

哈工大计算机专业考研复试

C 语言复习笔记

参考资料：C 语言程序设计第三版（苏小红），高等教育出版社
C 语言程序设计学习指导（苏小红），高等教育出版社

适合于有一定 C 语言基础的学者

编译环境：CodeBlocks

2018 年 2 月 28 日

~ 1 ~

FanfanWang

目录

哈工大计算机专业考研复试 C 语言复习笔记	1
第一章 为什么要学 C 语言	3
第二章 C 语言数据类型	3
第三章 简单的算术运算和表达式	4
第四章 键盘输入和屏幕输出	5
第五章 选择控制结构	6
第六章 循环控制结构	8
第七章 函数	9
第八章 数组	9



第一章 为什么要学 C 语言

不为什么，因为考试要考。

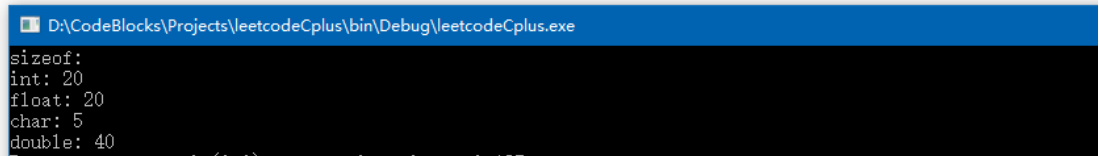
第二章 C 语言数据类型

C 语言标识符命名规则

- 只能以英文字母、数字、下划线组成；
- 必须以字母、下划线开头；
- 不允许使用关键字命名；
- 有最大长度限制。

Sizeof 的使用，返回数组的总大小，从下面的程序可以看出在 32 位系统中，一个 char 类型字符占 1 字节，int 和 float 占 4 字节，double 占 8 字节。在 16 位系统中，int 类型占两个字节。

```
int i[5] = {1,2,3,4,5};
float f[5] = {1,2,3,4,5};
char c[5] = {'a','b','c','d','e'};
double t[5] = {1,2,3,4,5};
printf("sizeof: \nint: %d\nfloat: %d\nchar: %d\ndouble: %d", sizeof(i), sizeof(f), sizeof(c), sizeof(t));
return 0;
```



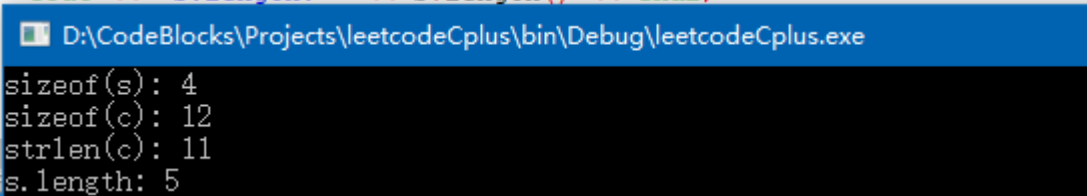
这个现象非常奇妙，区分 C++ 中两种声明字符串的方法。

Char c[] = "hello world"; 语句以字符数组的形式存储字符串，sizeof(c) 返回的是该字符数组中包括 '\0' 的所有字符个数，这种声明方式下要获取字符串长度要用 strlen() [其实可以理解为 C 语言的方式]；

String s = "hello"; 语句可以理解为类似于链表的声明结构，s 只代表该字符串的起始单元地址，故 sizeof(s) 只有 4 个字节，这种声明方式下获取字符串长度要用 .length() [其实可以理解为 C++ 的方式，对象引用]；

strlen() 与 length() 的适用对象不能互换，strlen() 要引用头文件 string.h。

```
char c[] = "hello world";
string s = "hello";
cout << "sizeof(s): " << sizeof(s) << endl;
cout << "sizeof(c): " << sizeof(c) << endl;
cout << "strlen(c): " << strlen(c) << endl;
cout << "s.length: " << s.length() << endl;
```



对于其他类型数组，想要获取数组长度，下面的语句是最快的方式：

```
Int len = sizeof(array) / sizeof(ElemType);
```



第三章 简单的算术运算和表达式

定义宏常量，宏常量又称字符常量，在程序中任何位置遇到宏常量都会把标识符替换为之后的字符串（这个过程称为宏替换），定义宏常量不用分号结尾：

```
#define 标识符 字符串
#define PI 3.1415
```

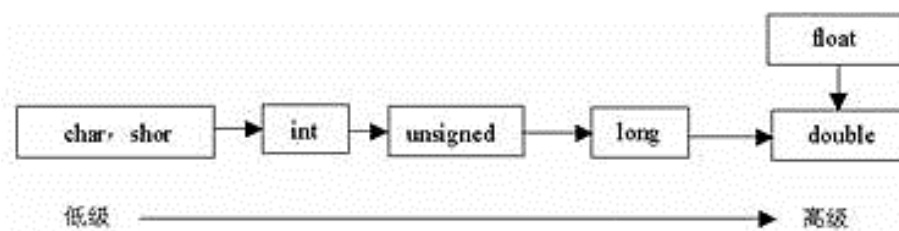
宏常量没有数据类型，编译时只进行简单的字符串替换；

Const 常量只读，只能在定义时赋初值，之后不可以修改，要加分号：

```
const double PI = 3.14159;
```

类型转换

自动类型转换，类型提升不会损失精度，可以默认进行：



赋值中会出现自动类型转换：

```
int n = 3;
float f = 9.3;
double d = 7.8864728364728;
printf("%d , %f , %lf\n", n, f, d);
cout << "-----" << endl;
n = f;
f = n;
d = f;
printf("%d , %f , %lf", n, f, d);
```

```

D:\CodeBlocks\Projects\leetcodeCplus\bin\Debug\leetcodeCplus.exe
3 , 9.300000 , 7.886473
-----
9 , 9.000000 , 9.000000

```

强制类型转换

(类型) 表达式

常用数学标准函数

```
#include <math.h>

sqrt(x) //平方根, x>=0
exp(x) //e^x
fabs(x) //绝对值
pow(x,y) //x^y
log(x) //lnx
log10(x) //lgx
sin(x)
cos(x)
```

常见错误



1/2 是整数除法，结果是 0，若想得到 0.5，需要进行强制类型转化变为浮点型进行计算。

浮点数不能进行求余运算

Float(x)这种强制转化不能改变 m 的数值和类型

```
int n = 3;
float f = 9.3;
cout << n << endl << f << endl;
n = (int) f;
cout << n << endl << f << endl;
```

```
D:\CodeBlocks\Projects\leetcodeCplus\bin\Debug\leetcodeCplus.exe
3
9.3
9
9.3
```

在进行强制类型转换时，由浮点型转为整型时对小数部分直接进行舍去，而不是四舍五入

```
float f = 9.3;
float t = 9.7;
cout << (int) f << endl << (int) t << endl;
```

```
D:\CodeBlocks\Projects\leetcodeCplus\bin\Debug\leetcodeCplus.exe
9
9
```

双引号括起来的字符串与宏名相同时不替换，因为宏定义不检查语法

算术表达式不能用++、--运算，(a+b) ++是错误的。

第四章 键盘输入和屏幕输出

转义字符

单引号 '\', 双引号 '\"'。转义字符按照单个字符计数

```
string str = "abc\"d";
cout << "the length of string is " << str.length() << endl;
```

```
D:\CodeBlocks\Projects\leetcodeCplus\bin\Debug\leetcodeCplus.exe
the length of string is 5
```

在 ASCII 码的取值范围中，char 类型和 int 类型可以进行相互类型转换不丢失信息，二者可以进行混合运算。

```
char c = 'a';
cout << c << endl << c+5 << endl << char(c+5) << endl;
```

```
D:\CodeBlocks\Projects\leetcodeCplus\bin\Debug\leetcodeCplus.exe
a
102
f
```

大小写转换



```
char cLow = 'a';  
char cHigh = 'A';  
cLow = cLow - 32; //大写->小写 减32  
cHigh = cHigh + 32; //小写->大写 加32
```

D:\CodeBlocks\Projects\leetcodeCplus\bin\Debug\leetcodeCplus.exe

A
a

Getchar 的返回值为键盘输入的字符

```
char c;  
c = getchar();  
putchar(c+32);
```

D:\CodeBlocks\Projects\leetcodeCplus\bin\Debug\leetcodeCplus.exe

A
a

格式化输入

```
int a,b;  
scanf("a = %d,b = %d",&a,&b);  
cout << a << endl << b << endl; //忽略对应的格式  
scanf("%2d%2d",&a,&b); //指定读入字符宽度  
cout << a << endl << b << endl;  
scanf("%d%c%d",&a,&b); //以任意字符分割  
cout << a << endl << b << endl;
```

D:\CodeBlocks\Projects\leetcodeCplus\bin\Debug\leetcodeCplus.exe

a = 12,b = 23
12
23
1234
12
34
11-34
11
34

Scanf 函数的返回值

```
cout << scanf("%d",&a) << endl;
```

D:\CodeBlocks\Projects\leetcodeCplus\bin\Debug\leetcodeCplus.exe

A
0

若读取失败，scanf 函数返回值为 EOF，EOF 为定义的宏常量，值为-1。

在 scanf 函数读取字符时，空格、转义字符、回车都会被当作有效字符读入。

第五章 选择控制结构

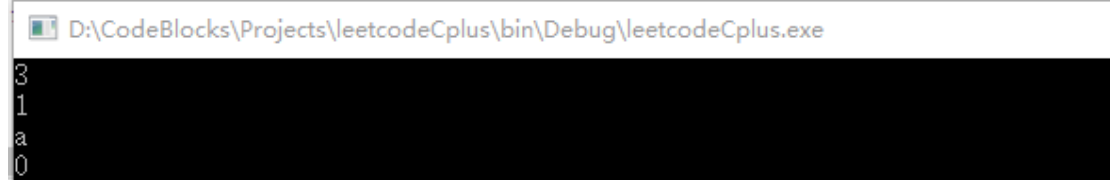
条件运算符是 C 语言中唯一一个三元运算符



```
1 max = a > b ? a : b
2
3 //equals to
4 if (a>b)
5     max = a;
6 else
7     max = b;
```

Scanf 函数不进行参数类型匹配检查，scanf 函数本身带有返回值，当正确读入数据时，返回值为已成功读入的数据项数，当数据类型错误，读取失败时，返回值为 0。出现读取失败时，此后所有的 scanf 都不能正确进行。

```
int a;
cout << scanf("%d",&a) << endl; //合法输入
cout << scanf("%d",&a) << endl; //非法输入
```

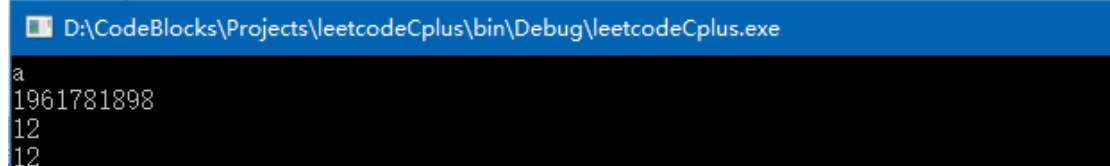


```
D:\CodeBlocks\Projects\leetcodeCplus\bin\Debug\leetcodeCplus.exe
a
3
a
1
a
0
```

值得注意的是，浮点型读入整型变量时，会读入整数位，而不是进行四舍五入，应格外注意。此外，字符对于数字而言是非法输入，数字对于字符而言则可以正常读入。

当出现输入错误时，可以使用 fflush()函数清除输入缓冲区中的内容

```
int a;
scanf("%d",&a);
cout << a << endl;
fflush(stdin);
scanf("%d",&a);
cout << a;
```



```
D:\CodeBlocks\Projects\leetcodeCplus\bin\Debug\leetcodeCplus.exe
a
1961781898
a
12
```

位运算

操作符	作用
-----	----

&	位逻辑与
	位逻辑或
^	位逻辑异或
~	位逻辑反
>>	右移（不是循环移位）
<<	左移（不是循环移位）

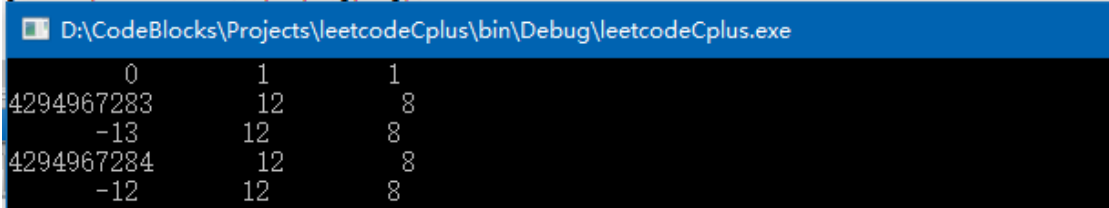
当进行按位与或时，最好使用 16 进制，在程序中这样表示：0x01 表示 0000 0001。所以，字符类型 a 的最高位强制 1 可以这样：a=a | 0x80。

看下面的程序，对于运算符&&和 ||，是针对十进制的数值进行运算的，只有 0||0=0，其他



均为非 0，4294967283 是 -13 的补码。此外，还需注意 ~ 运算符和 - 运算符的区别，~ 是按位取反，- 是取相反数。

```
int x = 12, y = 8;
printf("%d%d%d\n", !x, x | y, x & y);
printf("%u%d%d\n", ~x, x | y, x & y);
printf("%d%d%d\n", ~x, x | y, x & y);
printf("%u%d%d\n", -x, x | y, x & y);
printf("%d%d%d\n", -x, x | y, x & y);
```



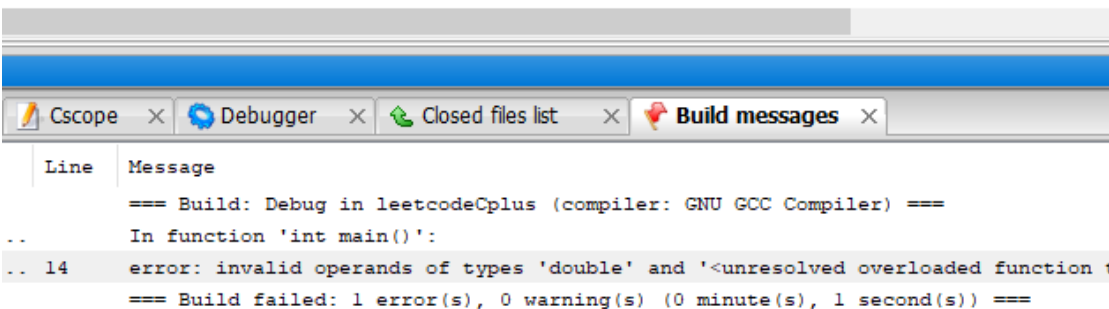
0	1	1
4294967283	12	8
-13	12	8
4294967284	12	8
-12	12	8

关系运算符的优先级低于算术运算符。

条件运算符？：为唯一的三元运算符。

测试浮点数是否相等不能 == 运算符，会出现编译错误，此时应使用两数相减小于一个极小数的办法。

```
float a = 1.1;
cout << a == 1.1 << endl;
```



Line	Message
..	=== Build: Debug in leetcodeCplus (compiler: GNU GCC Compiler) ===
..	In function 'int main()':
.. 14	error: invalid operands of types 'double' and '<unresolved overloaded function name>'
	=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ===

计算三角形面积海伦公式 $S = 1/4 \sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}$

第六章 循环控制结构

While 语句后加分号会导致死循环。

```
while (i < n);
```

当第一次测试条件就为假时，while 语句与 do-while 语句是不等价的，do 语句会执行一次循环内容，其余情况二者相同。

生成随机数

```
srand(time(NULL));
int n = rand() % 100 + 1 // 1~101之间的随机数
```


第七章 函数

全局变量会破坏函数的封装性。

变量的存储类型是指编译器为变量分配内存的方式

- 自动变量，又称动态局部变量，在每次进入函数时为其重新分配内存空间，函数结束时释放空间，作用于仅在函数内部。**在不同并列语句块中可以定义相同名字的变量，不会相互干扰，因为它们占据不同的内存单元，具有不同的作用域。**
如果不希望在函数内改变变量的值，只需用 `const` 修饰变量即可，将形参变为常量。
- 静态变量，`static`，静态变量与程序生命周期一致，动态变量与程序块生命周期一致。自动变量在定义时不会被自动初始化。静态局部变量与自动变量的作用域相同，但退出函数时静态局部变量不会被销毁，在下次进入函数时仍然可以使用，并保存着上次退出时的值。
- 外部变量，`extern`，保存在静态存储区内，生存周期与整个程序相同，没有现实初始化的外部变量会被自动初始化为 0。
- 寄存器变量，`register`，将使用频率高的变量放入寄存器中，执行速度更快。

程序模块化分解的基本原则：高聚合，低耦合，保证每个模块的相对独立性。

第八章 数组

一维数组在内存中占用的字节数为：数组长度 \times `sizeof`（基类型），二维数组占用的字节数为第一维长度 \times 第二维长度 \times `sizeof`（基类型）。

当形参被声明为二维数组时，可以省略第一维的长度说明，但第二维的长度说明不能省略。