

## 第二次实验报告

2018.10.2 PB16000702 韦璐

实验目的：

1. 数据类型和存储形式
2. 整数，实数，单个字符与字符串常量的表示形式
3. 常量，变量的定义
4. 运算符优先级和结合方向
5. 表达式类型和求值
6. 运算过程与类型转换
7. 输入输出

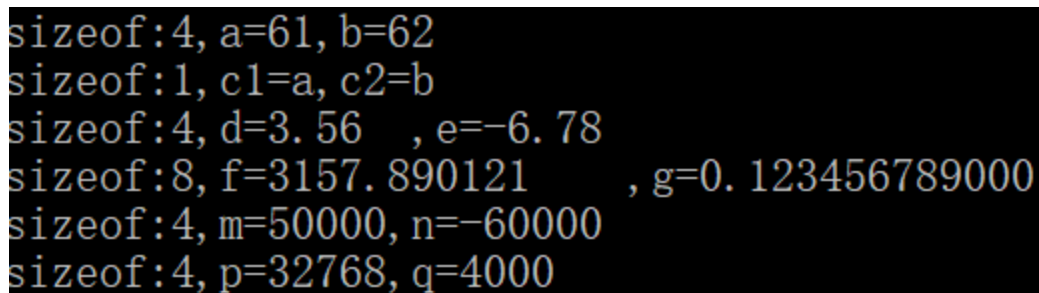
第一题

实验代码：

```
#include<stdio.h>

main(){
    int a,b;
    char c1,c2;
    float d,e;
    double f,g;
    long m,n;
    unsigned int p,q;
    a=61;b=62;
    c1='a';c2='b';
    d=3.56;e=-6.78;
    f=3157.890121;g=0.123456789;
    m=50000;n=-60000;
    p=32768;q=4000;
    printf("sizeof:%d,a=%d,b=%d\n",sizeof(int),a,b);
    printf("sizeof:%d,c1=%c,c2=%c\n",sizeof(char),c1,c2);
    printf("sizeof:%d,d=%-6.2f,e=%-6.2f\n",sizeof(float),d,e);
    printf("sizeof:%d,f=%-15.6f,g=%-15.12f\n",sizeof(double),f,g);
    printf("sizeof:%d,m=%ld,n=%ld\n",sizeof(long),m,n);
    printf("sizeof:%d,p=%u,q=%u\n",sizeof(unsigned),p,q);
}
```

调试分析：



```
sizeof:4, a=61, b=62
sizeof:1, c1=a, c2=b
sizeof:4, d=3.56, e=-6.78
sizeof:8, f=3157.890121, g=0.123456789000
sizeof:4, m=50000, n=-60000
sizeof:4, p=32768, q=4000
```

观察上面的输出结果，

- (1) 不同的数据需要不同的输出格式
- (2) 可以通过输出格式规定带有小数点的数据的输出方式
- (3) 向左对齐输出的时候即使数据后面有空位也不会被补全

实验总结:

1. 数据与类型: 常数, 变量, 函数和表达式是程序中的基本操作对象, 都隐式或显式地与一中数据类型相联系。数据类型是指数据的内在表现形式, 数据类型确定数据的存储形式和值域, 数据类型也确定相应数据所允许执行的运算等。
2. C 语言的数据类型: 有整型, 字符型, 实型 (单精度和双精度), 枚举类型, 数组类型, 结构体类型, 共用体类型, 指针类型和空类型。
3. 不同类型的数据在计算机中的存储格式也不同
  - ✚ 整数 (int) 在 32 位机以 4 字节 32 位存放, 存放形式是除了首位 (第三十一位) 以零表示正一表示负以外, 其他三十一位都是用二进制数表示数值。
    - ✧ 存储特点: 顺序存放, 精确表示。
    - ✧ Int 型整数占内存的一个字长, 所以在 32 位机上, int 型变量占四个字节; 在 16 位机上, int 型变量占二个字节。系统不同, 同类型的变量所占字节数也不同。
    - ✧ 整数的表示可以分为带符号和不带符号数两类。对于带符号的整数, 存储在计算机中的形式是该数的补码。
    - ✧ 注: 原码: 最高位为符号位 (正数为零, 负数为一), 其余位表示数值; 反码: 正数的反码同原码, 负数的反码就是除了符号位以外取反 (一变零, 零变一); 补码: 正数的补码同原码, 负数的补码等于反码加一。
  - ✚ 实数 (float) 在内存中一律以指数形式存放, 占四字节三十二位存放, 其存放形式分为两个部分, : 指数部分 (阶码) 和数值部分 (尾数)。这里, 以实数  $28.375_{10}=11100.011_2=0.11100011 \times 2^{101}$  (标准指数形式) 为例: 第一个字节, 也就是前面八位, 最高位是表示正负的, 其余的用来存放指数部分, 也就是阶码, 后面三个字节, 也就是数值存放的部分, 它的首位也是用来表示正负, 这里的正负可以表示整个数值的符号。它的存储特点是指数和小数分别存放, 近似表示。
  - ✚ 字符型 (char) 数据以一字节八位存放, 首位存放正负, 其他位存放 ASCII 码的二进制形式。
  - ✚ 字符串是用一对双引号括起来的零个或多个字符序列, 采用数组形式存放, 就是每个字符串中的每个字符占用一个字节, 汉字就两个, 按照从左到右的顺序依次存储在一段连续的内存空间中, 编译系统自动为串加串结束符 '\0', 以表示串的结束。要注意的是因为字符串的末尾都被加上了串结束符, 所以字符串在内部表示所占的空间要比实际字符多一个字节, 这个字节里面的每一位都是零。
4. 整常数的表示
  - ✚ 十进制整常数的取值范围是  $-2^{n-1} \sim 2^{n-1}-1$ ,  $n$  为机器字位数 (字长);
  - ✚ 长整数常数: 取值范围  $-2^{2n-1} \sim 2^{2n-1}-1$ , 占两个机器字长 (也不一定, 要看具体的系统), 表示形式是在整常数后加字母 "l" 或 "L"。
  - ✚ 八进制整常数: 由数字零打头, 后跟一个八进制数。例如: 05
  - ✚ 十六进制整常数: 以 "0x" 打头, 后跟一个十六进制数。
5. 实常数的表示: 在 c 语言中单精度和双精度实常数的表示方法相同, 编译程序总是把实常数处理成双精度后再进行运算。

实常数由三部分, 整数小数和指数部分组成, 它的组成规则是三个部分可以有一部分或两部分两部分缺省, 但整数部分和小数部分不能同时缺省, 如果一个实常数带小数

点，则小数点左右应至少有一边有数字，如果实常数带 e 或 E，则小数点两边都至少有一位数字，而且指数部分必须是整数。所以实常数一般有两种书写形式，小数形式和指数形式，实数的取值范围是  $10^{-38} \sim 10^{38}$ 。当数值太大或太小会产生溢出，一般一个普通的实常数在机内表示时具有 6~7 位十进制有效数字，双精度实数具有 16 位十进制有效数字。

## 6. 字符常数

字符数据包括单个字符（简称字符）和字符串两种。

✚ 单个字符常数是成由一对单引号括起来的单个字符（转义字符虽然从形式上像一个字符串，但实质上仍然是单个字符，它是为了表示一些特殊字符，如回车换行符等，不像字母可以直接实现），在内存中占一个字节存储单元。

转义字符的三种形式

✧ \后面跟着一个特定字符。

✧ \后面跟一个 1~3 位的八进制数，鸡肋表示法

✧ \x 后面跟一个一到二位的十六进制数

## 7. 符号常数

为了提高程序的可读性和常数修改的一致性，c 语言允许将程序中多次出现的常数定义为一个标识符，称之为符号常数，这个要先定义后使用，一般在程序的开头定义，定义的形式是：#define 标识符 常数 习惯上用大写字母表示。

8. 标识符的作用与命名规则：作用是标记常数，变量，自定义数据类型，函数及程序的名字。命名规则是以下划线或任意字符打头，在第一个字符以后，可以是任意字母，下划线或数字组成的字符序列，可以是空串。注意不要和已有的冲突，大小写字母是两个不同的标识符，一般除了符号常数都是小写字母，标识符长度不限，但只识别前八个字符。

9. 变量的定义：变量就是在程序运行过程中数值可以改变的量，其实是内存中的一个存储单元，必须先定义后使用，定义名称类型初始值和属性，变量的属性是规定了变量的作用域和存储类别，定义以后，这个变量的取值范围，能执行的运算操作，存储格式和单元就都规定了。

## 10. 运算符优先级和结合方向：

✚ 同一优先级的运算符级别相同，运算次序由结合方向决定

✚ 不同的运算符要求有不同的运算对象，按运算符所要求的对象个数分类，有单目，双目，三目

✚ C 编译程序处理的时候会尽量从左到右将若干字符组成运算符

## 11. 表达式的组成：

✚ 表达式可以有常数，变量，函数调用运算符和圆括号内的表达式组成

✚ 单个常数，变量或者单个函数调用都可以看作一个表达式

✚ 表达式运算的最后结果作为表达式的值，该值的类型就是表达式的类型

## 12. 算术运算符和算术表达式：加减乘除和%（模或者取余）

✚ 算术运算符的优先级：从高到低是负号 乘除% 加减

✚ 算术运算符的结合性是从左向右

✚ %运算符的两个操作数必须是整型

✚ 两个整型量相除，商必须是整数

✚ 可以进行整型，单精度双精度混合运算，此时先将整型，单精度量都转变为双精度，再进行运算。

✚ 表达式的计算和类型的转换都是逐步进行的

✚ 计算时应特别注意运算符的三个特性以及数据类型，特别是除法运算。

### 13. 自增运算符（减）

- ✚ 自增自减都是单目运算符，只能是变量
- ✚ 前缀自增减优先级是 2，结合方向右结合。后缀自增减优先级是一，左结合
- ✚ 自增减优先级最高
- ✚ 自增减使变量加减一但是对表达式的运算结果的影响与自增自减运算符的前置后置有关。
- ✚ 自增减出现在表达式中数目大于一的时候，自增减运算符前置时变量先自增或者自减，再以变化后的量参与表达式中的其他运算，后置的时候，先参与表达式的其他运算再自增自减，自增减单一的时候前置后置相同的。
- ✚ 出现多个自增减时候，自左向右取尽可能多的表达式参与
- ✚ 多数系统函数参数的求值顺序从右向左的，所以当你把 i 和 i++ 两个表达式并列，其实先计算右边的，所以左边的 i 也是做了加法以后的值。

### 14. 关系运算符和关系表达式

小于，小于等于，大于，大于等于，等于，不等于

- ✚ 关系运算符的结合性为从左到右，
- ✚ 综合运算时时候相关运算符的优先级时算数运算符大于 大于大于等于小于小于等于 大于等于不等于 大于赋值运算符
- ✚ 关系成立为真的时候，关系表达式的值为一，表示关系不成立，值是零。所以 c 的表达式值可以看作是整型量参与运算。
- ✚ 在 c 语言中，可以把表达式理解为一个条件，非零表达式是真，零是假就是表达式不成立
- ✚ 表达式的运算是逐步进行的
- ✚ 正确区分赋值运算符和关系运算符
- ✚ 不要直接进行实型量相等的比较，因为实型量是近似值

### 15. 逻辑运算符和逻辑表达式

C 语言提供的逻辑运算符有：！（逻辑非）&&（逻辑与）||（逻辑或）

- ✚ ！的结合性是从右向左，另外两个是从左向右
- ✚ 综合运算时相关运算符的优先级是！ 算数运算符 关系运算符 && || 赋值运算符
- ✚ C 编译系统在给出逻辑运算的结果的时候，真是给一，但是系统判断的时候只要不是零就都认为是真的。
- ✚ 根据结合性，对 && || 有规定：
  - ✧ 这两个运算符链接的表达式按照从左到右的顺序进行求值，也就是保证左边的运算分量优先得到运算，也就是说当左边的表达式能够确定整个逻辑表达式的值的时候就不再进行计算。要注意的是一旦判断进行了，那么那个未知数的值也就重新给定了。

### 16. 赋值运算符与赋值表达式

赋值表达式的形式：

左边的变量作为左操作数，中间的=是赋值运算符，右边的操作数是表达式

- ✚ 赋值运算符的结合方向是从右向左
- ✚ 左边一定是变量
- ✚ 赋值表达式的运算结果是将赋值运算符右边的 表达式的值赋给左边的变量
- ✚ 左边的变量的值和类型就是表达式的值和类型
- ✚ 若赋值运算符的两边操作数类型不一致，但都属于数值型和字符类型时，则=右边

的表达式值将自动转换成左边变量的类型（即赋值表达式的类型）

✚ 根据赋值表达式的定义，又可递归得到多重形式赋值表达式：

$l=j=k=0$  等价于 从右向左依次赋值

✚ 复合赋值运算符的组成形式：双目运算符

#### 17. 逗号运算符和逗号表达式

逗号表达式的一般形式是：1, 2, 3, ...。

✚ 结合性左右

✚ 优先级最低

✚ 连接两个或以上的表达式的时候最后一个表达式的值和类型作为该逗号表达式的值和类型

✚ 求解过程是从左往右求解

✚ 运算是逐步进行的，要注意变量值的变化

#### 18. 条件运算符和条件表达式

是一个三目运算符，结果：假如  $e_1$  成立，那么  $e_2$  的值是整个表达式的值，否则取  $e_3$  的值作为整个表达式的值。条件运算符的结合方向自右向左

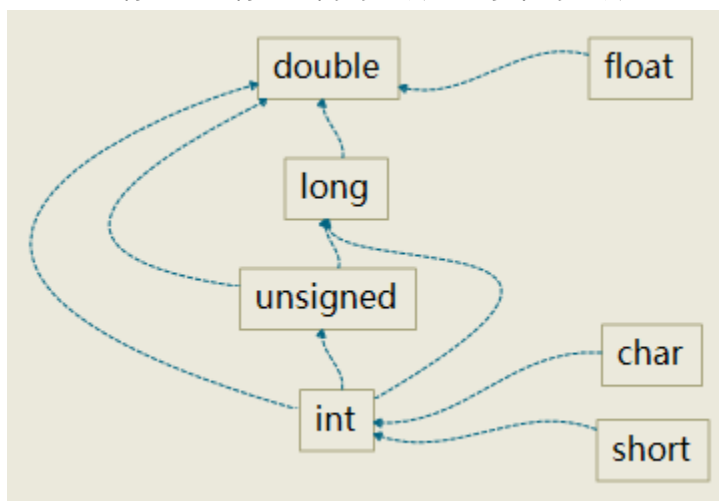
#### 19. 类型转换：

✚ 字符整型单双精度的混合运算时候类型转换的 三种规则是

✧ 属于同一类型，存储单元长度由短字节向长字节转换

✧ 整型和浮点型之间，低级向高级转换

✧ 整型量有符号与无符号之间的转换与长字节的转换



✧

✚ 而赋值运算的时候就不会管其他，右边表达式的类型一定要强制转换成左边变量的类型。

✚ 可以自己强制类型转换，就是直接将类型标识符放在表达式的前面

#### 20. 长度运算符

sizeof (类型标识符) sizeof 表达式

作用是给出运算对象以字节为单位的存储单元的长度

#### 21. 表达式语句

语法形式 表达式

就是后面加分号构成的语句，其中分号是语句的终结符，是语句的组成部分。

#### 22. C 语言的运算符，优先级，结合性和运算对象

#### 23. 输入输出补充

Putchar getchar

## 习题一

### 实验代码

```
#include<stdio.h>
main(){
    int a,b;
    char c1,c2;
    float d,e;
    double f,g;
    long m,n;
    unsigned int p,q;
    scanf("%d%d%c%c%f",&a,&b,&c1,&c2,&d,&e);
    scanf("%lf,%lf,%ld,%ld,%u,%u",&f,&g,&m,&n,&p,&q);
    printf("sizeof:%d,a=%d,b=%d\n",sizeof(int),a,b);
    printf("sizeof:%d,c1=%c,c2=%c\n",sizeof(char),c1,c2);
    printf("sizeof:%d,d=%-6.2f,e=%-6.2f\n",sizeof(float),d,e);
    printf("sizeof:%d,f=%-15.6f,g=%-15.12f\n",sizeof(double),f,g);
    printf("sizeof:%d,m=%ld,n=%ld\n",sizeof(long),m,n);
    printf("sizeof:%d,p=%u,q=%u\n",sizeof(unsigned),p,q);
}

#include<stdio.h>
main(){
    int a,b;
    char c1,c2;
    float d,e;
    double f,g;
    long m,n;
    unsigned int p,q;
    scanf("%d%d%c%c%f",&a,&b,&c1,&c2,&d,&e);
    scanf("%f,%f,%d,%d,%u,%u",&f,&g,&m,&n,&p,&q);
    printf("sizeof:%d,a=%d,b=%d\n",sizeof(int),a,b);
    printf("sizeof:%d,c1=%c,c2=%c\n",sizeof(char),c1,c2);
    printf("sizeof:%d,d=%-6.2f,e=%-6.2f\n",sizeof(float),d,e);
    printf("sizeof:%d,f=%-15.6f,g=%-15.12f\n",sizeof(double),f,g);
    printf("sizeof:%d,m=%ld,n=%ld\n",sizeof(long),m,n);
    printf("sizeof:%d,p=%u,q=%u\n",sizeof(unsigned),p,q);
}
```

### 调试分析

第一个程序告诉我们，对于每一个 scanf 的输入要小心，除了字母可以什么都不加以外，其他的都要空格，假如还有别的间隔符号，也一定要加上。对于第二个程序，就是告诉我们 lf ld f d 的区别，在使用 d 的时候，绝对值大于等于 32768 的数字都要溢出。另外 lf 是双精度输入，而 f 是单精度输入，所以由于小数点后面的位数太多了，所以会导致输出错误，f 是对应 float 的输出的，但是对于 printf 而言，二者没有区别，都用 f 进行输入。

### 实验结果

```
61 62ab3.56 -6.78
3157.890121, 0.123456789, 50000, -60000, 32768, 4000
sizeof:4, a=61, b=62
sizeof:1, c1=a, c2=b
sizeof:4, d=3.56 , e=-6.78
sizeof:8, f=3157.890121 , g=0.123456789000
sizeof:4, m=50000, n=-60000
sizeof:4, p=32768, q=4000
```

```
61 62ab3.56 -6.78
3157.890121, 0.123456789, 50000, -60000, 32768, 4000
sizeof:4, a=61, b=62
sizeof:1, c1=a, c2=b
sizeof:4, d=3.56 , e=-6.78
sizeof:8, f=0.000000 , g=0.000000000000
sizeof:4, m=50000, n=-60000
sizeof:4, p=32768, q=4000
```

实验总结

无

习题二

实验代码

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

```
main(){
```

```
    int i,j,m,n;
```

```
    i=8;j=10;
```

```
    m=++i;n=++j;
```

```
    printf("%d,%d,%d,%d\n",i,j,m,n);
```

```
}
```

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

```
main(){
```

```
    int i,j,m,n;
```

```
    i=8;j=10;
```

```
    m=i++;n=++j;
```

```
    printf("%d,%d,%d,%d\n",i,j,m,n);
```

```
}
```

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

```
main(){
```

```
    int i,j,m,n;
```



```

        i=8;j=10;
        printf("%d,%d,%d,%d\n",i,j,i++,j++);
    }
#include<stdio.h>
main(){
    int i,j,m,n;
    i=8;j=10;
    printf("%d,%d,%d,%d\n",i,j,++i,++j);
}
#include<stdio.h>
main(){
    int i,j,m=0,n=0;
    i=8;j=10;
    m+=i++;n--j;
    printf("i=%d,j=%d,m=%d,n=%d\n",i,j,m,n);
}

```

#### 调试分析

1~4: 程序出现这个结果的原因我分析是 因为系统函数的求值顺序是从右向左的，所以会先计算右边的第一个表达式，而我们知道，当自增运算符放在后面的时候，会先参与计算然后再加减，所以输出 n 的时候 j 还没有参与计算，所以输出 10，然后 j 就变成 11 了。而对于 i，由于前置，所以先加一，所以我们可以发现 m 和 i 的输出都是一样的。

5: 最后这个程序，先运行最后一个表达式，但是自减前置，所以 j 会先变成 9，然后 n 做减法，就得到解果了，然后 i 会先参与减法运算，给出 8，但是表达式计算完，也就同时给出减法的结果了，所以后面的 ij 都是这两个表达式计算完以后的结果。

#### 运行结果

9, 11, 9, 10

9, 11, 8, 11

9, 11, 8, 10

9, 11, 9, 11

i=9, j=9, m=8, n=-9

#### 实验总结

做实验的时候一定要注意好这种小地方，养成良好的编程习惯，不然就会导致以后经常出错，而且往往找不到为什么。

#### 习题三

##### 实验预习分析结果：

10

2/3

2.75



3.5

0.6

1

实验代码

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

```
main(){
```

```
    int a=2,b=7,c=2,d=3,f=3,g=4,h=4,i=3,p;
```

```
    double x=2.5,y=4.7,j=3.5,e=2.5,k,l,m,n,o;
```

```
    k=3.5+1/2+56%10;
```

```
    l=(a++*1/3);
```

```
    m=x+b%3*(int)(x+y)%2/4;
```

```
    n=(float)(c+d)/2+(int)j%(int)e;
```

```
    o=f=(f++g,f+5,f/5);
```

```
    p=(h>=i>=2)?1:0;
```

```
    printf("%lf %lf %lf %lf %lf %d",k,l,m,n,o,p);
```

```
}
```

实验结果

9.500000 0.000000 2.500000 3.500000 1.000000 0

调试分析

我换了一下数据类型和输出结果，除了小数位以外，几乎都是一样的。等号是从右向左的，所以加了等号并不会影响结果。那么为什么是这些输出呢？

第一个 我不知道/是取整的意思，所以这个应该是零，所以会有 9.5

第二个 表达式是从左向右，但是后置，所以是 2，而 2 除三取整，所以是零

第三个 也是因为不知道取整的意思

第四个 我对了

第五个 不知道取整

第六个 这个我想了很久，我真的不知道是为什么，我觉得应该是一

实验总结：所以还是最好要了解一些小的注意点，不然会有大麻烦

习题四

实验预习：

0

1

0

1

实验代码：

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

```
main(){
```

```
    int a=3,b=4,c=5,e,f,g,h,x,y;
```

```
    e=b>c&&b==c;
```

```
    f=!(a>b)&&!c||1;
```

```
    g=!(x=a)&&(y=b)&&0;
```

```
    h=!(a+b)+c-1&&b+c/2;
```

```
printf("%d%d%d%d",e,f,g,h);
}
```

实验结果：

**0101**

调试分析：和我的结果一样。抓住优先级的分析原则，然后非零的数取！会是零，！0=1，然后&&只区分零和非零两种即可。

实验总结：无

## 习题五

### 实验预习

24

60

0

124

### 实验代码

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

```
main(){
    int a=12;
    a+=a;
    printf("%d ",a);
    a=12;
    a*=2+3;
    printf("%d ",a);
    a=12;
    a/=a+a;
    printf("%d ",a);
    a=12;
    a+=a-=a*=a;
    printf("%d ",a);
}
```

### 调试分析

我的最后一个结果和实验的不一样，最下面的式子是因为右边的乘法先赋值了，然后又自己减自己，就是零，后来自己加自己，也还是零。

### 实验结果

**24 60 0 0**

### 实验总结

小心小心小心，不是优先级就是运算符的简化，这些都容易出错。

## 第六题

### 实验程序

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

```

#define PI 3.1415926535
main(){
    double r,V;
    scanf("%lf",&r);
    printf("%f ",r);
    V=(4.0/3.0)*PI*r*r*r;
    printf("%f",V);
}
#include<stdio.h>
main(){
    double r1,r2,r;
    scanf("%lf%lf",&r1,&r2);
    r=1.0/(1.0/r1+1.0/r2);
    printf("%f",r);
}
#include<stdio.h>
#include <math.h>
main(){
    double x=(3.31*pow(10,18)+2.10*pow(10,-7))/(7.16*pow(10,5)+2.01*pow(10,3));
    printf("%f",x);
}
#include<stdio.h>
main(){
    double a,b,c,d,e,y;
    scanf("%lf%lf%lf%lf%lf",&a,&b,&c,&d,&e);
    y=(a*b)/(c+d/e);
    printf("%f",y);
}

```

实验结果

```

2.0
2.000000 33.510322

```

```

2.0 2.0
1.000000

```

```

4609963649531.343700

```

```

1.0 2.0 3.0 4.0 5.0

```

调试分析：不会提示的错误又多了一个就是 scanf 输入的时候后面的地址没加&也可以编译通过就是没有结果。

实验总结：无

## 第七题

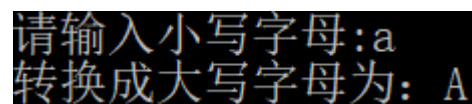
### 实验代码

```
#include <stdio.h>
main()
{
    char a;
    printf("请输入小写字母:");
    scanf("%c",&a);
    printf("转换成大写字母为: %c\n",a-32);
}
```

### 调试分析

大写字母的 ASCII 码就比小写字母的小 32，而且字符在计算机里的存储也是这个码，所以只要减掉 32 然后换一种输出方式就可以得到小写字母了。

### 实验结果



### 实验总结

无

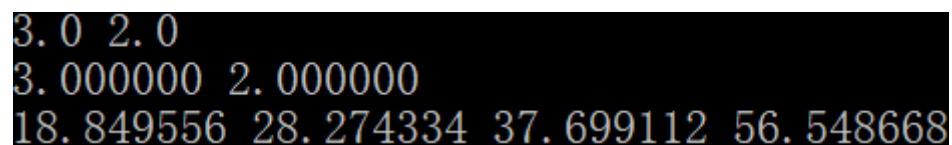
## 第八题

### 实验代码

```
#include<stdio.h>
#define PI 3.1415926535
main(){
    double r,h,C,S,L,V;
    scanf("%lf%lf",&r,&h);
    printf("%f %f\n",r,h);
    C=2*PI*r;
    S=PI*r*r;
    L=2*PI*r*h;
    V=PI*r*r*h;
    printf("%f %f %f %f",C,S,L,V);
}
```

调试分析：输出格式的时候注意空行和空格

### 实验结果



实验总结：无