۱) برنامه‌نویسی به زبان C ویژه‌ی میکروکنترلرها – بخش اول

۱-۱) فایل سرآیند (Header File)

فایل سرآیند فایلی است که با پسوند h. ذخیره می‌شود و شامل اعلان توابع (نه تعریف آن‌ها) و تعریف ماکروهایی است که بین چندین فایل به اشتراک گذاشته می‌شوند. دو نوع فایل سرآیند وجود دارد: الف) فایل‌های سرآیندی که به همراه کامپایلر به صورت پیش‌فرض وجود دارند و ب) فایل‌های سرآیندی که توسط برنامه نویس نوشته می‌شوند. برای دسترسی به فایل‌های سرآیند از داخل برنامه، باید با استفاده از دستور پیش‌پردازنده‌ی #include فایل سرآیند مدنظر فراخوانی شود. به عنوان مثال با استفاده از دستور زیر فایل سرآیند stdio.h فراخوانی می‌شود.

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

فرخوانی فایل سرآیند معادل با کپی کردن تمام محتوای فایل درون برنامه‌ی اصلی است.

☑ بهتر است تمام ثوابت، ماکروها، متغیرهای همگانی و همچنین نمونه اولیه توابع درون یک فایل سرآیند ذخیره شوند و هر جا که لازم بود فایل سرآیند فراخوانی شود.

برای فراخوانی فایل‌های سرآیندی که به طور پیش‌فرض همراه کامپایلر می‌آیند (system header files) از #include <headerfilename.h> و برای فراخوانی فایل‌های سرآیند نوشته شده توسط کاربر از #include "headerfilename.h" استفاده می‌شود.

۱-۲) توضیحات (Comments)

توضیحات بخش مهمی از هر زبان برنامه نویسی هستند. توضیحات در کجای برنامه که قرار بگیرند توسط کامپایلر نادیده گرفته می‌شوند. توضیحات می‌توانند شفافیت کد نوشته شده را بالا ببرند به نحوی که کد نوشته شده توسط یک کاربر، به راحتی توسط سایر کاربران قابل بازخوانی و فهم باشد که این امر می‌تواند به عیب‌یابی کد کمک کند. در پروژه‌های بزرگ زمانی که یک کد از هزاران خط تشکیل شده است، توضیحات می‌توانند بسیار راه‌گشا باشند. به طور کلی می‌توان گفت توضیحات به فهم بهتر کد نوشته شده کمک می‌کنند و هم‌چنین فرآیند عیب یابی را سرعت می‌بخشند. دو ترکیب کلی (syntax) برای توضیحات در زبان C وجود دارد، الف) توضیحات چند خطی که کل توضیحات با یک */ شروع شده و با یک /* خاتمه می‌یابد.

```
/*
 * Author: Alireza Nami
 * Purpose: To show a comment that spans multiple lines.
 * Language:  C
 */
```

ب) توضیحات تک خطی که هر آن چه بعد از // نوشته شود توسط کامپایلر به عنوان کامنت یا توضیحات در نظر گرفته می‌شود.

```
#define PI 3.1416    // This constant is called PI
```

۱-۳) نمایش رشته و کاراکتر توسط توابع printf

تابع printf یک رشته‌ی خروجی که می‌تواند شامل کاراکترهای ASCII باشد را نمایش می‌دهد.

```
printf("hello world!”);
```

۱-۴) جملات فرار (Escape Sequence)

جملات فرار در زبان C مجموعه‌ای از کاراکترها هستند که به جای نمایش خودشان ، یک عمل خاص روی نمایشگر انجام می‌دهند. جملات فرار شامل دو کاراکتر یا بیشتر هستند. به عنوان مثال \n موجب انتقال نشانگر به ابتدای خط بعدی می‌شود.

Escape Sequence	Meaning
\b	Backspace
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\0	Null

جدول ۱- ۱ جملات فرار در زبان برنامه‌نویسی C

۱-۵) نوع داده (Data Types)

برای اعلان نوع یک متغیر از Data type استفاده می‌شود. در واقع نوع داده اندازه حافظه‌ی اشغال شده توسط آن متغیر یا تابع و همچنین نوع یک متغیر یا تابع را برای کامپایلر مشخص می‌کند. قبل از ذخیره‌ی یک متغیر یا تابع، برنامه نویس باید برای نوع داده‌ی آن تصمیم گیری کند. داده‌های دنیای واقعی که برای سیستم‌های کامپیوتری قابل تعریف باشند را می‌توان به چند دسته‌ی اعداد (صحیح یا حقیقی)، کاراکترها و رشته‌ها تقسیم بندی کرد. اما نوع داده در زبان برنامه نویسی C به دو دسته‌ی کلی تقسیم می‌شود:

داده‌های صحیح

که خود شامل چندین زیر شاخه می‌شود:

۱- char : برای ذخیره‌ی کاراکترهای اسکی و همچنین اعداد بازه‌ی (-128,127) استفاده می‌شود.

۲- short: برای ذخیره‌ی اعداد صحیح با طول ۲ بایت.

۳- int: برای ذخیری اعداد صحیح با طول ۲ تا ۴ بایت (بسته به نوع کامپایلر)

۴- long: برای ذخیره‌ی اعداد صحیح با طول ۴ تا ۸ بایت (بسته به نوع کامپایلر)

۵- long long: برای ذخیره‌ی اعداد صحیح با طول ۸ بایت

*** تمام نوع داده به صورت پیش فرض علامتدار یا signed در نظر گرفته می‌شوند.**

**** کلمه‌ی unsigned می‌تواند به انواع نوع داده افزوده شود. به این ترتیب اندازه‌ی نوع داده تغییری نمی‌کند اما تنها داده‌های مثبت قابل قبول می‌شوند.**

داده‌های اعشاری

اعداد خیلی خیلی بزرگ و یا خیلی خیلی کوچک مطابق استاندار ممیز شناور IEEE 754 ذخیره می‌شوند. ۲ شکل ذخیره‌سازی برای این اعداد وجود دارد.

Single precision

Sign (1bit)	Exponent (8 bits)	Significand (23 bits)

$$\mathbf{12345}(\boldsymbol{significand}) \times \mathbf{10^{-4}}\ (\boldsymbol{exponent})$$

به این نوع داده float گفته می‌شود که تا ۶ رقم اعشار دقت دارد.

Double precision

Sign (1bit)	Exponent (11 bits)	Significand (52 bits)

به این نوع داده Double گفته می‌شود که تا ۱۵ رقم اعشار دقت دارد.

*** دقت شود که نوع داده‌ی unsigned double و unsigned float وجود ندارند.**

**** برای به دست آوردن اندازه‌ی یک متغیر می‌توان از تابع sizeof(type) استفاده کرد که خروجی این تابع اندازه‌ی متغیر یا نوع داده به بایت است.**

```
sizeof(short);
```

۱-۶) تعریف متغیر (Variable Definition)

تعریف یک متغیر به کامپایلر می‌گوید که چه قدر فضا برای آن متغیر در نظر بگیرد. در تعریف یک متغیر نوع داده‌ی آن مشخص می‌شود و می‌توان چندین متغیر را در یک خط تعریف کرد. به عنوان مثال:

```
unsigned int    i, j=0, k;
```

نام یک متغیر می‌تواند شامل حروف بزرگ و کوچک، اعداد و _ باشد. نخستین کاراکتر نام یک متغیر، نمی‌تواند عدد باشد. زبان برنامه نویسی C به کوچک و بزرگ بودن حروف حساس است.

۱-۷) اعلان متغیر (Variable Declaration)

اعلان یک متغیر به کامپایلر این اطمینان را می‌دهد که متغیری با نوع و نام مشخص وجود دارد تا کامپایلر بتواند بدون نیاز به جزئیات کامل در مورد متغیر ، برای کامپایل بیشتر اقدام کند. اعلان یک متغیر زمانی مفید است که برنامه نویس از چندین فایل استفاده می‌کند که در یکی از آن‌ها تعریف متغیر آمده و در دیگر فایل‌ها آن متغیر به کار برده شده است. از کلید واژه‌ی extern

هنگام اعلان یک متغیر در یک فایل زمانی که در فایل دیگری تعریف شده است، استفاده می‌شود. یک متغیر تنها یک بار تعریف می‌شود اما می‌تواند بارها در توابع و بلوک‌ها و فایل‌های مختلف اعلان شود. تفاوت تعریف، اعلان و مقدار دهی یک متغیر در مثال زیر آورده شده‌است:

```
// Variable declaration:
extern int a, b;
extern float f;
/* variable definition: */
   int a, b;
   float f;
/* actual initialization */
   a = 10;
```

۱-۸) شناساگر قالب (Format Specifier)

شناساگرهای قالب در زبان برنامه‌نویسی C برای اهداف ورودی و خروجی استفاده می‌شوند. با استفاده از این مفهوم ، کامپایلر می‌تواند نوع داده‌ای که قرار است توسط تابع scanf خوانده شود و یا توسط تابع printf نوشته شود را درک کند. در اینجا فهرستی از شناسگرهای قالب مطابق جدول ۱-۲ آورده شده‌است.

Format Specifier	Type
%c	**Character**
%hi	**Signed integer (short)**
%hu	**Unsigned Integer (short)**
%d	**Signed integer**
%u	**Unsigned int**
%l or %ld or %li	**Long**
%lu	**Unsigned int or unsigned long**
%lld	**Long long**
%llu	**Unsigned long long**
%f	**Float values**
%lf	**Double**
%e	**Scientific notation of floats**
%le	**Scientific notation of doubles**
%p	**Pointer**
%s	**String**
%x or %X	**Hexadecimal representation**
%o	**Octal representation**
%%	**Prints % character**

جدول ۱- ۲ شناسگرهای قالب در زبان C

* از . (نقطه) برای معین کردن عددهای بعد از ممیز استفاده می‌شود. مثال:

```
double number=25.63598721;
printf("%4.3lf",number); //at least 4 wide and a precision of 3

output: 25.636
```

در نهایت برای جمع‌بندی هر آنچه که تا کنون شرح داده شد جدول ۱-۳ آمده‌است.

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
char	1	-128 to 127	%c
float	4		%f
double	8		%lf

جدول ۱- ۳ انواع Data Type و Format specifier متناظر آن در زبان برنامه نویسی C

۱-۹) خواندن ورودی از کیبورد توسط تابع scanf

توسط تابع scanf می‌توان رشته‌ای که توسط کاربر از طریق کیبورد وارد می‌شود و شامل کاراکترهای اسکی باشد، خواند.

```
int testInteger;
    printf("Enter an integer: ");
    scanf("%d", &testInteger);
```

۱-۱۰) حوزه‌ی متغیر (Variable Scope)

حوزه در هر زبان برنامه‌نویسی قسمتی از برنامه است که یک متغیر تعریف شده بتواند وجود داشته باشد و فراتر از آن ناحیه متغیر غیر قابل دسترسی باشد. دو ناحیه برای حوزه‌ی تعریف یک متغیر وجود دارد:

- داخل یک تابع یا بلوک که این متغیر، متغیر محلی نامیده می‌شود.
- خارج از همه‌ی توابع که این متغیر، متغیر سراسری نامیده می‌شود.

متغیرهای محلی

متغیرهایی که در داخل یک تابع یا بلوک تعریف شده‌اند، متغیرهای محلی نامیده می‌شوند. این متغیرها را فقط می‌توان با دستوراتی که در داخل آن تابع یا بلوک هستند، استفاده نمود. متغیرهای محلی برای تابع‌های خارجی شناخته شده نیستند. مثال زیر نحوه استفاده از متغیرهای محلی را نشان می‌دهد. در اینجا همه متغیرهای a، b و c متغیرهای محلی تابع main هستند.

```
#include <stdio.h>

int main () {
  /* local variable declaration */
  int a, b;
  int c;
   /* actual initialization */
  a = 10;
  b = 20;
  c = a + b;
   printf ("value of a = %d, b = %d and c = %d\n", a, b, c);
   return 0;
}
```

متغیرهای سراسری

متغیرهای سراسری که خارج از تابع، معمولاً در بالای برنامه تعریف می‌شوند. متغیرهای سراسری مقدار خود را در طول عمر برنامه حفظ می‌کنند و می‌توان در هر یک از توابع تعریف شده برای برنامه به آنها دسترسی داشت.

```
#include <stdio.h>

/* global variable declaration */
int g;

int main () {
  /* local variable declaration */
  int a, b;

  /* actual initialization */
  a = 10;
  b = 20;
  g = a + b;
   printf ("value of a = %d, b = %d and g = %d\n", a, b, g);

  return 0;
}
```

۱-۱۱) کلاس‌های ذخیره‌سازی (Storage Classes)

کلاسهای ذخیره‌سازی برای توصیف ویژگیهای یک متغیر/تابع استفاده می شوند. این ویژگیها اساساً شامل حوزه، دید و طول عمر است که به ما کمک می کند تا وجود یک متغیر خاص را در طول زمان اجرای برنامه ردیابی کنیم.

خودکار (auto)

این کلاس ذخیره سازی پیش فرض برای همه متغیرهای اعلام شده در یک تابع یا یک بلوک است.

بیرونی (extern)

کلاس ذخیره سازی بیرونی به سادگی به ما می‌گوید که متغیر مورد نظر در جای دیگری و نه در همان بلوک که در آن استفاده می‌شود، تعریف شده است. اساساً ، مقدار در یک بلوک به متغیر اختصاص داده می‌شود و همچنین می‌توان مقدار آن را در یک بلوک دیگر نیز باز نویسی کرد یا تغییر داد. بنابراین یک متغیر بیرونی چیزی نیست جز یک متغیر سراسری که تعریف و مقداردهی شده‌است تا در جایی دیگر اعلان شود و مورد استفاده قرار گیرد. این متغیر می‌تواند در هر تابع/بلوک قابل دسترسی باشد. در واقع ما یک متغیر جدید را تعریف نمی‌کنیم بلکه در عوض فقط از متغیر سراسری استفاده می کنیم. هدف اصلی استفاده از متغیرهای بیرونی این است که می توان بین دو فایل مختلف که بخشی از یک برنامه بزرگ هستند به یک متغیر دسترسی پیدا کرد.

ایستا (static)

این کلاس ذخیره سازی برای اعلان متغیرهای ایستایی استفاده می شود که در نوشتن برنامه ها به زبان C به طور گسترده استفاده می‌شوند. متغیرهای ایستا دارای ویژگی حفظ مقدار خود حتی پس از خارج از حوزه‌ی خود هستند. بنابراین ، متغیرهای ایستا آخرین مقدار خود را حفظ می‌کنند. می‌توان گفت که آنها فقط یک بار مقدار دهی می‌شوند و تا پایان برنامه وجود دارند بنابراین، هیچ حافظه‌ی جدیدی به آنها اختصاص نمی‌یابد.

ثبات (register)

این کلاس ذخیره سازی کاملا مشابه کلاس ذخیره‌سازی خودکار است با این تفاوت که متغیرهای که با کلاس ذخیره‌سازی ثبات تعریف می‌شوند، در صورت وجود فضای خالی در ثبات‌های ریزپردازنده، در این ناحیه ذخیره خواهند شد.

۱-۱۲) توابع (Functions)

تابع شامل مجموعه‌ای از دستورات است که با هم یک وظیفه را انجام می‌دهند. هر برنامه C حداقل یک تابع به نام ()main دارد. برنامه‌نویس می‌تواند کد خود را به چندین بخش تقسیم کند که هر بخش توسط یک تابع پوشش داده‌شود. نحوه تقسیم کد بین توابع مختلف به خود برنامه‌نویس بستگی دارد، اما منطقاً تقسیم بندی به گونه ای است که هر تابع وظیفه خاصی را انجام دهد. اعلان تابع اطلاعاتی شامل نام تابع، نوع داده‌ای که برمی‌گرداند و آرگومان‌های تابع را به کامپایلر می‌دهد. تعریف تابع بدنه‌ی اصلی آن را مشخص می‌کند. کتابخانه‌های استاندارد زبان C تعداد زیادی تابع از پیش‌تعریف‌شده مانند printf و scanf در اختیار کاربر قرار می‌دهند تا برای مقاصد خاص در برنامه‌ی خود به کار ببرد.

مثال برای تعریف تابع:

```
/* function returning the max between two numbers */
int max(int num1, int num2) {
   /* local variable declaration */
   int result;
    if (num1 > num2)
      result = num1;
   else
      result = num2;
    return result;
}
```

مثال برای اعلان تابع:

```
int max(int num1, int num2);
int max(int, int);
```

۱-۱۳)تبدیل نوع داده (Type Casting)

به فرآیند تبدیل یک نوع داده به نوع دیگر، تبدیل نوع داده گفته می شود. به عنوان مثال، اگر بخواهیم یک مقدار با نوع داده‌ی long را در یک متغیر نوع int ذخیره کنیم، می توانیم با تبدیل نوع داده، یک مقدار long را درون یک متغیر int ذخیره کنیم. تبدیل نوع می‌تواند به صورت ضمنی (implicit) باشد که کامپایلر به طور خودکار انجام می‌دهد و یا می‌تواند با استفاده از اپراتورهای تبدیل توسط برنامه‌نویس به صورت صریح (explicit) انجام شود.

```
int main() {

   int sum = 23, count = 7;
   double mean;

   mean = (double) sum / count;
   printf("Value of mean : %f\n", mean );
}
```

پرسش‌ها و تمرین‌های برنامه‌نویسی

۱- یک دستور خروجی بنویسید که به کمک کاراکترهای قابل چاپ و escape sequence در یک رشته مطالب زیر را چاپ کند:

HELLO را چاپ کند.

یک tab افقی جلو برود.

بعد there را چاپ کند.

دو خط بعد از آن را خالی بگذارد.

در ابتدای خط چهارم "\"my name\"" را چاپ کند.

به جای my name نام خود را بنویسید مثلن "\"Jalal Ahmadi\"".

۲- اعلان و بدنه‌ی تابعی را بنویسید که دو متغیر با نام‌های num1 و num2 را که می‌توانند مقادیری صحیح بین ۰ تا ۱۰۰۰۰ داشته‌باشند را به عنوان ورودی دریافت کرده و خروجی آن مجموع این دو عدد تقسیم بر ۳ باشد. دقت کنید خروجی باید به صورت اعشاری تعریف شود در حالی که نوع ورودی عدد صحیح است.

۳- خروجی برنامه‌ی زیر چیست؟

```
#include <stdio.h>
void test(void);
int main()
{
    test();
    test();
    test();
    return 0;
}
void test(void){
    int i = 0;
    static int j=0;
    i++;
    j++;
    printf("i = %d  j = %d\n",i,j);
}
```

۴- تعریف توابع ذیل درست است یا خیر؟

```
/*a*/
void test1(int m, int n){
    return 3*m+n;
}
/*b*/
float test2(int i,float x){
    i = i + 7;
    x = 4.8 + i;
}
```

اگر غلط است تعریف درست را بنویسید یا تابع را اصلاح کنید.

۲) برنامه‌نویسی به زبان C ویژه‌ی میکروکنترلرها – بخش دوم

۲-۱) اشاره‌گرها (Pointers)

اشاره‌گر ها متغیرهایی هستند که نشانی یا آدرس متغیر دیگری را در خود ذخیره می‌کنند. برای معرفی متغیر از نوع اشاره‌گر به کامپایلر از علامت * استفاده می‌شود.

```
int* pAddress;
```

بیان می‌کند که pAddress متغیری از نوع اشاره‌گر است (متغیر اشاره‌گر) که به داده‌ای از نوع int اشاره می‌کند.

زمانی که می‌خواهیم یک متغیر اشاره گر تعریف کنیم از نحو (syntax) زیر استفاده می‌کنیم:

```
int* pRCC_REG;
int *pRCC_REG;
```

هر دو صحیح می‌باشند اما ترجیح ما استفاده از syntax اول است چرا که خوانایی بهتری دارد. اما در موردی که می‌خواهیم چندین اشاره‌گر را در یک خط تعریف کنیم باید حتما از syntax دوم استفاده کنیم یعنی:

```
int *pRCC_REG,*pGPIOA_REG,*pGPIOB_REG;
```

استفاده از کاراکتر p در شروع نام یک اشاره‌گر اختیاری است و تنها موجب خوانایی بهتر کد می‌شود. می‌دانیم آدرس پایه‌ی حافظه‌ی فلش در میکروکنترلر مورد نظر ما برابر است با 0x0800 0000. حال برای این که به این آدرس دسترسی داشته باشیم از syntax زیر استفاده می‌کنیم:

```
<data type>* pFlash_base = (<data type>*)0x80000000;
```

دقت کنید که برای تبدیل یک عدد به متغیر اشاره‌گر حتما نیاز است تا type cast انجام دهیم. بسته به نوع داده‌ای که تعریف می‌کنیم نوع داده‌ای که اشاره‌گر به آن اشاره می‌کند، متفاوت است. حال فرض کنید می‌خواهیم آدرس یک متغیر را درون اشاره‌گر ذخیره کنیم.

```
int myVariable = 25;
int* pMyVariable = &myVariable;
```

کاراکتر & می‌تواند به عنوان یک عملگر آدرس متغیر را برگرداند.

نوشتن اطلاعات از اشاره‌گر

برای نوشتن محتوای یک اشاره‌گر از کاراکتر * پیش از نام متغیر اشاره‌گر استفاده می‌کنیم و از آن به بعد مثل یک متغیر ساده با آن برخورد می‌کنیم.

```
int myVariable = 25;
unsigned char* pFlash_base = (unsigned char*)0x08000000;
*pFlash_base = 0xf4;
```

مطابق مثال بالا یک متغیر اشاره‌گر ایجاد کردیم و مقدار 0xf4 که یک بایت است را درون آن ذخیره کردیم.

خواندن اطلاعات از اشاره‌گر

برای خواندن اطلاعات یک اشاره گر نیز با گذاشتن کاراکتر * قبل از نام آن، مانند یک متغیر ساده اطلاعات را از آن می‌خوانیم.

```
unsigned char data;
data = *pFlash_base;
```

مقدار موجود در آدرس 0x0800 0000 که در مثال قبل برابر با 0xf4 قرار گرفت درون data ذخیره می‌شود.

۲-۲) فایل سرآیند stdint.h

در جلسه‌ی قبل گفتیم که برای داده‌های نوع int و long، بسته به کامپایلر، فضایی که برای ذخیره‌سازی در نظر گرفته می‌شود می‌تواند متفاوت باشد. به عنوان مثال کد زیر می‌تواند در یک کامپایلر به درستی اجرا شود و در کامپایلر دیگری عملکرد درستی نداشته باشد.

```
unsigned int counter = 0;
 while(1){
     counter++;
     if(counter>65536){
         //do task
     }
 }
```

در صورتی که نوع داده‌ی int ۲ بایت طول داشته باشد، دستور داخل if هیچ‌گاه اجرا نمی‌شود. اما اگر بیش از ۲ بایت طول داده‌ی نوع int باشد کد به درستی اجرا می‌شود.

برای حل این مشکل از کتابخانه stdint.h بهره برده می‌شود و به هر نوع داده اسمی مستعار تخصیص می‌یابد و طول هر نوع داده مستقل از نوع کامپایلر مقدار ثابتی است.

```
typedef signed char           int8_t;
typedef unsigned char         uint8_t;
typedef short                 int16_t;
typedef unsigned short        uint16_t;
typedef int                   int32_t;
typedef unsigned              uint32_t;
typedef long long             int64_t;
typedef unsigned long long    uint64_t;
```

۲-۳) عملگرها (Operators)

عملگر نمادی است که به کامپایلر می‌گوید روی یک یا چند عملوند عملیات ریاضی یا منطقی انجام دهد.

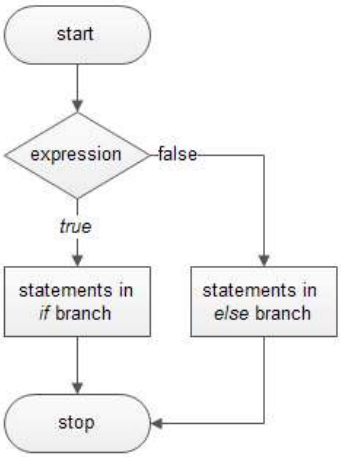
- عملگرهای ریاضی (+ - / * ++ -- %)
- عملگرهای ارتباطی (== >= <= != > <)
- عملگرهای منطقی (&& || !)
- عملگرهای بیتی (& | ^ ~)
- عملگرهای انتسابی (+= -= &= …)
- عملگرهای متفرقه (?)

۲-۴) جملات شرطی در زبان C (if-switch)

دستور IF-ELSE

با استفاده از دستور if می‌توانیم به کامپایلر بگوییم اگر یک شرط معین برقرار بود دنباله‌ای از دستورات را انجام بده و اگر برقرار نبود عملی دیگر را انجام بده.

```
if(boolean_expression) {
   /* statement(s) will execute if the boolean expression is true */
} else {
   /* statement(s) will execute if the boolean expression is false */
}
```



شکل ۲- ۱ فلوچارت if-else

دستور SWITCH

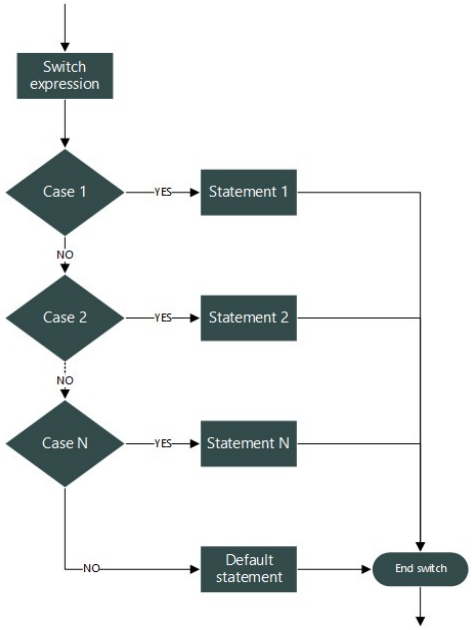
یک ساختار کنترلی انتخابی است که با آن می‌توان تعداد زیادی انشعاب در برنامه ایجاد کرد. مقدار عبارت شرطی switch با موردهای مختلف مقایسه می‌شود و در صورت تطبیق داشتن زیر روال برنامه ایجاد می‌گردد.

```
switch(expression) {

   case constant-expression  :
      statement(s);
      break; /* optional */

   case constant-expression  :
      statement(s);
      break; /* optional */

   /* you can have any number of case statements */
   default : /* Optional */
   statement(s);
}
```



شکل ۲- ۲ فلوچارت switch

عبارت switch حتما باید عدد صحیح یا کاراکتر باشد.

عملگر ؟

این عملگر مشابه ساختار IF-ELSE است.

```
variable = Expression1 ? Expression2 : Expression3
```

اگر عبارت ۱ صحیح باشد مقدار متغیر برابر عبارت ۲ در غیر این صورت مقدار متغیر برابر با عبارت ۳ است.

۲-۵) حلقه‌ها در زبان C (while, do while, for)

دستور WHILE

دستور WHILE مشابه دستور IF، شرطی را بررسی می‌کند اگر درست بود یک سری دستورات معین را اجرا می‌کند و دوباره شرط را بررسی می‌کند و اگر درست بود باز همان دستورات را اجرا می‌کند تا زمانی که عبارت شرطی مقدار صفر یا FALSE بگیرد یا یک دستور BREAK داخل حلقه اجرا شود.

```
while(condition) {
   statement(s);
}
```

دستور DO WHILE

دستور DO WHILE مانند دستور WHILE است با این تفاوت که دستورات داخل حلقه حتما حداقل یک بار اجرا می‌شوند.

```
do {
   statement(s);
} while( condition );
```

دستور FOR

حلقه FOR یک ساختار کنترل تکراری است که به ما این امکان را می‌دهد حلقه ای را چندین بار اجرا کنیم.

```
for ( init; condition; increment ) {
   statement(s);
}
```

در حلقه‌ی FOR ابتدا و تنها برای یک بار INIT اجرا می‌شود سپس شرط حلقه یا CONDITION بررسی می‌شود، در صورتی که صحیح بود دستورات داخل حلقه اجرا می‌شود و پس از آن INCREMENT یا DECREMENT صورت می‌گیرد. در نهایت این کار بارها و بارها تکرار می‌شود تا شرط داخل حلقه مقدار 0 به خود بگیرد.

۲-۶) عملگرهای بیتی (Bitwise operation)

از عملگرهای بیتی برای تغییر بیت‌های رجیسترهای میکروکنترلرها استفاده می‌کنیم. فرض کنید می‌خواهیم بیت‌های صفرم و دوم از رجیستر RCC_REG را یک کنیم اما نمی‌خواهیم دیتای سایر بیت‌های این رجیستر را تغییر دهیم. به همین دلیل **نمی‌توانیم** از روش زیر استفاده کنیم:

```
RCC_REG = 0b0000_00101;
```

چرا که ما با استفاده از مقداردهی مستقیم، بعضی از بیت ها را با صفر مقدار دهی کردیم که شاید از ابتدا مقدار دیگری داشتند و این کار می‌تواند موجب شود که برنامه به درستی کار نکند. به همین دلیل باید از عملگرهای بیتی استفاده کنیم.

عملگر شیفت به راست و چپ

در اغلب موارد برای انجام عملیات مختلف روی بیت‌ها احتیاج به دنباله‌ای از صفرها و یک‌ها داریم. برای ایجاد هر 1 در دنباله، عدد یک را به اندازه مکان آن بیت شیفت می‌دهیم و در نهایت تمام این یک های شیفت داده شده را با هم OR می‌کنیم.

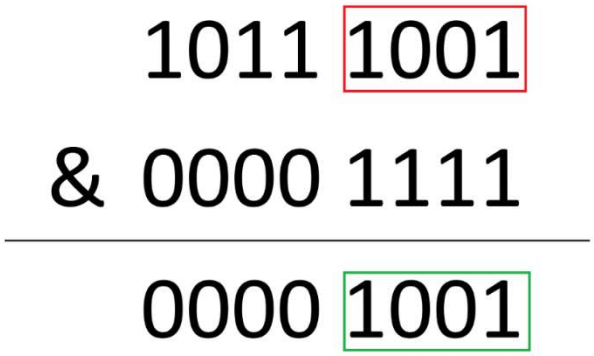
```
  1000 0000   (1<<7)
| 0010 0000   (1<<5)
| 0001 0000   (1<<4)
| 0000 1000   (1<<3)
| 0000 0001   (1<<0)
-------------------
  1011 1001
```

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

int main()
{
    uint8_t sample;
    sample = ((1<<7)|(1<<5)|(1<<4)|(1<<3)|(1<<0));
    printf("%x",sample);
    return 0;
}
```

READ

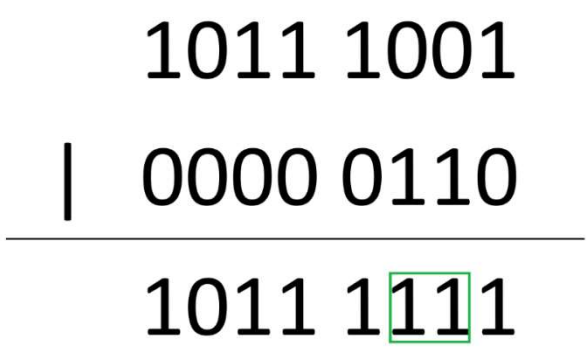
با استفاده از عملگر & می‌توان بیت‌های مورد نظر را از یک رجیستر یا متغیر خواند.




```
uint32_t*  RCC_AHB1ENR =(uint32_t*)0x40023830; //points to RCC_AHB1ENR
register
read_data = *RCC_AHB1ENR &((1<<3)|(1<<2)|(1<<1)|(1<<0));//read data from
register
```

SET

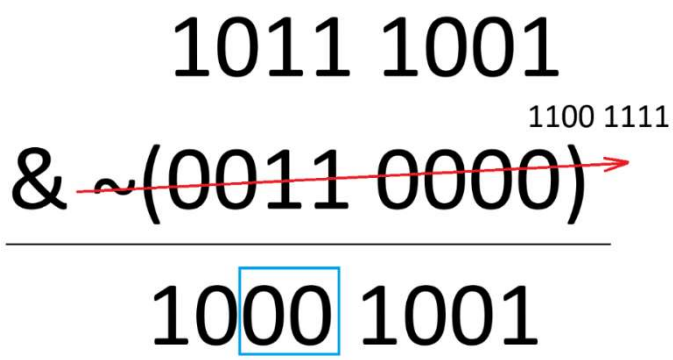
با استفاده از عملگر | می‌توان بیت‌های مورد نظر را از یک رجیستر یا متغیر SET نمود.




```
*RCC_AHB1ENR |=((1<<2)|(1<<1));
```

CLEAR

با استفاده از عملگرهای & و ~ می‌توان بیت‌های مورد نظر را از یک رجیستر یا متغیر CLEAR نمود.




```
*RCC_AHB1ENR &=~((1<<5)|(1<<4));
```

سه عمل پرکاربرد در برنامه نویسی میکروکنترلرها شرح داده شد. هم‌چنین برای معکوس کردن (TOGGLE) بیت ها نیز مشابه روش بالا می‌توان از عملگر ^ استفاده نمود.

۲-۷) توصیف‌کننده‌های نوع (Type Qualifiers)

در زبان برنامه نویسی C، توصیف‌کننده‌های نوع کلید واژه هایی هستند که برای اصلاح ویژگی های متغیرها استفاده می‌شوند. با استفاده از توصیف کننده‌های نوع، می‌توانیم خواص متغیرها را تغییر دهیم. در زبان برنامه نویسی C، type qualifier عبارتند از:

- Const
- volatile

Const

به بیان ساده const ، یک متغیر را به read-only تبدیل می‌کند یعنی متغیری که به صورت const تعریف شده باشد، در طول برنامه نمی‌تواند توسط برنامه نویس تغییر کند و تنها قابل خواند است.

```
uint32_t const a;
const uint32_t a;
```

هر دو تعریف از نظر syntax صحیح هستند. دقت داشته باشید که در متغیرهای محلی که به صورت const تعریف می‌شوند، به صورت مستقیم در برنامه قابل تغییر نیستند اما می‌توان اشاره‌گری به آن متغیر ایجاد کرد و از طریق آن اشاره گر محتوی متغیر را تغییر داد چرا که متغیرهای محلی در SRAM ذخیره می‌شوند. اما در متغیرهای سراسری که از نوع const تعریف شوند حتی با ایجاد اشاره‌گر به آنها، نمی‌توان تغییرشان داد چرا که این متغیرها در حافظه‌ی Flash میکروکنترلر ذخیره می‌شوند.

☑ کاربرد اصلی const در برنامه نویسی میکروکنترلرها در آدرس دهی به رجیسترهاست.

```
uint32_t* const pRCC_AHB1ENR =(uint32_t*)0x40023830;
```

در مثال بالا ما به کامپایلر می‌گوییم که آدرس رجیستر RCC_AHB1ENR برابر 0x4002_3830 است و در طی فرآیند کد نویسی نباید تغییر کند.

اشاره‌گر زیر را در نظر بگیرید:

```
uint32_t const* pData;
```

این اشاره‌گر به داده‌ای اشاره می‌کند که تنها قابل خواند است و قابل نوشتن نیست اما خود اشاره‌گر می‌تواند تغییر کند یعنی عبارت زیر مجاز نیست:

```
*pData = 23; ☒
```

اما عبارت زیر مجاز است:

```
pData = (uint32_t*)0x40002000;☑
```

به همین ترتیب دو مدل دیگر از تعریف به همراه const وجود دارد:

```
uint32_t* const pData;
```

یعنی اشاره‌گر غیرقابل تغییر به داده‌ی قابل تغییر.

```
uint32_t const* const pData;
```

اشاره‌گر غیر قابل‌تغییر به داده‌ی غیرقابل تغییر(فقط خواندنی).

Volatile

کلید واژه‌ی volatile برای اطلاع کامپایلر از این موضوع است که مقدار متغیر در هر لحظه می‌تواند بدون اینکه پردازشی توسط کد روی آن صورت گیرد، تغییر کند. به عنوان مثال فرض کنید که کلیدی به کی از ورودی‌ها میکرو متصل نمودیم. بدون اینکه عملی در کد صورت گیرد مقدار رجیستر مربوطه می‌تواند تغییر کند.

```
uint32_t volatile a;
volatile uint32_t a;
```

از کلید واژه volatile برای آدرس دهی رجیسترهای ادوات جانبی میکروکنترلر و همچنین در تعریف متغیرهای سراسری که توسط توابع وقفه داخلی میکرو به کار گرفته می‌شوند استفاده می‌کنیم.

☑ روش صحیح آدرس‌دهی رجیسترهای ادوات جانبی میکروکنترلر مطابق مثال زیر است:

```
uint32_t volatile* const pRCC_AHB1ENR = (uint32_t*)0x40002000;
```

می‌دانیم رجیسترهای میکروکنترلر ARM ۳۲ بیتی هستند پس نوع دیتایی که اشاره‌گر ذخیر می‌کند را برابر uint32_t قرار می‌دهیم. از طرفی می‌دانیم که آدرس این رجیسترها ثابت است پس نوع اشاره‌گر را برابر const قرار می‌دهیم. در نهایت به آن دلیل که ممکن است مقدار این رجیسترهای توسط کاربر از بیرون تغییر کند از کلید واژه‌ی volatile نیز استفاده می‌کنیم.

مثال: با مراجعه به reference manual میکروکنترلر stm32f103xx آدرس رجیسترهای RCC_APB2ENR، GPIOB_CRL، GPIOB_CRH، GPIOB_IDR و GPIOB_ODR را پیدا کنید و با ایجاد اشاره‌گرهایی به این آدرس‌ها و استفاده از عملگرهای بیتی برای میکروکنترلر stm32f103c6 برنامه‌ای بنویسید که به کمک یک کلید بتوان یک led را خاموش و روشن کرد.

```c
#include <stdint.h>

int main(void)
{
	uint32_t           volatile* const pRCC_APB2EN  = (uint32_t *)0x40021018;
	//APB2 peripheral clock enable register
	uint32_t           volatile* const pGPIOB_CRL   = (uint32_t *)0x40010c00;
	//Port B configuration register low
	uint32_t           volatile* const pGPIOB_CRH   = (uint32_t *)0x40010c04;
	//Port B configuration register high
	uint32_t const     volatile* const pGPIOB_IDR   = (uint32_t *)0x40010c08;
	//Port B input data register
	uint32_t           volatile* const pGPIOB_ODR   = (uint32_t *)0x40010c0c;
	//Port B output data register

	*pRCC_APB2ENR      |=  (1<<3);// port B enabled

	*pGPIOB_CRL        &= ~((1<<3)|(1<<2)|(1<<0));
	*pGPIOB_CRL        |=(1<<1);//pin B.0 output 2MHz speed push-pull
	*pGPIOB_ODR        &=~(1<<0);// output low

	*pGPIOB_CRL        &= ~((1<<6)|(1<<5)|(1<<4));
	*pGPIOB_CRL        |=(1<<7);//pin B.1 input
	*pGPIOB_ODR        |= (1<<1);//pin B.1 pull up
	for(;;){
		if(!(*pGPIOB_IDR & (1<<1))){
			for(uint32_t i=0;i<100000;i++);
			while(!(*pGPIOB_IDR & (1<<1)));
			*pGPIOB_ODR ^=(1<<0);
		}
	}
}
```