> Always continue; Never break ;

> 静心 心态 清晰的思路

@[toc]

Day05 常用的输入输出原理

scanf() 的使用

```
1 #include <stdio.h>
2 int scanf( const char *format, ... );
```

scanf

语法:

```
#include <stdio.h>
int scanf( const char *format, ... );
```

scanf() 函数根据由*format*(格式)指定的格式从**stdin**(标准输入)读取, 并保存数据到其它参数. 它和printf()有点类似. *format*(格式)字符串由控制字符, 空白字符和非空白字符组成. 控制字符以一个%符号开始, 如下:

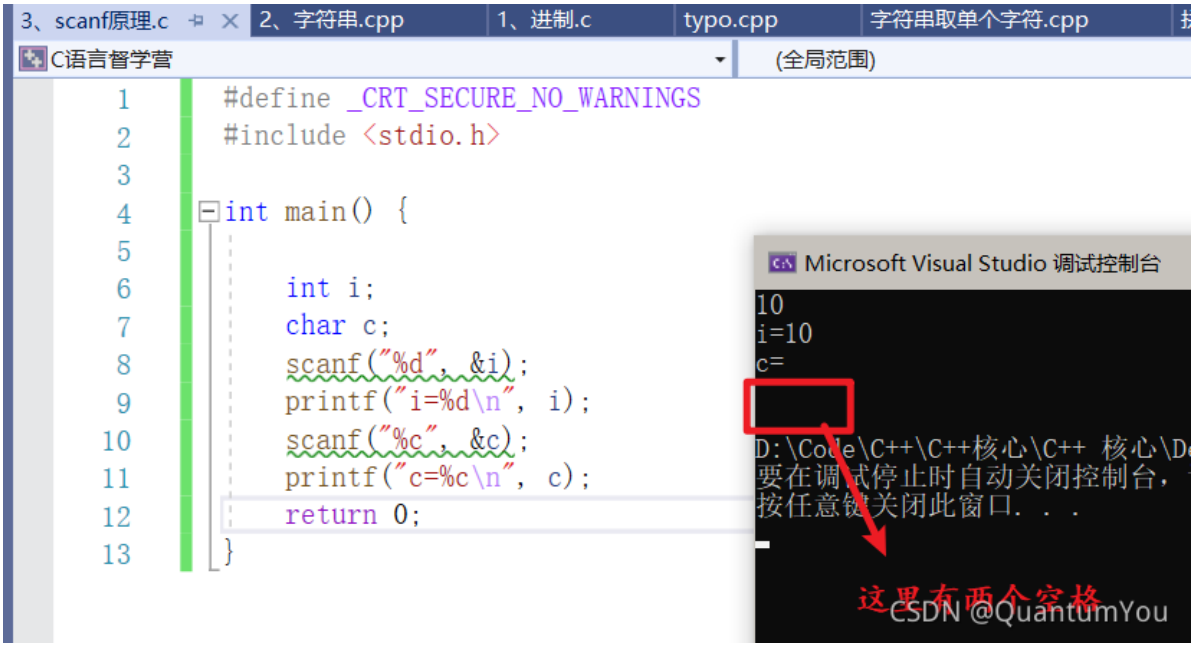
控制字符	说明
%c	一个单一的字符
%d	一个十进制整数
%i	一个整数
%e, %f, %g	一个浮点数
%o	一个八进制数
%s	一个字符串
%x	一个十六进制数
%p	一个指针
%n	一个等于读取字符数量的整数
%u	一个无符号整数
%[]	一个字符集
%%	一个精度符号

scanf()读取匹配*format*(格式)字符串的输入. 当读取到一个控制字符, 它把值放置到下一个变量. 空白(tabs, 空格等等)会跳过. 非空白字符和输入匹配, 然后丢弃. 如果是一个在%符号和控制符间的数量, 那么只有指定数量的字符转换到变量中. 如果scanf()遇到一个字符集(用%[]控制字符表示), 那么在括号中的任意字符都会读取到变量中. scanf()的返回值是成功赋值的变量数量, 发生错误时返回EOF.

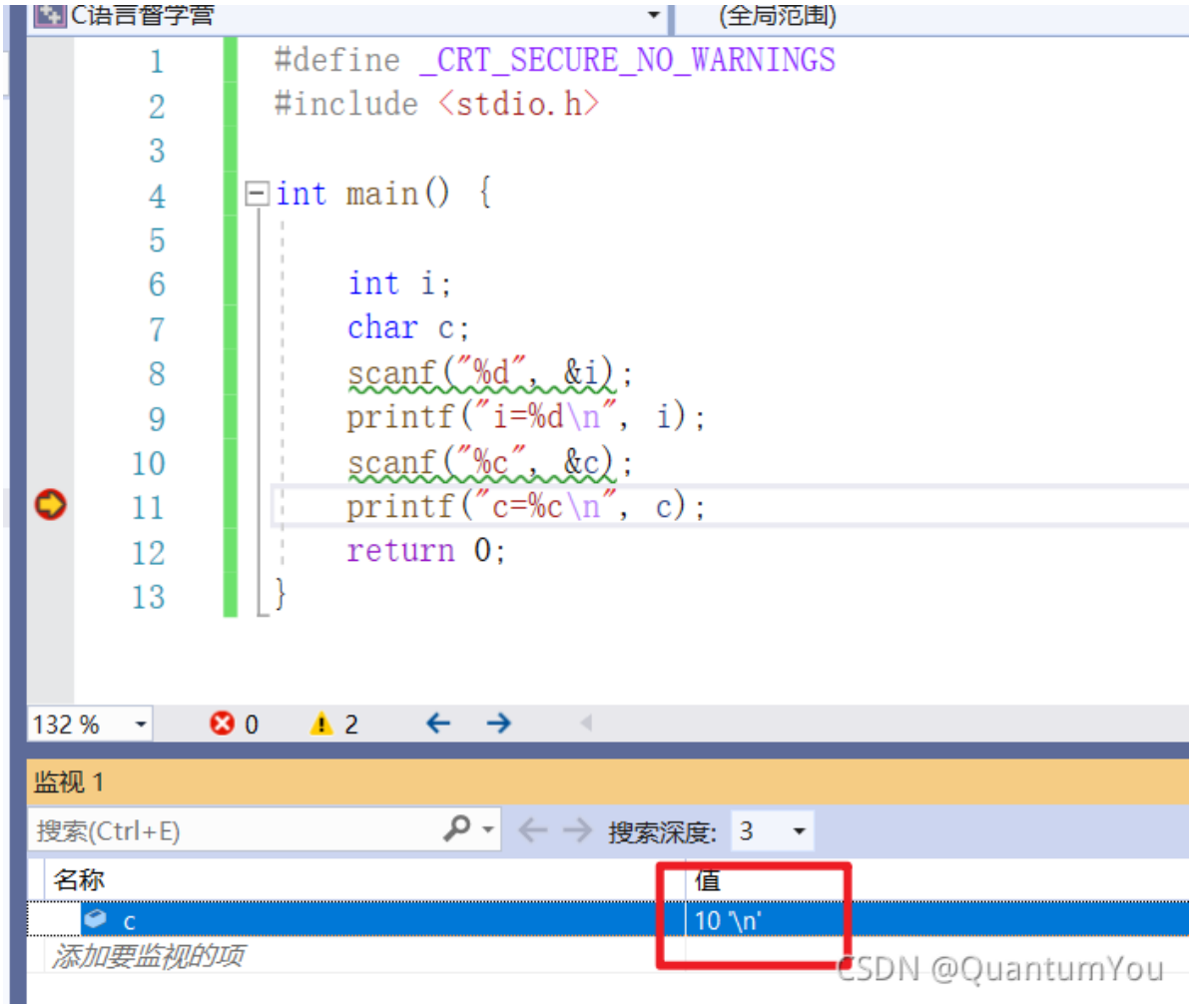
- `format` 是一个字符串，...是可变参数，参数的数目与`format`中的`%`的数目保持一致
- `%d` 一个十进制整数
- `%f` 一个浮点数
- `%c` 一个单一的字符

标准输入缓冲区的原理

- C语言未提供输入/输出关键字，其输入和输出是通过标准函数库来实现的。C语言通过`scanf`函数读取键盘输入，键盘输入又被称为标准输入。当 scanf函数读取标准输入时，如果还没有输入任何内容，那么 scanf函数会被卡住（专业用语为阻塞）.下面来看一个例子



执行时输入10,然后回车，显示结果如图所示。为什么第二个 scanf函数不会被阻塞呢？其实是因为第二个 scanf函数读取了缓冲区中的“`\n`”,即 scanf("%c",&c)实现了读取，打印其实输出了换行，所以不会阻塞



如上因为 `\n` 的ASCII 为10，确实被读取到了

缓冲区的原理

- **行缓冲**：在这种情况下，当在输入和输出中遇到换行符时，将执行真正的IO操作。这时，我们输入的字符先存放到缓冲区中，等按下回车键换行时才进行实际的IO操作。典型代表是标准输入缓冲区

（`stdin`）和标准输出缓冲区（`stdout`）如上面中的例子所示，我们向标准输入缓冲区中放入的字符为10\n'，输入`"\n"`（回车）后，scanf函数才开始匹配， scanf函数中的%d匹配整型数10，然后放入变量i中，接着进行打印输出，这时`'\n'`仍然在标准输入缓冲区（stdin）内，如果第二个 scanf函数为 `scanf（"%d"，&i）`，那么依然会发生阻塞，因为 scanf函数在读取**整型数、浮点数、字符串**（后面介绍数组时讲解字符串）时，会忽略"\n'（回车符）、空格符等字符（忽略是指 scanf函数执行时会首先删除这些字符，然后再阻塞）. scanf函数匹配一个字符时，会在缓冲区删除对应的字符。因为在执行 `scanf（"%c"，&c)` 语句时，不会忽略任何字符，所以 `scanf（"%c"，&c）` 读取了还在缓冲区中残留的`"\n"`

printf() 的输出运用

- `printf ()` 函数可以输出各种类型 的数据，包括整型、浮点型、字符型、字符串型等，实际原理是 `printf ()` 函数将这些类型的**数据格式化为字符串**后，放入标准输出缓冲区，然后通过`\n'`来刷新标准输出，并将结果显示到屏幕上
- 位于%和格式化命令之间的一个整数被称为**最小字段宽度说明符**，通常会加上足够多的空格或0使输出足够长。如果想填充0，那么就在最小字段宽度说明符前面放置0 .
- 另外，也可以使用一个精度修饰符，精度修饰符根据使用的格式代码的不同通常有着不同的含义·
- 用`%f` 精度修饰符指定想要的小数位数。例如，`%5.2f`会至少显示5位数字并带有2位小数的浮点数
- 用`%s`精度修饰符简单地表示一个最大的长度，以补充句点前的最小字段长度`printf`函数的所有输出都是右对齐的，除非在%符号后放置了负号。例如，`%-5.2f`会显示5位字符、2位小数位的浮点数并且左对齐

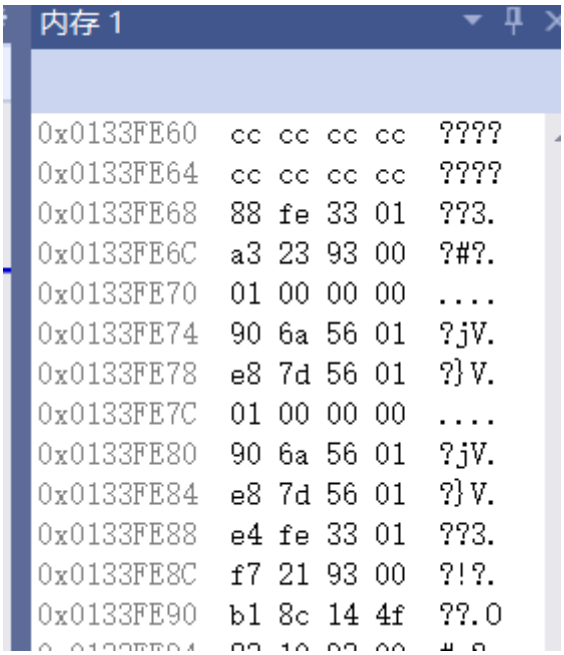
Day05 scanf() 循环读取

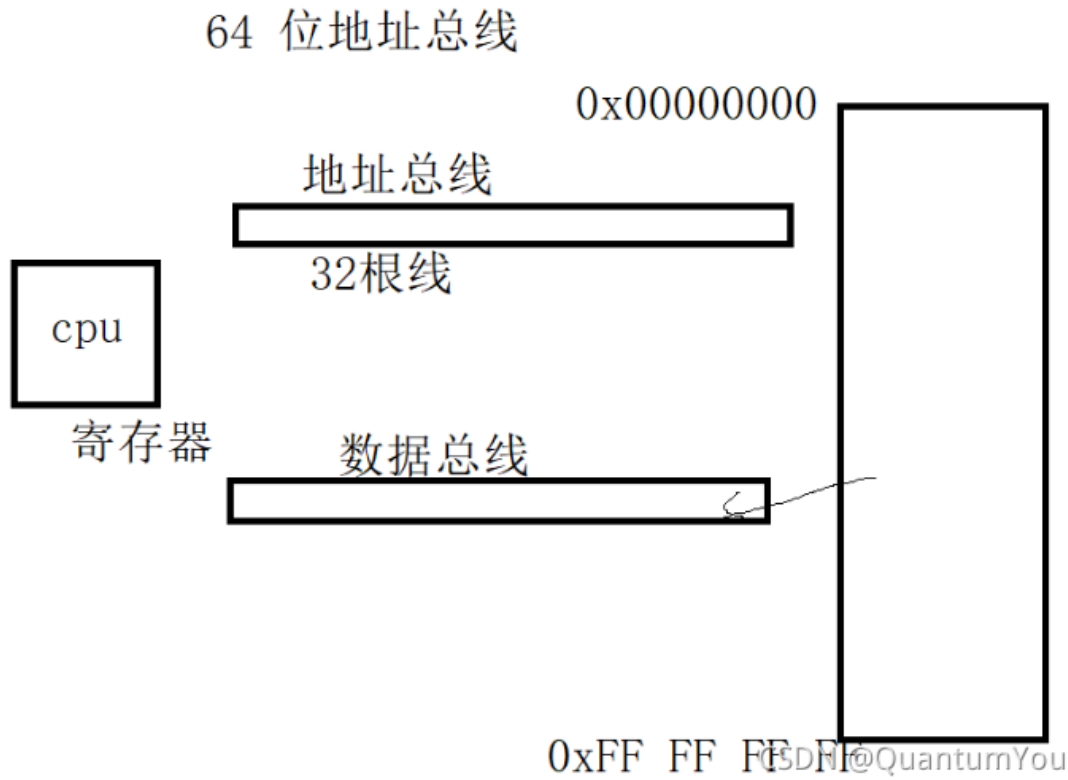
内存地址原理解析

- 32位地址总线 32位数据总线

比如： `0x  00 B5 F7 0F`

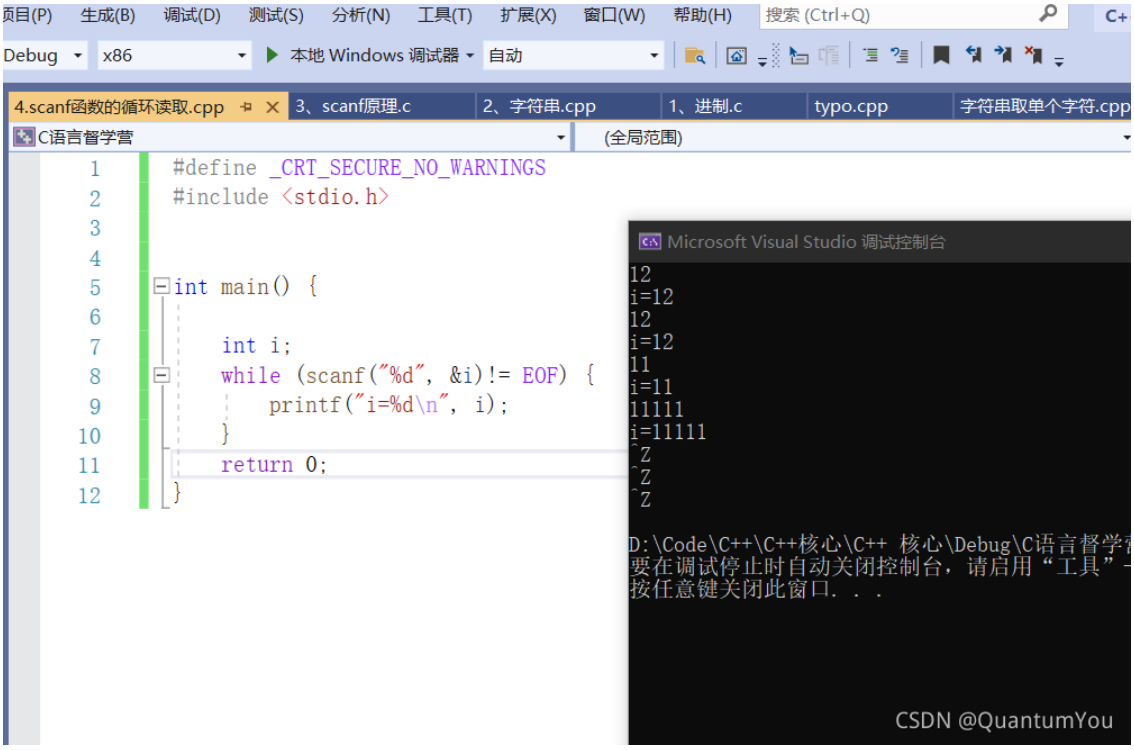
```
0000 0000 1011 0101 1111 0111 0000 1111
```



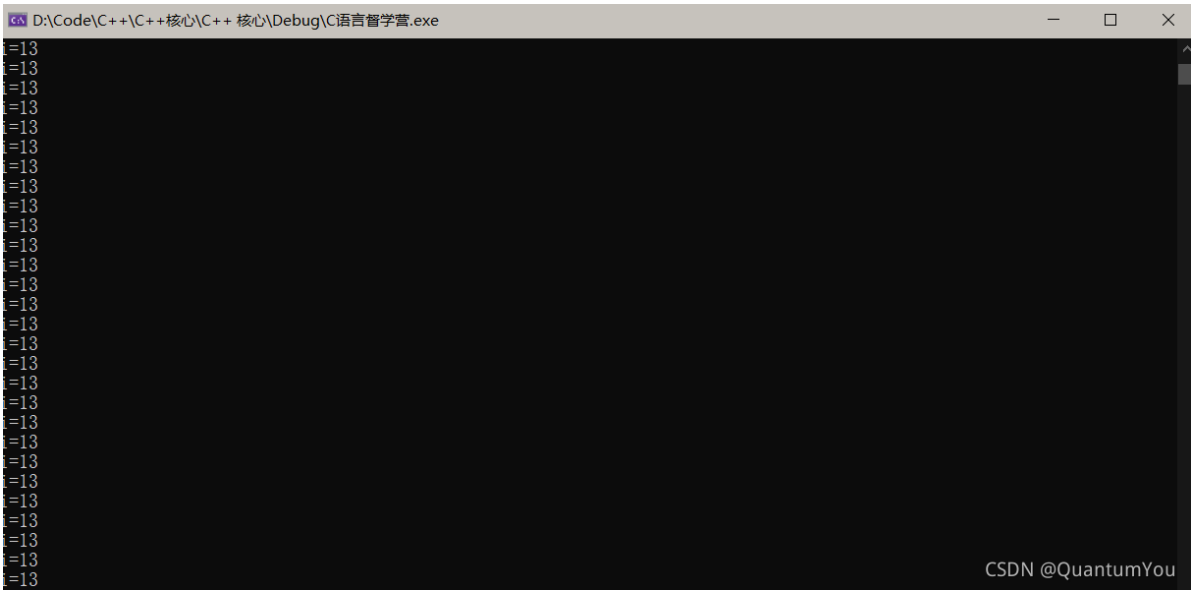


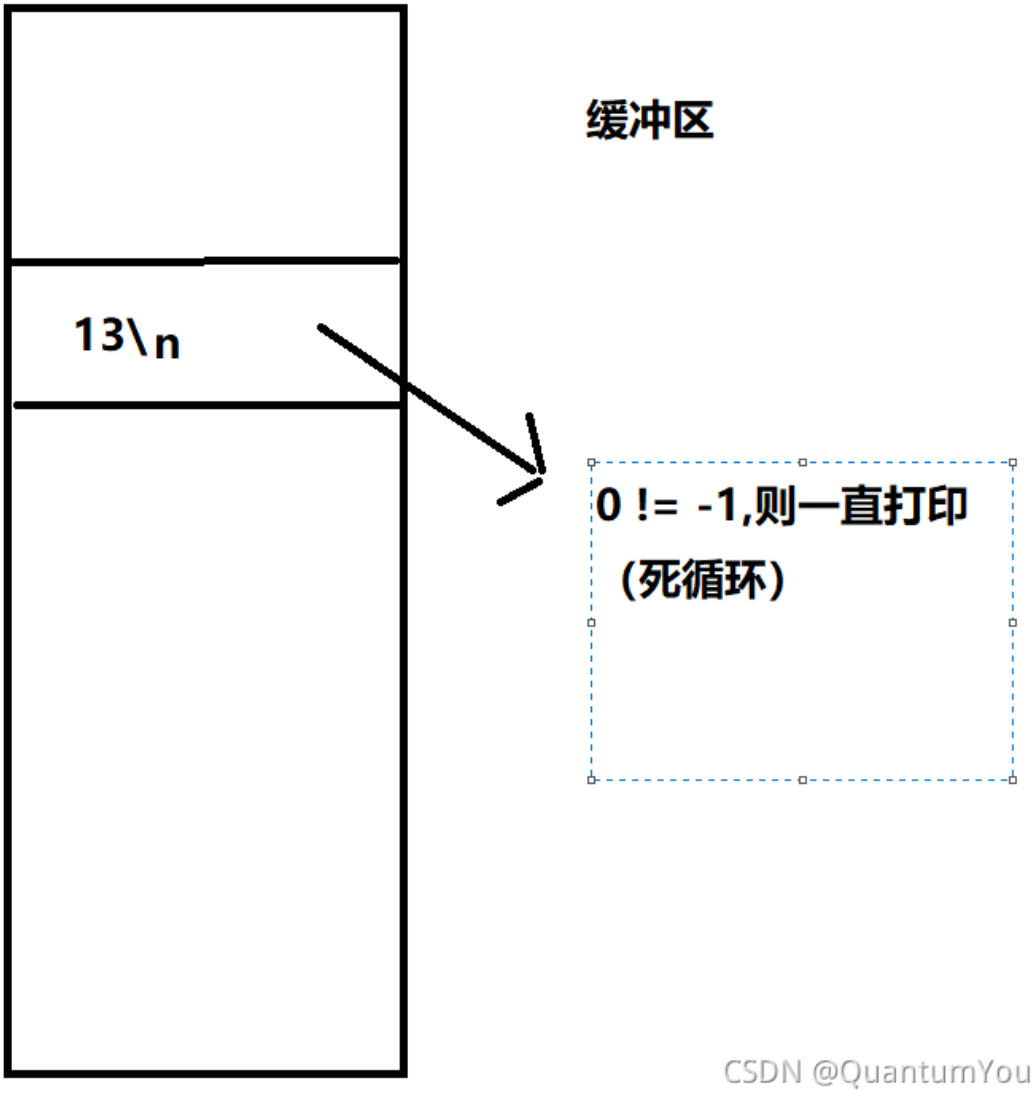
scanf() 循环读取原理

- 如下图所示，如果想输入多个整数（每次输入都回车），让 scanf函数读取并打印输出，那么我们需要一个 While循环.代码中为什么要加入 fflush（ stdin）函数呢？因为 fflush函数具有刷新（清空）标准输入缓冲区的作用。如果我们输错了，输入的为字符型数据，那么 scanf函数就无法匹配成功， scanf函数未匹配成功时其返回值为0，即ret的值为0，但这并不等于EOF，因为EOF的返回值为-1.当scanf函数无法匹配成功时，程序仍然会进入循环，这时会导致不断地重复打印。 最后我们按组合键Ctrl+Z （3次），让 scanf函数匹配失败，循环结束。



- **关于疯狂打印的原理解析**





- 如果scanf()遇到一个字符集(用%[]控制字符表示), 那么在括号中的任意字符都会读取到变量中. scanf()的返回值是成功赋值的变量数量, 发生错误时返回EOF.
- 解决办法，使用清空缓冲区 `rewind(stdin)`

- `rewind(stdin)` 一般用于整型和浮点型的时候

> 注意： scanf () 混合输入,读取多种类型的数据，混合输入时每次在%c之前需要加入一个空格

- `printf()` 控制输出格式，%4.2 ，代表四个字符（包括小数点），小数点后两位。

getchar函数介绍（了解）

- 使用 `getchar()` 函数可以一次从标准输入读取一个字符，它等价于 `char a, scanf("%c",&c)`

```
#include <stdio. h>
#include <stdlib. h>
int main {
 char c
 c=getchar() ;
 printf("you input alphabet=%c\n",c)
 system("pause")
}
```

getchar() 函数介绍（了解）

```
#include <stdio.h>
#include <stdlib.h>
int main() {

    char a, b, c;
    a = 'a';
    b = 'b';
    c = 'c';
    putchar(a);
    //putchar('\b');  //输出转义字符  退格键
    putchar(b);
    putchar(c);
    putchar('\n');  //输出转义字符
    system("pause");
}

```

Day06 运算符与表达式

运算符的分类

C 语言提供了 13 种类型的运算符，如下所示。

(1) 算术运算符 (+ – * / %)。

(2) 关系运算符 (> < == >= <= !=)。

(3) 逻辑运算符 (! && ||)。

(4) 位运算符 (<< >> ~ | ^ &)。

(5) 赋值运算符 (=及其扩展赋值运算符)。

(6) 条件运算符 (?:)。

(7) 逗号运算符 (,)。

(8) 指针运算符 (*和&)。

(9) 求字节数运算符 (sizeof)。

(10) 强制类型转换运算符 ((类型))。

(11) 分量运算符 (. ->)。

(12) 下标运算符 ([])。

(13) 其他 (如函数调用运算符())。

- 注意：关于逻辑运算符的使用易错项

- 类似于“ 逻辑短路 ”

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>


int main()
{
    int a = 2;
    if (3 < a < 10) //如果要判断 3<a 同时 a<10,要用逻辑运算符
    {
        printf("a is right\n");
    }
    else {
        printf("a is wrong");
    }
```


- 在上述的例子中无论 a 输入什么值， 都是输出 `a is right` ；

关于判断两个浮点数是否相等

```c
//判断两个浮点数是否相等,
float f = 234.56;
if (f == 234.56)
{
    printf("f is equal to 234.56\n");
}
else {
    printf("f is not equal to 234.56\n");
}
```


- 注意： 上述的输出为 下者，原因在于IEEE 的浮点数（计算机组成原理的相关知识）

正确写法如下：

```
    //判断两个浮点数是否相等,必须用下面的方法
    float f = 234.56;
    if (f - 234.56>-0.0001 && f-234.56<0.0001)
    {
        printf("f is equal to 234.56\n");
    }
    else {
        printf("f is not equal to 234.56\n");
    }
}
```


逻辑表达式

- 注意： 在C语言中没有布尔值，只有真与假（即为零和非零）
- 关于逻辑非的，注意事项，`int j = 10 , int i = !! j ;` i 的最后的值并不是负负得正为 `10` 而是 `1`

关于类型转化栈溢出

- 如下代码

```
int main(){
   char a ;
   scanf("%d",%a) ;
   printf("%c\n",a) ;
}
```

- 上述代码发生如下报错

- 通过调试，发现我们操作的空间超出了变量本身占用的空间。

- 正确代码写法如下：

```
int main(){
   int  a ;
   scanf("%d",%a) ;
   printf("%c\n",a) ;
}
```

整型数在0-128之间可以用%c 输出