنترلی در برنامه استفاده می‌شود.

ساختار کلی این دستور بصورت زیر است:

```
if ( شرط ) {
دستورالعمل‌های ۱;
}
else{
دستورالعمل‌های ۲;
}
```

در این دستور ابتدا شرط if، مورد بررسی قرار می‌گیرد که اگر درست باشد دستورالعمل‌های ۱ انجام می‌شوند و در غیر اینصورت دستورالعمل‌های ۲ اجرا می‌شوند.

مثال ۲-۱۲ :

```
unsigned  char  a,b;
if (a > b){
  a++ ;
  b+=10;}
else{
  a-- ;
  b-=10;
}
```

در مثال ۲-۱۲ اگر متغیر a از متغیر b بزرگتر باشد a یک واحد افزایش پیدا کرده و به متغیر b، عدد ۱۰

اضافه می‌شود و در غیر اینصورت متغیر a یک واحد کم شده و از متغیر b، عدد ۱۰ کم می‌شود.

مثال ۲-۱۳:

```
unsigned char zx,sd;
if (zx==sd) zx++;
else sd++;
```

در مثال ۲-۱۳ اگر هر دو متغیر با هم برابر باشند، به متغیر zx یک واحد اضافه می‌شود و در غیر اینصورت یک واحد به متغیر sd اضافه خواهد شد. همچنین می‌بینید که علامت‌های { } وجود ندارند. زمانی که یک دستورالعمل وجود داشته باشد نیازی به علامت { و } نمی‌باشد.

مثال ۲-۱۴:

```
unsigned char data,i;
bit x,y;
if ((x==1)&&(y==0))
{
data*=10;
i=data;
}
```

در مثال ۲-۱۴ متغیرهای تک بیتی x و y با هم مقایسه می‌شوند که اگر x=1 باشد و y=0 باشد data در عدد 10 ضرب می‌شود و در متغیر i کپی می‌شود. همچنین در این مثال مشاهده کردید که علامت } پایین دستور if آمده است و هیچ فرقی ندارد. ضمناً همیشه لازم نیست که با هر if یک else وجود داشته باشد ولی عکس این قضیه صادق نیست.

مثال ۲-۱۵:

```
unsigned int x,y,z;
if ((x==y) && (z==15)) x++;
else if ((x>=y) && (z==15)) y++;
else if ((x<=y) && (z==15)) z++;
```

همان طور که می‌بینید از دستور else if برای شرط‌های متوالی استفاده شده است. اگر شرط سطر دوم درست باشد متغیر x یک واحد افزایش پیدا می‌کند در غیر اینصورت شرط سطر سوم مورد ارزیابی قرار می‌گیرد و همین طور تمامی دستورات به طور متوالی تست می‌شوند.

۲. دستور حلقه for :

از این دستور برای ساختار حلقه شرطی در برنامه و زمانی که نیاز به کانتر حلقه باشد، استفاده می‌شود. فرم کلی این دستور بصورت زیر است:

```
for ( مقداردهی اولیه ; شرط حلقه ; شمارنده حلقه ) {
دستورالعمل‌ها;
}
```

در دستور for، ابتدا متغیر حلقه، مقداردهی اولیه می‌شود و در هر مرحله از اجرای حلقه یک واحد شمارنده حلقه افزایش پیدا می‌کند و تا زمانی که شرط حلقه درست باشد این حلقه تکرار می‌شود.

مثال ۲-۱۶:

```
unsigned  char i;
unsigned int w;
for ( i=0 ; i < 25 ; i++){
w+=5;
}
```

در مثال ۲-۱۶ متغیر حلقه، یعنی i در ابتدای حلقه صفر است و در هر مرحله که حلقه تکرار می‌شود یک واحد افزایش پیدا می‌کند تا زمانی که مساوی عدد 25 شده و شرط حلقه نقض شده و خط برنامه از حلقه خارج می‌شود. بنابراین این حلقه 25 بار تکرار می‌شود و مقدار ۵، در هر مرحله از تکرار حلقه به متغیر w افزوده می‌شود. همچنین در این دستور بدلیل وجود یک دستورالعمل می‌توانستیم علامت‌های {} را حذف کنیم. توجه کنید که مقداردهی اولیه یک بار در ابتدای ورود به حلقه انجام می‌گیرد و در هر تکرار حلقه مقداردهی اولیه انجام نمی‌شود.

حلقه در حلقه با دستور for:

در بسیاری از موارد نظیر تأخیرهای طولانی، ساعت، تقویم سالیانه و ... ما نیاز به حلقه‌های تودرتو داریم که آنها را با استفاده از دستور for بوجود می‌آوریم.

مثال ۲-۱۷:

```
unsigned  char i,j,z=2;
for ( i=10 ;i>0 ;i--){
     for ( j=0 ;j <=50;j++){
       z*=10;
       if (z ==140) z=2;
     }
 }
```

در مثال ۲-۱۷ برنامه ابتدا وارد حلقه اول شده و متغیر i=10 می‌شود و چون بزرگتر از عدد صفر (شرط درست) است وارد حلقه دوم شده و متغیر حلقه دوم یعنی j=0 می‌شود و چون کوچکتر مساوی عدد 51 می‌باشد این حلقه ادامه پیدا می‌کند و متغیر z در عدد 10 ضرب شده و با دستور if مورد ارزیابی قرار می‌گیرد که اگر مقدار آن برابر عدد 140 شود متغیر z را برابر 2 نماید. بنابراین حلقه دوم 51 مرتبه تکرار می‌شود و بعد برنامه از حلقه دوم خارج می‌شود و یک واحد از شمارنده حلقه اول کم شده و چون شرط همچنان برقرار است دوباره وارد حلقه دوم می‌شود. بنابراین نتیجه می‌گیریم که حلقه دوم 10 مرتبه در حلقه اول تکرار می‌شود و در هر مرحله خود حلقه دوم 51 مرتبه تکرار می‌شود.

ایجاد کردن حلقه بی نهایت با استفاده از دستور for بصورت زیر است:

```
for(;;){
دستورالعمل‌ها;
}
```

۳. دستور حلقه شرطی while :

این دستور نیز برای تکرار اجرای دستورات در یک حلقه برنامه می‌باشد با این تفاوت که فاقد شمارنده حلقه است. این دستور ابتدا شرط حلقه را تست می‌کند اگر شرط درست باشد، خط برنامه وارد حلقه می‌شود و در غیر اینصورت این حلقه هرگز اجرا نمی‌شود. این دستور بعد از هر بار تکرار حلقه، شرط را بررسی می‌کند. اگر نتیجه شرط حلقه درست باشد (غیر صفر باشد) حلقه را ادامه می‌دهد و در غیر اینصورت از حلقه خارج می‌شود.

فرم کلی این دستور بصورت زیر می‌باشد:

```
while (شرط) {
دستورالعمل‌ها;
}
```

مثال ۲-۱۸ :

```
unsigned char a,b;
while (a > b){
 a=(a/b*2);
}
```

در مثال ۲-۱۸ ابتدا متغیر a با b مقایسه شده که اگر a بزرگتر از b باشد، برنامه وارد حلقه شده و در هر بار تکرار حلقه شرط مورد تست قرار می‌گیرد. همان طور که قبلاً گفته شد در موقعیکه یک دستورالعمل وجود دارد می‌توانیم علامت‌های {} را قرار ندهیم اما گذاشتن آنها خطا محسوب نمی‌شود.

ایجاد کردن حلقه بی نهایت با استفاده از دستور while بصورت زیر است:

```
while (1){
دستورالعمل‌ها;
}
```

۴. دستور حلقه شرطی do – while :

این دستور نیز مانند دستور قبلی است با این تفاوت که، ابتدا اجازه تکرار یک بار حلقه را می‌دهد و در پایان حلقه، شرط را بررسی می‌کند. فرم این دستور بصورت زیر است:

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

مثال ۲-۱۹ :

```
unsigned char i,sw,y;
do{
i++;
y=sw*i;
}while(i<10);
```

در مثال ۲-۱۹ ابتدا برنامه وارد حلقه شده و متغیر شرط حلقه یک واحد افزایش پیدا می‌کند و متغیر sw در i ضرب شده و حاصل آن در y قرار می‌گیرد و شرط در پایان حلقه تست می‌شود اگر شرط درست بود دوباره حلقه تکرار می‌شود و در غیر اینصورت از حلقه خارج می‌گردد.

ایجاد کردن حلقه بی نهایت با استفاده از دستور do – while بصورت زیر است:

```
do {
دستورالعمل‌ها;
}while (1);
```

۵. دستور break :

این دستور برای شکستن و خارج شدن بدون شرط از حلقه می‌باشد. هرگاه برنامه به این دستور برسد از ساختار حلقه (منظور علامت {}) خارج شده و برنامه از اولین دستور بعد از ساختار حلقه ادامه می‌یابد. معمولاً دستور break همراه با دستور switch استفاده می‌شود.

۶. دستور switch :

یکی از دستورات مهم زبان C دستور مقایسه‌ای switch می‌باشد. این دستور یک عبارت (متغیر) را با یک سری از اعداد ثابت مقایسه می‌کند و عمل خاصی را مطابق مقایسه انجام شده صورت می‌دهد. در این دستور عبارت (متغیر) با مقدارهای ثابت قرار داده شده مقایسه می‌شود و اگر با مقداری که جلوی دستور case قرار می‌گیرد برابر بود، بعد از انجام دستورالعمل آن، با استفاده از دستور break سریعاً از ساختار دستور switch خارج می‌گردد.

فرم کلی این دستور بصورت زیر است:

```
switch (عبارت) {
case ۱ مقدار :
دستورالعمل‌های اول;
break;
case ۲ مقدار :
دستورالعمل‌های دوم;
break;
default:
دستورالعمل‌های پیش فرض;
}
```

در دستور switch، عبارت فقط با مقادیر ثابت مقایسه می‌شود. بنابراین نمی‌توان در جلوی دستور case از متغیر استفاده کرد. همچنین در این دستور می‌توانیم default را قرار ندهیم که در این صورت اگر عبارت با هیچ یک از مقادیر ثابت جلوی دستورات case برابر نباشد از ساختار دستور switch خارج می‌شود. اگر در یک case از دستور break استفاده نشود، با مقدار قسمت بعدی or می‌شود.

مثال ۲-۲۰:

```
unsigned char x,z;
switch(x){
case '1' : z=10;
break;
case 26 : z=20;
break;
default : z=50;
}
```

در مثال ۲-۲۰ مقدار x با عدد اسکی 1 مقایسه می‌شود که اگر درست باشد z=10 می‌شود و از دستور switch خارج می‌شود ولی اگر برابر نبود با عدد 26 دسیمال مقایسه می‌شود که اگر درست باشد z=20 خواهد شد و از دستور switch خارج می‌شود اما اگر با هیچ یک از مقادیر caseها برابر نبود z به طور پیش فرض برابر 50 می‌شود و از switch خارج می‌شود.

۷. دستور goto:

این دستور برای پرش به یک برچسب محلی در داخل یک تابع می‌باشد. فرم آن بصورت زیر است:

```
goto  برچسب ;
```

مثال ۲-۲۱:

```
Loop:
دستورالعمل‌ها;
goto  Loop;
```

۸. دستور continue:

هرگاه خط برنامه به این دستور برسد برنامه به ابتدای حلقه تکرار، پرش می‌کند و شرط بررسی می‌شود که اگر نادرست باشد حلقه خاتمه می‌یابد. فرم این دستور بصورت زیر است:

```
continue;
```

مثال ۲-۲۲:

```
unsigned char x=1,i,n;
while(x){
   i++;
   if(n==10){
    x=0;
    continue;
  }
}
```

در مثال ۲-۲۲ برنامه وارد حلقه می‌شود که شرط آن صفر نبودن x است. متغیر i در هر مرحله تکرار حلقه، یک واحد افزایش پیدا می‌کند و متغیر n نیز که در قسمت دیگری مثل وقفه‌ها مقداردهی می‌شود مورد تست قرار می‌گیرد که اگر برابر با عدد 10 بود x=0 شده و به دستور continue رسیده و

خط برنامه به ابتدای حلقه پرش کرده و شرط مورد ارزیابی قرار می‌گیرد و چون شرط برقرار نیست برنامه از حلقه خارج می‌گردد.

۹. دستور typedef :

با استفاده از این دستور ما می‌توانیم برای داده‌ها در زبان C یک نام جدید تعریف کنیم و نوع داده متغیرها را با نام جدید تعریف کنیم. فرم استفاده از این دستور بصورت زیر است:

typedef نام قدیمی نوع داده نام جدید نوع داده ;

مثال ۲-۲۳ :

```
typedef unsigned int 2byte;
```

اگر بصورت مثال ۲-۲۳ ابتدای برنامه، نامی برای نوع داده در نظر گرفته شود می‌توان یک متغیر ۱۶ بیتی بدون علامت را به این شکل تعریف کرد:

```
2byte  x;
```

توابع در زبان C

در حقیقت یک برنامه به زبان C از یک تابع اصلی به نام main و چند تا تابع فرعی با هر نام مجازی تشکیل شده است. تمام دستوراتی که به شما آموختیم در قالب بدنه یک تابع می‌توانند استفاده شوند و استفاده از دستورات در خارج از یک تابع، معنی و مفهومی ندارند. تابع، زیر برنامه‌ای است که دستورات و فراخوانی توابع دیگر در آن وجود داشته و عمل خاصی را انجام می‌دهد.

هر برنامه‌ای در زبان C دارای یک تابع اصلی است. منظور از تابع اصلی، تابع main یک برنامه می‌باشد. در هنگامی که یک برنامه آغاز می‌شود اولین تابعی که اجرا می‌شود تابع main می‌باشد. در صورتی که تابع اصلی با نام main نداشته باشیم نرم‌افزار کامپایلر، اعلام خطا می‌کند. به خاطر بسپارید که تابع اصلی مانند ریشه یک درخت می‌ماند، بنابراین این تابع می‌تواند انشعاب داشته باشد و دیگر توابع را فراخوانی کند ولی نمی‌توان از توابع فرعی تابع اصلی را فراخوانی کرد.

توجه

توابع فرعی می‌توانند همدیگر را فراخوانی کنند به شرط آنکه تابعی که می‌خواهد فراخوانی شود از قبل به کامپایلر معرفی شده باشد که برای این کار دو راه وجود دارد یکی اینکه تابع فرعی که می‌خواهد فراخوانی شود در برنامه نوشتاری بالاتر از تابعی که فراخوانی می‌خواهد در آن انجام شود بیاید و دیگر اینکه توابع در ابتدای برنامه معرفی شوند. با برنامه‌های متعددی که در این کتاب وجود دارد شما به راحتی این اصل را درک خواهید کرد. البته در برنامه‌های این کتاب بیشتر از روش اول استفاده شده است.

فرم کلی تعریف تابع بصورت زیر است:

کلاس ذخیره‌سازی تابع نوع برگشتن تابع نام تابع (آرگومان‌های تابع) {

دستورالعمل‌ها ;

```
return    متغیر یا عدد;
}
```

همان طور که قبلاً در مورد تعریف کلاس ذخیره‌سازی متغیرها بحث کردیم در تعریف توابع نیز می‌توان کلاس ذخیره‌سازی را تعریف کرد اما الزامی نیست. همچنین می‌توان نوع داده برگشتی توسط یک تابع را مشخص کرد. برگشت یک تابع یعنی متغیر یا مقدار عددی که با دستور return تعیین می‌گردد. در واقع توابع مانند یک فرمول ریاضیاتی می‌باشند که در نهایت پس از انجام اعمال خاصی در برنامه، یک حاصلی را نیز به همراه خواهند داشت. اگر تابعی نوع داده برگشتی آن تعریف نشود، نوع داده پیش فرض کامپایلر می‌باشد که معمولاً از نوع int است. اگر تابعی که استفاده می‌کنیم دارای برگشت نباشد نوع برگشتن آن را با اشاره‌گر void تهی می‌کنیم. نام تابع نیز اختیاری است و هر نامی که مجاز باشد (حروف لاتین a تا z، A تا Z و اعداد 0 تا 9 به شرط آنکه ابتدای نام نباشد و یا استفاده از علائمی نظیر آندر لاین "_") را می‌توان قرار داد. اگر یک تابع در داخل پرانتز خود متغیرهایی داشته باشد، آرگومان یا پارامتر تابع نامیده می‌شود و تابع می‌تواند ضمن فراخوانی، مقادیری را نیز به عنوان پارامتر ورودی به همراه داشته باشد. اگر یک تابع دارای آرگومان نباشد در داخل پرانتز تابع، کلمه کلیدی void یا خالی گذاشته می‌شود. برای فراخوانی یک تابع نیز از نام تابع با علامت پرانتز ( ) که به ; ختم می‌شود استفاده می‌کنیم.

مثال ۲-۲۴: یک تابع فرعی با نام display تعریف کرده و آن را فراخوانی کنید.

```
void   display( ){
دستورالعمل‌ها;
}
void  main( ){
display( );
دستورالعمل‌ها;
}
```

همان طور که گفته شد توابع قبل از فراخوانی می‌بایست تعریف شوند حال اگر شما از فایل‌های کتابخانه‌ای استفاده کنید در واقع در داخل این فایل‌ها توابعی که عمل خاصی را انجام می‌دهند تعریف شده است و ما بعد از معرفی فایل کتابخانه‌ای در برنامه، می‌توانیم توابع آن را نیز فراخوانی کنیم که در ادامه مطالب همین فصل به توابع کتابخانه‌ای نیز خواهیم پرداخت.

مثال ۲-۲۵: برنامه یک شمارنده 0 تا 9 با سون سگمنت کاتد مشترک را بنویسید؟

```
#include <mega16.h>          //معرفی میکروکنترلر استفاده شده
#include <delay.h>           //معرفی کتابخانه تأخیر زمانی
unsigned char i;             //for تعریف متغیر ۸ بیتی برای کانتر حلقه
unsigned char mask(unsigned  char num) {   //تابع برگشتی کد سون سگمنت
switch(num) {                //مقایسه مقدار متغیر با اعداد زیر
 case 0 : return 0x3f;       //کد معادل عدد صفر برای سون سگمنت کاتد مشترک
```

```
 case 1 : return 0x06;     //کد معادل عدد یک برای سون سگمنت کاتد مشترک
 case 2 : return 0x5b;     //کد معادل عدد دو برای سون سگمنت کاتد مشترک
 case 3 : return 0x4f;     //کد معادل عدد سه برای سون سگمنت کاتد مشترک
 case 4 : return 0x66;     //کد معادل عدد چهار برای سون سگمنت کاتد مشترک
 case 5 : return 0x6d;     //کد معادل عدد پنج برای سون سگمنت کاتد مشترک
 case 6 : return 0x7d;     //کد معادل عدد شش برای سون سگمنت کاتد مشترک
 case 7 : return 0x07;     //کد معادل عدد هفت برای سون سگمنت کاتد مشترک
 case 8 : return 0x7f;     //کد معادل عدد هشت برای سون سگمنت کاتد مشترک
 case 9 : return 0x6f;     //کد معادل عدد نه برای سون سگمنت کاتد مشترک
 default: return 0x00;     //کد پیش فرض
 }
}
void main(){                          //تابع اصلی برنامه
PORTA=0x00;                           //مقدار اولیه پورت متصل به سون سگمنت را صفر قرار می‌دهیم
DDRA=0xff;                            //تعیین پورت A به عنوان خروجی
while(1){                             //ایجاد کردن حلقه بی نهایت برای تکرار شمارش
for(i=0 ; i<=9 ; i++){                //ایجاد یک حلقه که ۱۰ مرتبه تکرار می‌شود
PORTA=mask(i);   //برگشت تابع کد معادل سون سگمنت کانتر حلقه بوده و به خروجی ارسال می‌شود
delay_ms(1000);                       //تأخیر زمانی ۱ ثانیه برای شمارش
  }
 };
}
```

اگر در درس مدارهای منطقی سون سگمنت را آموزش ندیده‌اید به فصل چهارم کتاب مراجعه کنید. در مثال ۲-۲۵ ابتدا برنامه وارد تابع اصلی main می‌شود و پس از تنظیم پورت متصل به سون سگمنت، وارد یک حلقه بی نهایت می‌شود و یک حلقه for جهت شمارش و تغییر مقدار آرگومان تابع mask که فراخوانی می‌شود، استفاده شده است. در واقع مقدار i در موقع فراخوانی در متغیر num کپی می‌شود. متغیر num به عنوان پارامتر یا آرگومان تابع معرفی شده است. مقدار num در دستور switch قرار می‌گیرد و مطابق با هر یک از مقادیر ثابت که برابر بود یک کد معادل سون سگمنت کاتد مشترک را بر می‌گرداند. این مقدار برگشتی در PORTA که به سون سگمنت کاتد مشترک وصل شده است فرستاده می‌شود. مقدار داده برگشتی ۸ بیتی بدون علامت می‌باشد.

آرایه‌ها

به متغیرهای به هم پیوسته که یکجا تعریف می‌شوند، آرایه گفته می‌شود و نحوه‌ی تعریف کردن آنها مشابه متغیرها می‌باشد. آرایه‌ها می‌توانند یک بعدی یا چند بعدی بصورت ماتریسی باشند.

فرم کلی تعریف آرایه یک بعدی بصورت زیر است:

نوع داده نام آرایه[اندازه متغیرها];

نوع داده می‌تواند یکی از انواع داده باشد و نام آرایه نیز می‌تواند یک نام مجاز از حروف لاتین بوده و تعداد عضوهای موجود در آرایه نیز می‌تواند تعریف شوند. ضمناً برای دسترسی به اعضای آرایه می‌بایست نام آرایه به اضافه شماره عضو در بین دو علامت [ ] قرار بگیرد.

مثال ۲-۲۶: یک آرایه تک بعدی با چهار عضو تعریف کنید و به آنها مقدار بدهید؟

```
unsigned char code[4];          //تعریف یک آرایه یک بعدی با ۴ عضو
code[0]=0x10;                   //مقداردهی به عضو اول
code[1]=0xaf;                   //مقداردهی به عضو دوم
code[2]=0x19;                   //مقداردهی به عضو سوم
code[3]=0x66;                   //مقداردهی به عضو چهارم
```

مثال ۲-۲۷: آرایه قبلی را تعریف کنید و همان لحظه اول مقداردهی را انجام دهید؟

```
unsigned char code[4]={0x10,0xaf,0x19,0x66};
```

مثال ۲-۲۸: برنامه یک شمارنده 0 تا 9 را این بار با استفاده از آرایه ذخیره شده در حافظه ثابت میکروکنترلر بنویسید؟

```
#include <mega16.h>                 //معرفی کتابخانه میکروکنترلر استفاده شده
#include <delay.h>                  //معرفی کتابخانه تأخیر زمانی
flash unsigned char display[]={     //کدهای سون سگمنت کاتد مشترک در حافظه ثابت
0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
void main(){                        //تابع اصلی برنامه
unsigned char i;                    //تعریف متغیر ۸ بیتی برای کانتر حلقه for
PORTA=0x00;                         //مقدار اولیه پورت متصل به سون سگمنت را صفر قرار می‌دهیم
DDRA=0xff;                          //تعیین پورت A به عنوان خروجی
while(1){                           //ایجاد کردن حلقه بی نهایت برای تکرار شمارش
for(i=0 ; i<=9 ; i++){              //ایجاد یک حلقه که ۱۰ مرتبه تکرار می‌شود
PORTA=display[i];                   //کانتر حلقه اندیس عضو آرایه را تعیین کرده و به خروجی ارسال می‌کنیم
delay_ms(1000);                     //تأخیر زمانی ۱ ثانیه برای شمارش
  }
 };
}
```

در مثال ۲-۲۸ ابتدا در تابع main متغیر محلی i تعریف می‌شود و پورت A به عنوان خروجی تنظیم می‌گردد و برنامه وارد حلقه بی نهایت while می‌شود و حلقه for تشکیل می‌گردد در هر بار که حلقه for تکرار می‌شود شمارنده حلقه یعنی i افزایش می‌یابد و به ترتیب اعضای آرایه display را به پورت A ارسال می‌کند و در اینجا تأخیر حلقه یک ثانیه است هر وقت i=10 شود شرط حلقه for بر قرار نخواهد بود و وارد حلقه بی نهایت می‌شود و مجدداً وارد حلقه for می‌شود.

فرم کلی تعریف آرایه دو بُعدی بصورت زیر است:

نوع داده نام آرایه[اندازه بُعد ۱][اندازه بُعد ۲];

اندازه بُعد اول تعداد سطرها و اندازه بُعد دوم تعداد ستون‌های آرایه را تعیین می‌کند که در واقع یک ماتریس را بوجود می‌آورد. ترتیب نوشتن مقدار اولیه آرایه‌های دو بعدی باید رعایت شود.

مثال ۲-۲۹: یک آرایه دو بعدی تعریف کنید و به اعضای آن دسترسی پیدا کنید.

```
unsigned char x,j;
unsigned char matrix[2][3]={{'0','1','2'},
                            {'3','4','5'}};
x= matrix[1][2];
j= matrix[0][0];
```

در مثال ۲-۲۹ مقدار قرار گرفته در متغیر x='5' خواهد بود چون سطر دوم و ستون سوم انتخاب شده است و در متغیر j نیز عدد اسکی '0' قرار می‌گیرد. اعداد می‌توانند در آرایه‌ها با هر مبنایی باشند و لزومی ندارد که حتماً اعداد اسکی باشند و دقت کنید اندیس اعضا از صفر شروع می‌شود.

رشته‌ها

در واقع رشته‌ها همان آرایه‌هایی هستند که از کاراکترهای به هم پیوسته تشکیل شده‌اند و بین دو علامت " " تعریف می‌گردند. رشته‌ها در برنامه‌نویسی فقط بصورت تک بُعدی تعریف می‌شوند.

مثال ۲-۳۰: یک رشته بدون تعیین اعضا تعریف کنید.

```
char name[]="alvandi";
```

هر رشته به یک کاراکتر تهی null (برابر 0 دسیمال یا '\0' اسکی) ختم می‌شود. کاراکتر تهی نشان دهنده پایان رشته می‌باشد بنابراین اگر آرایه‌هایی داشته باشیم که اندازه آن ۸ باشد فقط از ۷ مکان آن استفاده می‌شود که آخری یک کاراکتر تهی می‌باشد.

ساختمان (ساختارها)

ساختارها مجموعه‌ای از اعضای متغیرها با نوع داده‌های مختلف، تحت عنوان یک نام می‌باشند. تاکنون هر متغیری را که تعریف می‌کردیم می‌بایست نوع داده آن نیز ذکر شود و اگر متغیری با نوع داده دیگری بخواهد تعریف شود باید به طور جداگانه تعریف شود. اما توسط ساختارها ما می‌توانیم برای اعضای ساختمان خود یک نام کلی در نظر بگیریم.

فرم کلی تعریف ساختار بصورت زیر است:

```
struct <نام ساختمان>
{ اعضای ساختمان ;
} اسامی متغیرهای ساختمان ;
```

کلمه کلیدی struct نشان‌دهنده تعریف ساختمان می‌باشد و نام ساختمان می‌تواند از حروف لاتین باشد و اعضای ساختمان نیز می‌توانند متغیرهایی با هر نوع داده‌ای باشند و اسامی متغیرهای ساختمان نیز می‌توانند با هر نامی از حروف لاتین تعریف شوند. در واقع این اسامی متغیرها هست که به اعضای داخل ساختمان دسترسی دارند.

مثال ۲-۳۱:

```
struct time {
char contrl;
unsigned int hour,minute,seconds;
}par;
...
par.contrl=0x12;
par.minute=0x60;
```

برای دسترسی به اعضای ساختار از علامت "." استفاده می‌کنیم و همچنین اگر اشاره‌گری از نوع ساختمان تعریف شده باشد با استفاده از علامت "->" قابل دسترسی است.

مثال ۲-۳۲:

```
struct time *ptr;
ptr->seconds=30;
```

یونیون‌ها (اتحادها)

اتحادها نیز مانند ساختارها یک نوع داده گروهی می‌باشند. تفاوت آنها با ساختارها در این است که از مکان‌های حافظه اختصاص یافته به اتحادها، بین اعضای گروه به طور مشترک استفاده می‌شود.

فرم کلی تعریف اتحادها بصورت زیر است:

```
union   نام اتحاد {
اعضای اتحادها ;
} اسامی متغیرهای اتحادها ;
```

مثال ۲-۳۳:

```
union save_R{
char x,y;
int z;
}s_data;
```

نحوه‌ی دسترسی به اعضای اتحادها نیز مانند ساختارها می‌باشد. همان طور که گفتیم میزان فضای اشغال شده توسط اتحادها کمتر از ساختارها می‌باشد. در مثال ۲-۳۳ دو متغیر ۸ بیتی x و y و یک متغیر ۱۶ بیتی داریم. پس اگر از نوع ساختمان باشد در مجموع ۳۲ بیت حافظه مورد نیاز است. ولی چون از اتحادها استفاده کردیم، فقط به ۱۶ بیت نیاز داریم و از طرفی چون، برنامه یک مکان ۱۶ بیتی را برای اعضای گروه به اشتراک می‌گیرد، در هر لحظه می‌توان فقط از یکی از اعضای گروه استفاده کرد. در اتحادها بزرگترین متغیر، تعیین کننده میزان فضای حافظه برای یک اتحاد می‌باشد.

شمارش‌ها

نوع داده شمارشی، برای تعریف یک مجموعه متناهی جهت نام‌گذاری یا شماره‌گذاری است.

فرم کلی تعریف نوع داده شمارشی بصورت زیر است:

```
enum نام شمارش{
اعضای شمارش ;
اسامی متغیرهای شمارش } ;
```

مثال ۲-۳۴:

```
enum color{
red;
bulo;
green;
}color1,color2;
```

در این مثال به عنصر اولی یعنی red عدد ۱ و به دومی یعنی bulo عدد ۲ و به سومی عدد ۳ نسبت داده می‌شود و همین طور اگر ما کلی عنصر در یک شمارشگر قرار دهیم اعداد صحیح به آنها نسبت داده می‌شود مگر اینکه کاربر شماره عنصر را تعیین کند مثلاً green=6 ، آنگاه به عنصر سوم عدد ۶ نسبت داده می‌شود.

۲-۳ توابع کتابخانه‌ای استاندارد

کتابخانه‌های ctype، stdlib، string، math جزء کتابخانه‌های استاندارد در زبان C می‌باشند و کامپایلر CodeVisionAVR نیز دارای این کتابخانه‌ها می‌باشد. توابع کتابخانه‌ای در مسیر نصب نرم‌افزار در پوشه INC و LIB قرار دارند که ما در صورت نیاز، می‌بایست آنها را در ابتدای برنامه معرفی کنیم و از توابع آمادهٔ آنها استفاده کنیم. برخی از این توابع عبارتند از :

```
STDIO,STDARG,SPI,SLEEP,SETJMP,PCF8583,MEM,LM75,LCD4X40
LCD,io,GRAY,BCD,43USB355,86RF401,DS1307,DELAY,mega16
TINY13,90S8535
```

تذکرات مهم :

۱. قبل از استفاده از هر تابع کتابخانه‌ای با دستور #include<نام کتابخانه.h> باید آن کتابخانه در ابتدای برنامه معرفی گردد.
۲. توابعی که قبل از آنها کلمه کلیدی **void** وجود دارد موقع فراخوانی نباید **void** را تایپ کنید و همچنین برای توابع برگشتی نیز نباید نوع داده برگشتی موقع فراخوانی تایپ گردد.
۳. توابعی که برگشت‌پذیر هستند و مقداری که برمی‌گردانند ممکن است علامت‌دار یا بدون علامت باشد که باید در نظر گرفته شود.
۴. توابعی که پارامتر ورودی دریافت می‌کنند خواه برگشت‌پذیر یا برگشت‌ناپذیر باشند باید به آنها متناسب با پذیرش مقدار، آرگومان ارسال گردد.
۵. توابعی که پارامتر ورودی آنها رشته یا آرایه می‌باشد موقع ارسال آرگومان نباید علامت کروشه [ ] تایپ گردد و فقط نام رشته یا آرایه را می‌نویسیم.

۶. توجه کنید موقع فراخوانی باید در انتهای هر تابع ; را قرار دهید.
۷. تمامی توابع باید با حروف کوچک فراخوانی شوند.
۸. منظور از علامت * در برخی از توابع، اشاره‌گر رشته‌ای می‌باشد.

کتابخانه ctype.h

unsigned char isalnum(**char** c)

اگر c یک کاراکتر پارامتر ورودی، یکی از حروف الفبایی-رقمی لاتین (شامل حروف و ارقام) باشد، مقدار 1 را بر می‌گرداند و در غیر اینصورت مقدار 0 را بر می‌گرداند.

unsigned char isalpha(**char** c)

اگر c یک کاراکتر پارامتر ورودی، یکی از حروف الفبای لاتین (فقط شامل حروف) باشد مقدار 1 را بر می‌گرداند و در غیر اینصورت مقدار 0 را بر می‌گرداند.

unsigned char isascii(**char** c)

اگر c یک کاراکتر پارامتر ورودی باشد. اگر این کاراکتر یکی از کاراکترهای اسکی در محدوده اعداد (۰ تا ۱۲۷) باشد مقدار 1 و در غیر اینصورت مقدار 0 را بر می‌گرداند.

unsigned char iscntrl(**char** c)

اگر c یک کاراکتر کنترلی به عنوان پارامتر ورودی تابع باشد. اگر این کاراکتر ورودی در محدوده اعداد (۰ تا ۳۱ یا ۱۲۷) باشد مقدار 1 و در غیر اینصورت مقدار 0 را بر می‌گرداند.

unsigned char isdigit(**char** c)

اگر c یک کاراکتر پارامتر ورودی یکی از اعداد دسیمال ۰ تا ۹ باشد این تابع مقدار 1 و در غیر اینصورت مقدار 0 را بر می‌گرداند.

unsigned char islower(**char** c)

اگر c یک کاراکتر پارامتر ورودی یکی از حروف کوچک لاتین a تا z باشد، این تابع مقدار 1 و در غیر اینصورت مقدار 0 را بر می‌گرداند.

unsigned char isprint(**char** c)

اگر c یک کاراکتر پارامتر ورودی یکی از کاراکترهای قابل چاپ در محدوده (۳۲ تا ۱۲۷) باشد این تابع مقدار 1 و در غیر اینصورت مقدار 0 را بر می‌گرداند.

unsigned char ispunct(**char** c)

اگر c یک کاراکتر پارامتر ورودی یکی از کاراکترهای علائمی مانند (؟، ! و ...) باشد این تابع مقدار 1 و در غیر اینصورت مقدار 0 را بر می‌گرداند.

unsigned char isspace(**char** c)

اگر c یک کاراکتر پارامتر ورودی، یکی از کاراکترهای فضای خالی ' ' و یا Carrige return ("\r\n") باشد. این تابع مقدار 1 و در غیر اینصورت مقدار 0 را بر می‌گرداند.

unsigned char isupper(**char** c)

اگر c یک کاراکتر پارامتر ورودی، یکی از حروف بزرگ لاتین A تا Z باشد. این تابع مقدار ۱ و در غیر اینصورت مقدار 0 را بر می‌گرداند.

unsigned char isxdigit(**char** c)

اگر c یک کاراکتر پارامتر ورودی، یکی از اعداد هگزاد دسیمال 0 تا F باشد. این تابع مقدار ۱ و در غیر اینصورت مقدار 0 را بر می‌گرداند.

char toascii(**char** c)

این تابع کاراکتر c را به عنوان پارامتر ورودی می‌گیرد و معادل اسکی آن را بر می‌گرداند.

unsigned char toint(**char** c)

این تابع کاراکتر c را به عنوان پارامتر ورودی بر حسب هگزاد دسیمال می‌گیرد و معادل رقمی بر حسب دسیمال اعداد 0 تا 15 را بر می‌گرداند.

char tolower(**char** c)

این تابع کاراکتر c را به عنوان پارامتر ورودی می‌گیرد و اگر این کاراکتر حرف بزرگ باشد آن را به حرف کوچک تبدیل می‌کند و آن را بر می‌گرداند، در غیر اینصورت خود کاراکتر c را بر می‌گرداند.

char toupper(**char** c)

این تابع کاراکتر c را به عنوان پارامتر ورودی می‌گیرد و اگر این کاراکتر حرف کوچک باشد آن را به حرف بزرگ تبدیل می‌کند و آن را بر می‌گرداند، در غیر اینصورت خود کاراکتر c را بر می‌گرداند.

کتابخانه stdlib.h

int atoi(**char** *str)

یک متغیر رشته‌ای به عنوان پارامتر ورودی می‌پذیرد و آن را به عدد صحیح تبدیل می‌کند.

long int atol(**char** *str)

این تابع یک متغیر رشته‌ای به عنوان پارامتر ورودی می‌پذیرد و آن را به عدد صحیح بلند می‌کند.

void itoa(**int** n, **char** *str)

این تابع یک متغیر عدد صحیح n و یک متغیر رشته‌ای str را به عنوان پارامتر ورودی می‌گیرد و عدد صحیح را تبدیل به کاراکترهای اسکی کرده و در متغیر رشته‌ای str ذخیره می‌کند.

void ltoa(**long int** n, **char** *str)

این تابع یک متغیر عدد صحیح بلند n و یک متغیر رشته‌ای str را به عنوان پارامتر ورودی می‌گیرد و عدد صحیح بلند را تبدیل به کاراکترهای اسکی کرده و در متغیر رشته‌ای str ذخیره می‌کند.

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

این تابع یک متغیر اعشاری n و یک متغیر عدد صحیح دسیمال و یک متغیر رشته‌ای str را به عنوان پارامتر ورودی می‌گیرد و متغیر اعشاری را تبدیل به کاراکترهای اسکی کرده و در متغیر رشته‌ای str

ذخیره می‌کند. مقدار متغیر decimals دقت ارقام اعشار برای تبدیل به کاراکتر اسکی را تعیین می‌کند.

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

این تابع یک متغیر اعشاری n و یک متغیر عدد صحیح دسیمال و یک متغیر رشته‌ای str را به عنوان پارامتر ورودی می‌گیرد و متغیر اعشاری را تبدیل به نماد علمی معادل کاراکترهای اسکی کرده و در متغیر رشته‌ای str ذخیره می‌کند. مقدار متغیر decimals دقت ارقام اعشار را تعیین می‌کند. به طور مثال با ۲ رقم دقت اعشار، عدد 12.3598×10^{-5} را تبدیل به رشته "12.35e-5" می‌نماید.

float atof(**char** *str)

یک متغیر رشته عددی را به عنوان پارامتر ورودی می‌پذیرد و آن را به عدد اعشاری تبدیل می‌کند.

int rand (**void**)

این تابع در لحظه فراخوانی یک عدد صحیح از بین ارقام ۰ تا ۳۲۷۶۷ را به طور تصادفی بر می‌گرداند.

void srand(**int** seed)

این تابع مانند تابع قبلی عمل می‌کند با این تفاوت که رقم شروع برای انتخاب رقم تصادفی را به عنوان پارامتر ورودی می‌گیرد و سرانجام یک رقم تصادفی بر می‌گرداند.

کتابخانه math.h

unsigned int abs(**int** x)

این تابع یک متغیر عدد صحیح به عنوان پارامتر ورودی می‌گیرد و قدر مطلق آن را بر می‌گرداند.

unsigned long labs(**long** int x)

یک متغیر عدد صحیح بلند به عنوان پارامتر ورودی می‌گیرد و قدر مطلق آن را بر می‌گرداند.

float fabs(**float** x)

این تابع یک متغیر عدد اعشاری به عنوان پارامتر ورودی می‌گیرد و قدر مطلق آن را بر می‌گرداند.

int max(**int** a, **int** b)

تابع دو متغیر ۱۶ بیتی a و b را به عنوان پارامتر ورودی می‌گیرد و مقدار بزرگتر را بر می‌گرداند.

int min(**int** a, **int** b)

تابع دو متغیر ۱۶ بیتی a و b را به عنوان پارامتر ورودی می‌گیرد و مقدار کوچکتر را بر می‌گرداند.

signed char sign(**int** x)

این تابع یک متغیر عدد صحیح را به عنوان پارامتر ورودی می‌گیرد و اگر متغیر منفی باشد -1، صفر باشد 0 و مثبت باشد +1 را بر می‌گرداند.

signed char fsign(**float** x)

این تابع یک متغیر عدد اعشاری را به عنوان پارامتر ورودی می‌گیرد و اگر متغیر منفی باشد -1، صفر باشد 0 و مثبت باشد +1 را بر می‌گرداند.

float sqrt(**float** x)

یک متغیر عدد اعشاری را به عنوان پارامتر ورودی می‌گیرد و مقدار $\sqrt{x}$ را بر می‌گرداند.

float exp(**float** x)

یک متغیر عدد اعشاری x را به عنوان پارامتر ورودی می‌گیرد و حاصل e^x را بر می‌گرداند.

float log(**float** x)

یک متغیر عدد اعشاری را به عنوان پارامتر ورودی می‌گیرد و حاصل Ln(x) را بر می‌گرداند.

float log10(**float** x)

یک متغیر عدد اعشاری را به عنوان پارامتر ورودی می‌گیرد و لگاریتم در مبنای ۱۰ را بر می‌گرداند.

float pow(**float** x, **float** y)

این تابع دو متغیر اعشاری x و y را به عنوان پارامتر ورودی می‌گیرد و مقدار x^y را بر می‌گرداند.

float sin(**float** x)

متغیر اعشاری را به عنوان پارامتر ورودی می‌گیرد و سینوس آن را بر حسب رادیان بر می‌گرداند.

float cos(**float** x)

متغیر اعشاری را به عنوان پارامتر ورودی می‌گیرد و کسینوس آن را بر حسب رادیان بر می‌گرداند.

float tan(**float** x)

متغیر اعشاری را به عنوان پارامتر ورودی می‌گیرد و تانژانت آن را بر حسب رادیان بر می‌گرداند.

float asin(**float** x)

یک متغیر اعشاری را به عنوان پارامتر ورودی می‌گیرد و معکوس سینوس آن عدد را بر حسب رادیان بر می‌گرداند.

float acos(**float** x)

یک متغیر اعشاری را به عنوان پارامتر ورودی می‌گیرد و معکوس کسینوس آن عدد را بر حسب رادیان بر می‌گرداند.

float atan(**float** x)

یک متغیر اعشاری را به عنوان پارامتر ورودی می‌گیرد و معکوس تانژانت آن عدد را بر حسب رادیان بر می‌گرداند.

کتابخانه string.h

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

این تابع رشته str2 را به انتهای رشته str1 متصل می‌کند.

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

این تابع رشته str2 که در حافظه ثابت قرار دارد را به انتهای رشته str1 متصل می‌کند.

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

این تابع حداکثر n کاراکتر از رشته str2 را به انتهای رشته str1 متصل می‌کند.

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

حداکثر n کاراکتر از رشته str2 که در حافظه ثابت قرار دارد را به انتهای رشته str1 متصل می‌کند.

```
char *strchr(char *str, char c)
```

این تابع رشته str را برای محل اولین وقوع کاراکتر ورودی c جستجو می‌کند و آن را به یک اشاره‌گر نسبت می‌دهد و در غیر اینصورت اشاره‌گر برابر کاراکتر Null (تهی) می‌شود.

```
char *strrchr(char *str, char c)
```

این تابع رشته str را برای محل آخرین وقوع کاراکتر ورودی c جستجو می‌کند و آن را به یک اشاره‌گر نسبت می‌دهد و در غیر اینصورت اشاره‌گر برابر کاراکتر Null (تهی) می‌شود.

```
signed char strcmp(char *str1, char *str2)
```

این تابع رشته str1 را با رشته str2 مقایسه می‌کند. اگر str1 > str2 مقدار بزرگتر از صفر و اگر str1 = str2 مقدار صفر و اگر str1 < str2 مقدار کوچکتر از صفر (عدد منفی) را بر می‌گرداند.

```
signed char strcmpf(char *str1, char flash *str2)
```

این تابع رشته str1 را واقع در حافظه SRAM با رشته str2 واقع در FLASH مقایسه می‌کند. اگر str1 > str2 ، مقدار بزرگتر از صفر و اگر str1 = str2 مقدار صفر و اگر str1 < str2 مقدار کوچکتر از صفر را بر می‌گرداند.

```
char *strcpy(char *dest, char *src)
```

این تابع رشته src را به رشته dest کپی می‌کند.

```
char *strcpyf(char *dest, char flash *src)
```

این تابع رشته src واقع در حافظه Flash را به رشته dest واقع در حافظه SRAM کپی می‌کند.

کتابخانه bcd.h

```
unsigned char bcd2bin(unsigned char n)
```

این تابع یک پارامتر ورودی n بر حسب کد BCD می‌گیرد و کد باینری معادل آن را بر می‌گرداند.

```
unsigned char bin2bcd(unsigned char n)
```

این تابع یک پارامتر ورودی n که می‌تواند در محدوده اعداد 0 تا 99 باشد را می‌گیرد و کد BCD معادل آن را بر می‌گرداند.

کتابخانه delay.h

توجه داشته باشید مقدار تأخیر توابع این کتابخانه به مقدار فرکانس کاری برنامه در نرم‌افزار کامپایلر که از مسیر Project/Configure/ C compiler تنظیم می‌شود، وابسته است.

```
void delay_us(unsigned int n)
```

این تابع یک عدد ثابت به عنوان پارامتر ورودی در محدوده اعداد 0 تا 65535 می‌گیرد و بر حسب میکرو ثانیه تأخیر زمانی ایجاد می‌کند. به طور مثال برای تأخیر 100µS داریم :

```
delay_us(100);
```

```
void delay_ms(unsigned int n)
```

این تابع یک متغیر عدد صحیح به عنوان پارامتر ورودی در محدوده اعداد 0 تا 65535 می‌گیرد و بر حسب میلی ثانیه تأخیر زمانی ایجاد می‌کند.

به طور مثال برای تأخیر یک ثانیه 1000ms داریم :

```
delay_ms(1000);
```

همچنین فرض کنید یک متغیر مانند TD=250 مفروض باشد :

```
delay_ms(TD);
```

در تابع فوق به مدت ۲۵۰ میلی ثانیه تأخیر ایجاد می‌شود.

نکته دیگری که در مورد تابع delay_ms() باید به آن اشاره کنیم این است که این تابع به طور اتوماتیک تایمر watchdog را هر 1ms توسط دستور اسمبلی wdr، Reset می‌کند. این ویژگی زمانی که در برنامه، تایمر watchdog را فعال کرده باشیم یک مزیت محسوب می‌شود.

تمرین‌های فصل ۲

۲-۱ مقادیر متغیرهای برنامه را در پایان، یعنی زمانی که برنامه وارد حلقه بی نهایت می‌شود، تعیین نمائید؟

```
#include <mega16.h>
unsigned char x=2,n=2,j=5,k,i,z;
void main(){
while (x-->0){
 for(k=0;k<=2*n-2;k++)
  for(i=2;i>0;i--,j++)
    z+=(x*i)+j;
 }
while (1);    //حلقه بی نهایت
}
```

۲-۲ یک دیتای چهار بیتی از نیبل (۴ بیت) بالایی پورت A بخوانید و به اندازه مقدار خوانده شده یک LED را روشن کرده و سپس آن را خاموش کنید؟

۲-۳ دو کلید فشاری با نام‌های Up و Down و یک LED به میکروکنترلر متصل کنید. حال برنامه چشمک زدن LED را طوری بنویسید که توسط کلیدها بتوان زمان خاموش و روشن شدن LED را کم یا زیاد کرد؟

۲-۴ برنامه‌ای بنویسید که معادله زیر را تحقق بخشد و حاصل معادله را بر روی ۹ عدد LED متصل به پورت‌های D و C نمایش دهد. در این حالت PD0 را بیت نهم فرض بگیرید. مقدار X معادله را به طور مکرر از نیبل (۴ بیت) بالایی پورت A بخوانید؟

$$Y = X^2 + 4X + 7$$

۲-۵ یک دیتای ۸ بیتی به طور مداوم از پورت B بخوانید و بررسی نمایید که اگر عدد زوج است

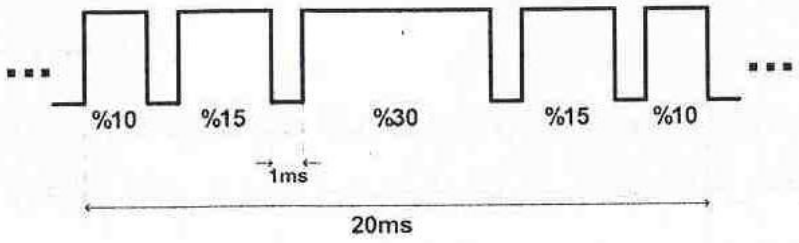
یک پالس مربعی متناوب 1KHZ تولید گردد و اگر عدد فرد بود پالس متوقف شود؟

۶-۲ برنامه‌ای بنویسید که عملکرد یک انکدر (رمزگذار) ۸ به ۳ را همراه با فعال‌ساز، شبیه‌سازی نماید؟

۷-۲ برنامه‌ای بنویسید که عملکرد یک RAM با ظرفیت 16×8bit را همراه با فعال‌ساز، شبیه‌سازی نماید؟

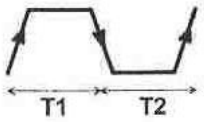
۸-۲ برنامه‌ای بنویسید که عملکرد یک Full Adder را شبیه‌سازی نماید؟

۹-۲ توسط توابع تأخیر زمانی یک پالس مربعی متناوب شبه سینوس تمام موج مانند پالس زیر با زمان تناوب ۲۰ میلی ثانیه ایجاد نمائید؟



۱۰-۲ با استفاده از توابع تأخیر زمانی برنامه‌ای بنویسید که یک موج مربعی با فرکانس ثابت 50HZ تولید نماید و با استفاده از دو کلید فشاری Up و Down بتوان دیوتی سایکل پالس را بین 10% تا 90% تغییر داد. شکل موج تولیدی را به گیت یک MOSFET متصل کنید و یک موتور DC به عنوان بار در نظر بگیرید؟

$$D.C=\frac{T1}{T1+T2}\times 100$$



۱۱-۲ برنامه‌ای بنویسید که توسط توابع تأخیر زمانی، دو پالس مربعی همزمان با فرکانس‌های 2HZ و 8HZ بر روی پایه‌های PC0 و PC1 ایجاد نماید؟

۱۲-۲ مسئله ۲-۱۱ را به ازای تولید پالس‌های مربعی 2HZ و 3HZ تکرار نمائید؟

۱۳-۲ برنامه‌ای بنویسید که توسط توابع تأخیر زمانی، سه پالس مربعی همزمان با فرکانس‌های 1KHZ، 2KHZ و 10KHZ به ترتیب بر روی پایه‌های PD0 ، PD1 و PD2 ایجاد نماید؟

۱۴-۲ برنامه‌ای بنویسید که بتوان توسط دو کلید فشاری Up و Down مقدار 0 تا 9 را تنظیم کرد و بصورت باینری بر روی چهار LED نمایش دهد؟

۱۵-۲ برنامه مسئله ۲-۱۴ را برای نمایش با سون سگمنت تک رقمی آند مشترک تکرار کنید؟

۱۶-۲ برنامه یک شمارنده صعودی 0 تا 9 با نمایشگر سون سگمنت را بنویسید که دارای کلید فشاری

Play/Pause باشد به گونه‌ای که با زدن آن شمارنده شروع و با زدن مجدد آن شمارش در حالت مکث(توقف موقت) قرار گیرد؟

۲-۱۷ برنامه‌ای بنویسید که در صورت بسته بودن یک کلید، یک موج 1KHZ و در صورت باز بودن یک موج 5KHZ بر روی PC0 ایجاد نماید؟

۲-۱۸ برنامه‌ای بنویسید که وضعیت دو کلید را تست کند و برحسب کد باینری خوانده شده، یک موج بر حسب کیلوهرتز تولید نماید به طور مثال اگر وضعیت کلیدها بصورت باینری "10" بود یک موج 2KHZ تولید گردد؟

۲-۱۹ برنامه‌ای بنویسید که در صورت بسته بودن یک کلید، یک موج 4KHZ و در صورت باز بودن یک موج 8KHZ بر روی PD2 ایجاد نماید و عدد موج تولیدی بر روی یک سون سگمنت تک رقمی نمایش داده شود؟

۲-۲۰ برنامه‌ای بنویسید که بتوان توسط یک کلید فشاری موج‌های 1K, 2K, 3K, 4K را انتخاب کرد و آن موج علاوه بر تولید شدن، بر روی سون سگمنت تک رقمی نمایش داده شود؟

۲-۲۱ برنامه‌ای بنویسید که توسط توابع تأخیر زمانی، یک موج مربعی با فرکانس 1KHZ با دیوتی سایکل 70% ایجاد نماید؟

۲-۲۲ برنامه یک شمارنده یک رقمی را بنویسد که وضعیت یک کلید را بخواند، در صورت بسته بودن کلید، شمارنده اعداد زوج و در صورت باز بودن کلید، اعداد فرد را شمارش کند؟