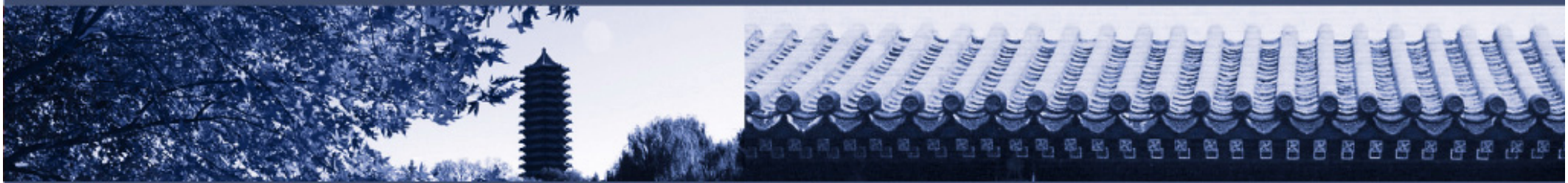
2018年春

程序设计实习(I): C++程序设计

第一讲 C语言补充知识

刘家瑛
liujiaying@pku.edu.cn



课前多吼歪

- 关于基础
- 关于上机
  - 第三周开始
  - 周六晚, 可调整
- 关于POJ
  - 会有新的分组, 具体要求等教学网助教通知
  - noi.openjudge.cn (部分题目分类, 可进行练习)
- 关于作业
  - 两周后提交, 具体布置及要求见教学网

C语言知识巩固和补充

- [回顾] 命令行参数
- 输入输出语句
- 位运算
- 函数指针

C语言知识巩固和补充

- [回顾] 命令行参数
- 输入输出语句
- 位运算
- 函数指针

命令行参数

- 命令行方式启动程序时，

  程序名称和其后那些字符串，统称为**命令行参数**

- 命令行参数可以有一个或多个，以空格分隔

- 例：在DOS窗口输入，

  copy file1.txt file2.txt

  - copy，file1.txt，file2.txt 就是命令行参数

如何在程序中获得命令行参数呢？

命令行参数

```
int main(int argc,char * argv[]) { … }
```

- 参数**argc** -- 启动程序时, 命令行参数的个数

  C/C++语言规定, 可执行程序程序本身的文件名,

  也算一个命令行参数 → argc的值至少是1

- 参数**argv** -- 数组, 其中每个元素都是一个**char* 类型的指针**

  该指针指向一个字符串, 这个字符串里就存放着命令行参数

  - argv[0]指向的字符串就是第一个命令行参数, 即可执行程序的文件名
  - argv[1]指向第二个命令行参数
  - argv[2]指向第三个命令行参数 ……

小练习

能处理命令行参数的C程序的main函数中，第二个参数的类型是：

A) char **

B) char

C) char *

小练习

能处理命令行参数的C程序的main函数中，第二个参数的类型是：

A) char **

B) char

C) char *

答案：A

C语言知识巩固和补充

- [回顾] 命令行参数
- 输入输出语句
- 位运算
- 函数指针

C语言的输入输出语句

- 需要#include <cstdio>
  - scanf() 将输入读入变量
  - printf() 将变量内容输出

scanf() 语句(函数)

```
int scanf(const char *fmt[,address,...]);
```

- 参数可变的函数
- 第一个参数是格式字符串
- 后面的参数是变量的地址
- 函数作用:

按照第一个参数指定的格式,将数据读入后面的变量

* 参数可变的函数的参考阅读(不要求掌握)
http://qiujiejia2008.blog.163.com/blog/static/795053062008219392466l/

scanf 返回值

- **>0** 成功读入的数据项个数
- **0** 没有项被赋值
- **EOF** 第一个尝试输入的字符是EOF (结束)

Note:

对于POJ上某些题, 返回值为EOF → 判断输入数据已经全部读完

* 关于EOF原理的参考阅读 (不要求掌握)

http://www.ruanyifeng.com/blog/2011/11/eof.html

POJ判断读入结束

```
while(scanf("%c", &c)!= EOF) {
    printf("%c", c);
}

while(std::cin >> n) {
    std::cout << n << std::endl;
}
```


printf() 语句(函数)

```
int printf(const char *fmt[,argument,...]);
```

- 参数可变的函数
- 第一个参数是格式字符串
- 后面的参数是待输出的变量
- 函数作用:

  按照第一个参数指定的格式,

  将后面的变量在屏幕上输出

printf 返回值

- 成功打印的字符数;
- **<0** 输出出错;

格式字符串里的格式控制符号

- **%d** 读入或输出**int**变量
- **%c** 读入或输出**char**变量
- **%f** 读入或输出**float**变量
- **%s** 读入或输出**char *** 变量
- **%lf** 读入或输出**double** 变量
- **%e** 以**科学计数法**格式输出数值
- **%x** 以**十六进制**读入或输出 **int** 变量
- **%u** 读入或输出**unsigned int**变量
- **%I64d** Windows下读入或输出 **_int64**变量 (64位整数)
- **%lld** Linux下读入或输出**long long**变量
- **%p** 输出**指针地址值**
- **%.5lf** 输出double变量, 精确到小数点后5位

```
#include <cstdio>
using namespace std;
int main() {
        int a;
        char b;
        char c[20];
        double d = 0.0;
        float e = 0.0;
        int n = scanf("%d%c%s%lf%f", &a, &b, c, &d, &e);
        printf("%d %c %s %lf %e %f %d", a, b, c, d, e, e, n);
        return 0;
}
```


```
int n = scanf("%d%c%s%lf%f", &a, &b, c, &d, &e);
printf("%d %c %s %lf %e %f %d", a, b, c, d, e, e, n);
```

input:

123a teststring 8.9 9.2

output:

123 a teststring 8.900000 9.200000e+000 9.200000 5

input:

123ateststring 8.9 9.2

output:

123 a teststring 8.900000 9.200000e+000 9.200000 5

input:

123 a teststring 8.9 9.2

%c会读取缓冲区中的换行或空格

output:

123 a 0.000000 0.000000e+000 0.000000 3

```
#include <cstdio>
using namespace std;
int main() {
        int a, b;
        char c;
        char s[20];
        long long n = 98765432100011111ll;
        scanf("%d %c,%s%x%I64d", &a, &c, s, &b, &n);
        printf("%d %x %u %s %p %x %d %I64d",
                a, a, a, s, s, b, b, n);
        return 0;
}
```

scanf函数格式中，
如有非控制符也非空格字符，
且输入数据中相应位置也出现该字符
→ 该字符会被跳过

input:

-28 K, test ffee 1234567890123456

output:

-28 fffffe4 4294967268 test 0012FF60 ffee 65518 1234567890123456

常见错误

```
#include <cstdio>
using namespace std;
int main() {
  char * s;
  scanf( "%s", s);
}
```

- 错在何处?

  s没有初始化

  不知道指向何处

  往其指向的地方写入数据, 不安全!

读取一行

```
char * gets(char * s);
```

- 从标准输入读取一行到字符串s
- 如果成功, 返回值就是 s 地址
- 如果失败, 返回值是 NULL
- 可以根据返回值是 NULL判定输入数据已经读完

Note: 调用时要确保 s 指向的缓冲区足够大,
否则可能发生内存访问错误

读取一行

```
#include <cstdio>
using namespace std;
int main()  {
  char s[200];
  char * p = gets(s);
  printf("%s:%s", s, p);
  return 0;
}
```

input:

Welcome to Beijing!

output:

Welcome to Beijing!:Welcome to Beijing!

sscanf 函数和 sprintf函数

```
int sscanf(const char * buffer, const
char * format[, address, ...]);
```

和scanf的区别在于: 它是从buffer里读取数据

```
int sprintf(char *buffer, const char
*format[, argument, ...]);
```

和printf的区别在于: 它是往buffer里输出数据

```
#include <cstdio>
using namespace std;
int main() {
        int a, b; char c;
        char s[20];
        char szSrc[] = "-28 K, test ffee 1234567890123456";
        char szDest[200];
        long long n = 9876543210001111;
        sscanf(szSrc, "%d %c,%s%x%lld", &a, &c, s, &b, &n);
        sprintf(szDest, "%d %x %u %s %p %x %d %lld",
                        a, a, a, s, s, b, b, n);
        printf("%s", szDest);
        return 0;
}
```

output:

-28 fffffe4 4294967268 test 0012FF60 ffee 65518 1234567890123456

C语言知识巩固和补充

- [回顾] 命令行参数
- 输入输出语句
- 位运算
- 函数指针

负整数的表示方式

- 解决方案之一：设置"符号位"
- 最左边一位(最高位)作为符号位: 表示整数的正负
  - 符号位为**0**, 说明该整数是**非负的**
  - 符号位为**1**, 说明该整数是**负的**
- 除符号位外的其余位,
  - 非负整数 = 绝对值
  - 负整数 = 绝对值取反再加1

(取反就是把0变成1, 把1变成0)

- 为简单起见，下表以16位的计算机为例，列出了几个整数及其在计算机中的表示形式：

整数	16位二进制表示形式	十六进制表示形式
0	0000 0000 0000 0000	0000
1	0000 0000 0000 0001	0001
257	0000 0001 0000 0001	0101
32767	0111 1111 1111 1111	7FFF
-32768	1000 0000 0000 0000	8000
-1	1111 1111 1111 1111	FFFF
-2	1111 1111 1111 1110	FFFE
-257	1111 1110 1111 1111	FEFF

- 以 **-1** 为例来说明负数的表示方法
- -1的符号位为1, 绝对值的二进制表示形式为:

  000 0000 0000 0001
- 取反后得到:

  111 1111 1111 1110, 加1后变成

  111 1111 1111 1111, 再补上最高位的符号位,

  最终得到其二进制表示形式为:

  1111 1111 1111 1111

由负整数的二进制表示形式算出其绝对值的方法, 就是将所有位取反, 然后再加1

位运算

- 对某个整数类型变量中的某一位 (bit) 进行操作
  - 判断某一位是否为1
  - 只改变其中某一位, 而保持其他位都不变
- C/C++语言提供了六种位运算符来进行位运算操作:

&	按位与
|	按位或
^	按位异或
~	取反
<<	左移
>>	右移

按位与

- 按位与运算符 "&"
  - 双目运算符
  - 功能

    将参与运算的两操作数各对应的二进制位

    进行与操作
  - 只有对应的**两个二进位均为1**时→ 结果的对应二进制位才为1, 否则为0

按位与

[例]: 表达式 "21 & 18" 的计算结果是16

(即二进制数10000), 因为:

21 用二进制表示就是:

0000 0000 0000 0000 0000 0000 0001 0101

18 用二进制表示就是:

0000 0000 0000 0000 0000 0000 0001 0010

二者按位与所得结果是:

0000 0000 0000 0000 0000 0000 0001 0000

按位与

- 按位与运算 "&"

  将某变量中的某些位清0或保留某些位不变

- 如果需要将int变量n的低8位全置成0, 而其余位不变, 则可以执行:

  ```
  n = n & 0xffffff00;
  ```

  也可以写成:

  ```
  n &= 0xffffff00;
  ```

  如果n是short类型的，则只需执行:

  ```
  n &= 0xff00;
  ```


按位与

[例]: 如何判断一个int变量n的第7位

(从右往左, 从0开始数) 是否是1?

- 只需看表达式 "n & 0x80" 的值是否等于0x80即可

按位或

- 按位或运算符 "|"
  - 双目运算符
  - 功能

    将参与运算的两操作数各对应的二进制位

    进行或操作
  - 只有对应的**两个二进位都为0**时→结果的对应二进制位才是0, 否则为1
- 按位或运算

  将某变量中的某些位置1或保留某些位不变

按位或

[例]: 表达式 "21 | 18" 的值是23

21： 0000 0000 0000 0000 0000 0000 0001 0101

18： 0000 0000 0000 0000 0000 0000 0001 0010

21|18: 0000 0000 0000 0000 0000 0000 0001 0111

- 例, 如果需要将int型变量n的低8位全置成1, 而其余位不变，则可以执行：

```
n |= 0xff;
```


按位异或

- 按位异或运算符 "^"
  - 双目运算符
  - 功能: 将参与运算的两操作数各对应的二进制位进行**异或**操作
  - 只有对应的两个二进位**不相同**时, 结果的对应二进制位是**1**, 否则为0
- 例, 表达式 "21 ^ 18" 的值是7 (即二进制数111)

  21： 0000 0000 0000 0000 0000 0000 0001 0101

  18： 0000 0000 0000 0000 0000 0000 0001 0010

  21^18: 0000 0000 0000 0000 0000 0000 0000 0111
- 异或运算的特点是:

  如果 a^b=c, 那么就有 c^b = a以及c^a=b

  此规律可以用来进行最简单的加密和解密

按位非

- 按位非运算符 "~"
  - 单目运算符
  - 功能是将操作数中的二进制位0变成1，1变成0
- 例, 表达式 "~21" 的值是无符号整型数 0xffffffea

  21:　0000 0000 0000 0000 0000 0000 0001 0101

  ~21:　1111 1111 1111 1111 1111 1111 1110 1010
- 下面的语句

```
printf("%d, %u, %x", ~21, ~21, ~21);
```

- 输出结果就是:

  -22, 4294967274, ffffffea

左移运算符

- 左移运算符 "<<"
  - 双目运算符
  - 将左操作数的各二进位全部左移若干位后得到的值
  - 右操作数指明了要左移的位数
  - 左移时, 高位丢弃, 低位补0
  - 左移运算符不会改变左操作数的值

左移运算符

- 例, 常数9有32位, 其二进制表示是:

  0000 0000 0000 0000 0000 0000 0000 1001

- 因此, 表达式 "9<<4" 的值, 就是将上面的二进制数左移4位, 得:

  0000 0000 0000 0000 0000 0000 1001 0000

  即为十进制的144

  - 左移1位 → 等于是乘以2
  - 左移n位 → 等于是乘以2^n
  - 左移操作比乘法操作快得多

```
#include <iostream>
using namespace std;
int main() {
    int n1 = 15;
    short n2 = 15;
    unsigned short n3 = 15;
    unsigned char c = 15;
    n1 <<= 15;
    n2 <<= 15;
    n3 <<= 15;
    c <<= 6;
    printf( "n1=%x, n2=%d, n3=%d, c=%x, c<<4=%d",
            n1, n2, n3, c, c << 4);
}
```

上面程序的输出结果是:

n1=78000, n2=-32768, n3=32768, c=c0, c<<4=3072

n1: 0000 0000 0000 0000 0000 0000 0000 1111
n2: 0000 0000 0000 1111
n3: 0000 0000 0000 1111
c: 0000 1111

n1 <<= 15 (变成78000)

0000 0000 0000 0111 1000 0000 0000 0000

n2 <<= 15 (变成-32768)

1000 0000 0000 0000

n3 <<= 15 (变成 32768)

1000 0000 0000 0000

c <<= 6 (变成 c0)

1100 0000

c << 4 这个表达式是先将 c 转换成整型

0000 0000 0000 0000 0000 0000 1100 0000

然后再左移, c<<4=3072

右移运算符

- 右移运算符 ">>"
  - 双目运算符
  - 把 ">>" 的左操作数的各二进位全部右移若干位后得到的值
  - 要移动的位数就是 ">>" 的右操作数
  - 移出最右边的位就被丢弃
- 对于有符号数, 如long/int/short/char类型变量
  在右移时, 符号位 (即最高位) 将一起移动
- 大多数C/C++编译器规定,
  如果原符号位为1, 则右移时高位就补充1
  如果原符号位为0, 则右移时高位就补充0

右移运算符

- 对于无符号数, 如unsigned long/int/short/char类型的变量, 则**右移**时, 高位总是**补0**
- 右移运算符不会改变左操作数的值
- 实际上, 右移n位, 就相当于左操作数除以2^n, 并且将结果**往小里取整**

-25 >> 4 = -2

-2 >> 4 = -1

18 >> 4 = 1

```
#include <iostream>
using namespace std;
int main() {
        int n1 = 15;
        short n2 = -15;
        unsigned short n3 = 0xffe0;
        unsigned char c = 15;
        n1 = n1>>2;
        n2 >>= 3;
        n3 >>= 4;
        c >>= 3;
        printf( "n1=%d, n2=%d, n3=%x, c=%x", n1, n2, n3, c);
}
```

上面的程序输出结果是:

n1=3, n2=-2, n3=ffe, c=1

n1: 0000 0000 0000 0000 0000 0000 0000 1111
n2: 1111 1111 1111 0001
n3: 1111 1111 1110 0000
c: 0000 1111

n1 >>= 2 变成3

0000 0000 0000 0000 0000 0000 0000 0011

n2 >>= 3 变成-2

1111 1111 1111 1110

n3 >>= 4 变成 ffe

0000 1111 1111 1110

c >>= 3 变成1

0000 0001

思考题：

有两个int型的变量a和n (0 <= n <= 31)，

要求写一个表达式，使该表达式的值和a的第n位相同

答案：

(a >> n) & 1

思考题:

有两个int型的变量a和n (0 <= n < 31),

要求写一个表达式，使该表达式的值和a的第n位相同

答案:

(a >> n) & 1

或:

(a & (1 << n)) >> n

C语言知识巩固和补充

- [回顾] 命令行参数
- 输入输出语句
- 位运算
- 函数指针

函数指针

- 程序运行期间，每个函数都会占用一段连续的内存空间
- 函数名就是该函数所占内存区域的起始地址

  (也称"入口地址")
- 将函数的**入口地址**赋给一个指针变量，使该指针变量指向该函数
- 通过指针变量就可以调用这个函数

这种指向函数的指针变量称为 **"函数指针"**

函数指针

- 函数指针定义的一般形式为：

  类型名 (* 指针变量名)(参数类型1, 参数类型2, ...);

  - 类型名 – 表示被指函数的返回值的类型
  - (参数类型1, 参数类型2, …) – 分别指函数的所有参数的类型
  - 例如：

    ```
    int (*pf)(int, char);
    ```

    - 表示pf是一个函数指针, 它所指向的函数
    - 返回值类型应是int, 该函数应有两个参数, 第一个是int 类型, 第二个是char类型

函数指针

- 用一个原型匹配的函数的名字给一个函数指针赋值
- 要通过函数指针调用它所指向的函数，写法为：

  函数指针名(实参表);

- 下面的程序说明了函数指针的用法

```
#include <stdio.h>
void PrintMin(int a, int b){
    if( a < b )
        printf("%d", a);
    else
        printf("%d", b);
}
int main(){
    void (* pf)(int, int);
    int x = 4, y = 5;
    pf = PrintMin;
    pf(x, y);
    return 0;
}
```

上面的程序输出结果是：4

函数指针应用：快速排序库函数qsort

```
void qsort(void *base,
      int nelem,
      unsigned int width,
      int (* pfCompare)(const void *, const void *));
```

- base是待排序数组的起始地址
- nelem是待排序数组的元素个数
- width是待排序数组的每个元素的大小(以字节为单位),

最后一个参数

- pfCompare是一个函数指针, 指向一个"比较函数"

快速排序库函数qsort

- 排序就是一个不断比较并交换位置的过程
- qsort如何在连元素的类型是什么都不知道的情况下，比较两个元素并判断哪个应该在前呢?

[答案]: qsort函数在执行期间, 会通过pfCompare指针调用一个 "比较函数"

→用以判断两个元素哪个更应该排在前面

- 这个 "比较函数" 不是C/C++的库函数,
  而是由使用qsort的程序员编写的
- 调用qsort时, 将 "比较函数" 的名字作为实参传递给pfCompare

快速排序库函数qsort

- qsort函数的用法规定,"比较函数" 的原型应是:
  int 函数名(const void * elem1, const void * elem2);
- 该函数的两个参数, elem1和elem2, 指向待比较的两个元素
- * elem1和 * elem2就是待比较的两个元素

该函数必须具有以下行为:

1) 如果 * elem1应该排在 * elem2 **前面**, 则函数返回值是**负整数**
(任何负整数都行)

2) 如果 * elem1和* elem2哪个排在前面都行, 那么函数**返回0**

3) 如果 * elem1应该排在 * elem2 **后面**, 则函数返回值是**正整数**
(任何正整数都行)

快速排序库函数qsort的实现

- 下面的程序, 功能是调用qsort库函数, 将一个unsigned int数组按照个位数从小到大进行排序
- 比如 8, 23, 15三个数, 按个位数从小到大排序, 就应该是 23, 15, 8

```
#include <stdio.h>
#include <stdlib.h>
int MyCompare(const void * elem1, const void * elem2 ){
  unsigned int * p1, * p2;
  p1 = (unsigned int *) elem1;
  p2 = (unsigned int *) elem2;
  return  (* p1 % 10)  - (* p2 % 10 );
}

#define NUM 5
int main(){
  unsigned int an[NUM] = { 8, 123, 11, 10, 4 };
  qsort(an, NUM, sizeof(unsigned int), MyCompare);
  for(int i = 0; i < NUM; i ++ )
      printf("%d ", an[i]);
        return 0;
}
```

上面程序的输出结果是:

10 11 123 4 8

思考题

- 如果要将an数组从大到小排序，那么MyCompare函数该如何编写？

C语言知识巩固和补充

- [回顾] 命令行参数
- 输入输出语句
- 位运算
- 函数指针
- [补] 动态内存分配
- [附录] C语言标准库函数

补：动态内存分配

- 在C++中，通过 **new运算符** 来实现**动态内存分配**
- **第一种用法：**

  P = new T;

  T是任意类型名，P是类型为T *的指针
- 动态分配出一片大小为 **sizeof(T)** 字节的内存空间，并将该内存空间的**起始地址**赋值给P
- 比如：

```
int * pn;
pn = new int;    //(1)
* pn = 5;
```

语句(1)，即动态分配了一片4个字节大小的内存空间

pn 会指向这片空间，通过pn,可以读写该内存空间

补：动态内存分配

- new 运算符的第二种用法

  用来动态分配一个任意大小的数组：

  P = new T[N];

  T是任意类型名，P是类型为T * 的指针

  N代表 "元素个数"，它可以是任何值为正整数的表达式，表达式里可以包含变量，函数调用

- 这样的语句动态分配出 N × sizeof(T)个字节的内存空间，这片空间的起始地址被赋值给P

- 例如：

```
int * pn;
int i = 5;
pn = new int[i * 20];
pn[0] = 20;
pn[100] = 30;   //编译没问题，运行时导致数组越界
```

补：动态内存分配

- 如果要求分配的空间太大，操作系统找不到足够的内存来满足，那么动态内存分配就会失败。保险做法是在进行较大的动态内存分配时，要判断一下分配是否成功
- 判断的方法是：

  如果new表达式返回值是NULL，则分配失败；否则分配成功
- 例如：

```
int * pn = new int[200000];
if( pn == NULL )
        printf( “内存分配失败” );
else
        printf( “内存分配成功” );
```


补：动态内存分配

- 程序从操作系统动态分配所得的内存空间，使用完后应该释放，交还操作系统，以便操作系统将这片内存空间分配给其他程序使用
- C++提供 **"delete" 运算符**，用以释放动态分配的内存空间
- delete运算符的基本用法是：

  delete 指针；
- 该指针必须是指向动态分配的内存空间的，否则运行时很可能会出错. 例如：

```
int * p = new int;
* p = 5;
delete p;
delete p;     //本句会导致程序异常
```

补：动态内存分配

- 如果是用new动态分配了一个数组，那么释放该数组的时候，应以如下形式使用 delete 运算符：

  delete [] 指针;

例如：

```
int * p = new int[20];
p[0] = 1;
delete [] p;
```


补：动态内存分配

Note: new 运算符动态分配的内存空间 → 一定要用

delete 运算符进行释放

- 否则即便程序运行结束, 这部分内存空间仍然有可能不会被操作系统收回 (取决于操作系统如何设计),
- 从而成为被白白浪费掉的内存垃圾

- 这种现象也称为 "内存泄漏"

附录：C语言标准库函数

数学函数

- 数学库函数声明在math.h中，主要有：

abs(x)	求整型数x的绝对值
cos(x)	x(弧度)的余弦
fabs(x)	求浮点数x的绝对值
ceil(x)	求不小于x的最小整数
floor(x)	求不大于x的最小整数
log(x)	求x的自然对数
log10(x)	求x的对数(底为10)
pow(x,y)	求x的y次方
sin(x)	求x(弧度)的正弦
sqrt(x)	求x的平方根

附录：C语言标准库函数

字符处理函数

在ctype.h中声明，主要有：

- int isdigit(int c)　　判断c是否是数字字符
- int isalpha(int c)　　判断c 是否是一个字母
- int isalnum(int c)　　判断c是否是一个数字或字母
- int islower(int c)　　判断 c 是否是一个小写字母
- int isupper(int c)　　判断 c 是否是一个大写字母
- int toupper(int c)　　如果 c 是一个小写字母，则返回其大写字母
- int tolower (int c)　　如果 c 是一个大写字母，则返回其小写字母

附录：C语言标准库函数

字符串和内存操作函数

字符串和内存操作函数声明在string.h中，常用的有：

- char * strchr(char * s, int c)
  如果s中包含字符c, 则返回一个指向s第一次出现的该字符的指针; 否则返回NULL
- char * strstr(char * s1, char * s2)
  如果s2是s1的一个子串, 则返回一个指向s1中首次出现s2的位置的指针; 否则返回NULL
- char * strlwr(char * s) 将s中的字母都变成小写
- char * strupr(char * s) 将s中的字母都变成大写
- char * strcpy(char * s1, char * s2)
  将字符串s2的内容拷贝到s1中去
- char * strncpy(char * s1, char * s2,int n)
  将字符串s2的内容拷贝到s1中去，但是最多拷贝n个字节。
  如果拷贝字节数达到n, 那么就不会往s1中写入结尾的’\0’

附录：C语言标准库函数

字符串和内存操作函数

- char * strcat(char * s1, char * s2) 将字符串s2添加到s2末尾
- int strcmp(char * s1, char * s2)
  比较两个字符串，大小写相关。如果返回值小于0，则说明s1按字典顺序在s2前面；返回值等于0，则说明两个字符串一样；返回值大于0，则说明s1按字典顺序在s2后面。
- int stricmp(char * s1, char * s2)
  比较两个字符串，大小写无关。其他和strcmp同。
- void * memcpy(void * s1, void * s2, int n)
  将内存地址s2处的n字节内容拷贝到内存地址s1。
- void * memset(void * s, int c, int n)
  将内存地址s开始的n个字节全部置为c。

附录：C语言标准库函数

字符串转换函数

- 将字符串转换为整数，或将整数转换成字符串等这类功能。它们定义在stdlib.h中：
- int atoi(char *s)
  将字符串s里的内容转换成一个整型数返回。比如，如果字符串s的内容是“1234”,那么函数返回值就是1234
- double atof(char *s)
  将字符串s中的内容转换成浮点数

附录：C语言标准库函数

字符串转换函数

- char *itoa(int value, char *string, int radix);
  将整型值value以radix进制表示法写入string
- 比如：

  char szValue[20];

  itoa(32,szValue,10) 则使得szValue的内容变为 “32”

  itoa(32,szValue,16) 则使得szValue的内容变为 “20”.