整型

概念： 表示整数类型的数据

语法：

```
int a = 123 ;
```

1. 先向系统申请一片内存， 名字为 a

2. 确定该内存的大小为4字节 （1字节byte = 8 位bit)

3. 把 123 转换成二进制并存放到该内存中

注意：

- 整型在 32位系统以及64位系统中都占用 4字节
- 取值范围

```
$ getconf INT_MAX
2147483647
$ getconf INT_MIN
-2147483648
```

整型的修饰符：

- short 短整型，用于修饰整型的尺寸变为原本的一半，减少内存的开支，缩小取值范围
- long 长整型， 用于修饰整型的尺寸使其尺寸变大（取决与系统）， 增加内存开支， 扩大取值范围
- long long 长长整型， 用于修饰整型的尺寸使其尺寸变大（取决与系统）， 增加内存开支， 扩大取值范围 （在64位系统中 long 与 long long的大小一致）
- unsigned 无符号整型 ， 用来修饰整型并去掉符号位，使得整型数据没有负数，可以提正整数的取值范围 （0 - 4294967295）
    - 整型数据在二进制存储时最高位（第31位）表示符号位，如果为1 则表示负数反之则表示正数

•

整数的存储方式：

原码： 正整数是直接使用原码进行存储的，比如100这个正整数，则直接把100 转换成二进制直接存储。

100 --> 0000 0000 0000 0000 0000 0000 0110 0100

补码：负数则是使用补码来存储， 补码 = 原码的绝对值 取反+ 1

取反加1时符号位不变

比如 -100 ：

100 --> 1000 0000 0000 0000 0000 0000 0110 0100

取反-> 1111 1111 1111 1111 1111 1111 1001 1011

加1 --> 1111 1111 1111 1111 1111 1111 1001 1100

溢出：

当超过取值范围时则会编程相邻的最小值/最大值

整型输出：

```
int c = 100;
printf("十六进制：%#x\n"  , c );
printf("十进制：%d\n"  , c );
printf("八进制：%#o\n"  , c );

// 输出结果
十六进制：0x64
十进制：100
八进制：0144

```

sizeof运算符，用于计算 变量/类型 的大小。

```
int a = 123 ;
sizeof(int);
sizeof(a);
//注意括号内部可以写变量类型， 也可以写变量名
```

浮点型（实型）：

概念：用来表达一个实数的数据类型

分类：

- 单精度浮点型 float , 典型尺寸 4 字节
- 双精度浮点型 double , 典型尺寸 8 字节
- 长双精度 long double， 典型尺寸 16 字节

占用的内存越多则精度越高

浮点数的存储：

IEEE 浮点标准采用如下形式来表示一个浮点数

$$V = (-1)^s \times M \times 2^E$$

以 32 位浮点数为例，其存储器的内部情况是这样的：

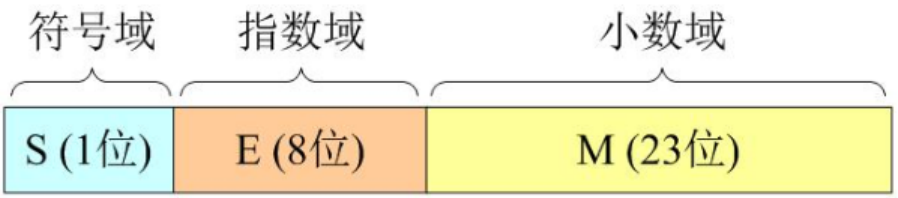


图 2-9 float 型浮点数内部存储结构

注意：

```
float f = 3.14; // 浮点数3.14 通过以上公式计算得到的二进制码被存放与内存f中
printf("f:%f\n" , f ); // 使用浮点的计算方法来解析内存f中的值(二进制)
printf("d:%d\n" , f );// 直接使用整型的计算方法来直接解析内存f 中的值 (二进制)
```

所有的数据都会被转换成二进制进行存储，如果想到的到正确的数据，必须使用正确的理解方式（类型），来解析二进制数据。

字符类型：

```
char c = 'K' ;
```

1. 申请一片内存并且命名为 c
2. 确定内存的大小为 char （1字节）
3. 把字符 ‘K’ 的ASCII码值转换为二进制，并存储到该内存中

4.

计算机中存储的所有数据都是以二进制的形式存在的， 因此字符必须映射某一个数字才能够被存放到计算机中，这个映射的表就成为 ASCII 表，可以使用man手册来查看：

```
man ascii
```

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0' (null character)	100	64	40	@
001	1	01	SOH (start of heading)	101	65	41	A
002	2	02	STX (start of text)	102	66	42	B
003	3	03	ETX (end of text)	103	67	43	C
004	4	04	EOT (end of transmission)	104	68	44	D
005	5	05	ENQ (enquiry)	105	69	45	E
006	6	06	ACK (acknowledge)	106	70	46	F
007	7	07	BEL '\a' (bell)	107	71	47	G
010	8	08	BS '\b' (backspace)	110	72	48	H
011	9	09	HT '\t' (horizontal tab)	111	73	49	I
012	10	0A	LF '\n' (new line)	112	74	4A	J
013	11	0B	VT '\v' (vertical tab)	113	75	4B	K
014	12	0C	FF '\f' (form feed)	114	76	4C	L
015	13	0D	CR '\r' (carriage ret)	115	77	4D	M
016	14	0E	SO (shift out)	116	78	4E	N
017	15	0F	SI (shift in)	117	79	4F	O
020	16	10	DLE (data link escape)	120	80	50	P
021	17	11	DC1 (device control 1)	121	81	51	Q

```
 char c = '1' ;

printf("字符：%c\n",c); // 以字符的形式来解析内存 c 的内容得到 对应的字符 1
printf("整型ASCII值：%d\n",c); //以十进制整型来解析内存c 的内容 ， 得到1多对应的ASCII值
```

注意：

字符实质上是一个单字节的整型，因此支持所有整型的操作

```
char k = 'H' ;
printf("%c\n" , k + 1 );
printf("%c\n" , k - 1 );
```

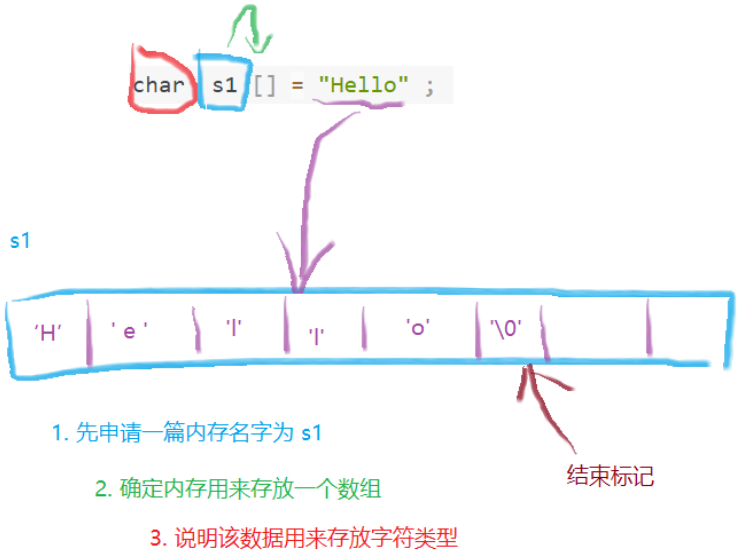
字符串：

字符串的表现形式有两种：

形式一 数组（可读，可写）：（存储）

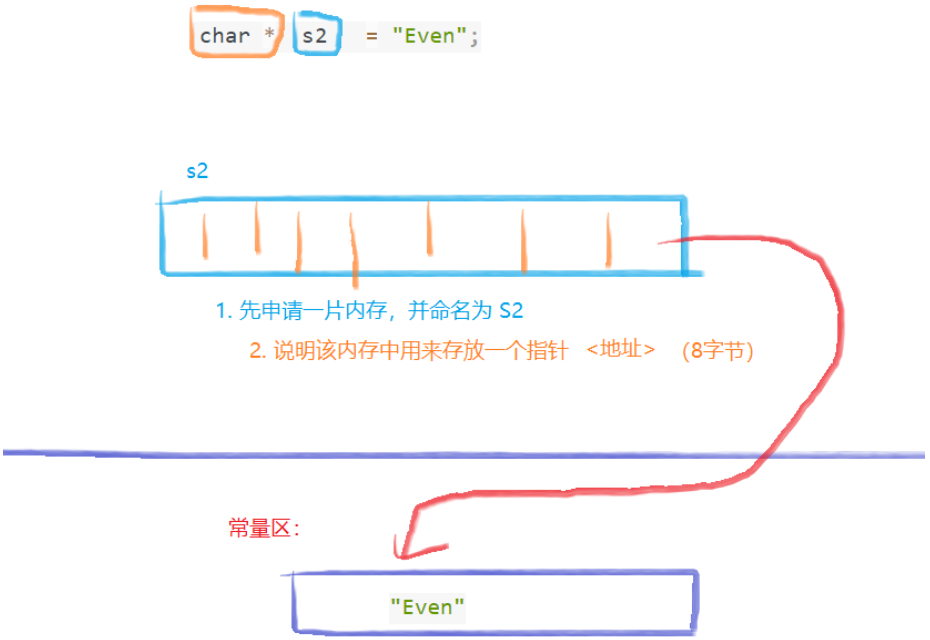
```
char s1 [] = "Hello" ; //使用一个数组来存放字符串 "Hello"
```

以上语句其实是把一个字符串常量 "Hello"， 复制到数组 s1 所代表的内存当中



形式二 指针 （只读）：（指向）

```
char * s2 = "Even"; // 使用一个指针来指向常量字符串
```




```
char s1[] = "Hello";
char *s2  = "Even";
```

```
3
4 printf("%s %s\n" , s1 , s2  )
```

布尔类型：

概念： 布尔类型用来表示真/假 （非零则真）

真： true

假： false

注意在使用布尔类型是需要包含他的头文件： <stdbool.h>

```
1 bool a = 1 ; // 真
2 bool a = 0 ; // 假
3 bool a = true // 真
4 bool a = false // 假
```

一般布尔类型的变量，可以用于逻辑判断比如 if / while ， 或者用于函数的返回值。

布尔类型的大小：

```
1 bool a = true ;
2
3 printf("sizeof(bool):%ld \n ",sizeof(bool) );
4 printf("sizeof(a):%ld \n ",sizeof(a) );
5 printf("sizeof(true):%ld \n ",sizeof(true) );
6 printf("sizeof(false):%ld \n ",sizeof(false) );
7 printf("false:%d \n ",false );
8 printf("true:%d \n ",true );
9
10 //运行结果：
11 sizeof(bool):1
12 sizeof(a):1
13 sizeof(true):4
14 sizeof(false):4
15 false:0
16 true:1
```

常量与变量：

概念： 不可以被改变的内存，被称为常量，可以被改变的内存则成为变量

```
1 int a = 100 ; // a是一个变量， 而 100 则是常量
```

```
float f = 3.1415; // f 是一个变量， 而 3.1415 则是常量
char s1[] = "abcdefg" ; // s1 是一个变量 , 而"abcdefg" 则是常量 （字符串常量）
```

常量类型：

- 100 ： 整型常量
- 100L： 长整型 long
- 100LL : 长长整型 long long
- 100UL : 无符号的长整型 unsigned long
- 3.14 ：编译器默认升级为双精度浮点型
- 3.14L : 长的双精度浮点型
- 'a' : 字符常量
- "Hello" : 字符串常量（指针 char * ）

标准输入：

概念： 标准输入一般指的是键盘的设备文件， 从键盘获取数据就成为标准输入

```
scanf( ); // 扫描键盘 （格式化输入数据 --> 从键盘中获得指定类型的数据）
头文件：
 #include <stdio.h>
函数原型：
 int scanf(const char *format, ...);
参数分析：
 format --> 格式化
 ... --> 省略号 , 根据format 所写的格式控制符，对应一个内存地址
返回值：
  成功 返回具体获取到的项目数
  失败 返回0



getchar( ) ; // 获取一个字符 （默认从标准输入文件中获取）

  函数原型：
  int getchar(void);
  参数：
  无
  返回值：
  成功 返回一个ASCII值， 代表获得的字符 （unsigned char ）
  失败 返回 EOF 也就是 -1
```

实例：

```c
int main(int argc, char const *argv[])
{
    int num = 0;
    char c = '0';
    // & 取地址符号 ， 获得 内存 num 的地址
    int ret_val =  scanf("%d", &num );
    while(getchar() != '\n');  // 清空 由 scanf 所留在缓冲区的内容
    //   如果不清空有可能会导致下一次使用标准输入缓冲区异常（有上一次的数据没有被读取）

     printf("返回值: %d , 获取的数据为: %d\n" , ret_val , num );

     ret_val = scanf("%c", &c );
     while(getchar() != '\n'); // 清空 由 scanf 所留在缓冲区的内容
     printf("返回值: %d , 获取的数据为: %c\n" , ret_val , c );

     printf("EOF:%d\n" , EOF ); //EOF 实际上是一个值为-1 的宏

     return 0;
}

```

注意：

如果以后需要从标准输入中获取数据得到乱码或未知数据，则可以尝试使用getchar进行清空再 获取。

作业：

1. 设计一个程序实现用户输入字符，则程序输出其对应的ASCII码值。
2. 设计一个程序实现用户输入一个ASCII值， 则程序输出其对应的字母（字符）。
3. 编写一个程序输出26个小写字母或者大写字母【拓展】

4. 编写一个程序，用户输入华氏温度F，程序输出摄氏温度C，结果保留2位小数。

$$c = \frac{5}{9}(f - 32)$$

转换公式： C = 5×（F - 32）÷ 9

预习：

运算符

控制流