计算机程序设计基础(1)
--- C语言程序设计(9)

孙甲松
sunjiasong@tsinghua.edu.cn
电子工程系 信息认知与智能系统研究所
罗姆楼6-104
电话: 13901216180/62796193
2022.11.

期中机考时间安排与要求

第九周（11月7日-11日）五个时间段：

周二(11月8日)中午13:30开始准备,14:00正式开始,15:30结束。

周二(11月8日)下午16:00开始准备,16:30正式开始,18:00结束。

周二(11月8日)晚上18:30开始准备,19:00正式开始,20:30结束。

周三(11月9日)晚上18:30开始准备,19:00正式开始,20:30结束。

周五(11月11日)晚上18:30开始准备,19:00正式开始,20:30结束。

考试时间1个半小时，包括：1小时正式考试，30分钟延长。提前30分钟准备。

请按照二级选课时间去机房考试。因机房座位容量有限，未按照二级选课时间考试的同学可能会在超容时被疏导到其他时间段。

每位同学必须在五个时间段中选一个参加上机考试，不参加的将没有期中考试成绩。每位同学只允许进行一次上机考试。如有同学进行多次考试,核实后将按照所有考试成绩的最低分进行登记。

考试范围：截止到第7章循环，主要考循环与分支。

期中机考时间安排与要求

评分规则：

在1小时正式考试时间内完成并一次性通过验收者得满分10分。

在1小时到1小时15分钟内完成者，扣1分。

在1小时15分钟到1小时30分钟内完成者，扣2分。

1小时30分钟未完成者酌情评分。

此外,不论正式考试时间或延长时间,每次验收失败额外扣1分。

考试纪律：

考试设备为中央主楼机房电脑，不允许使用自带计算机进行考试。考试时，每个人自己独立完成考题的编程和调试。禁止连入Internet，禁止使用手机、U盘等设备。可以使用教材等参考书籍。请各位同学严格遵守清华大学教育教学纪律，一旦发现作弊等不端行为，将按照清华大学校规严肃处理。

线上机考：

未返校同学参加线上机考，请根据助教在网络学堂发的通知做好准备。

第9章 数组

9.1 数组的基本概念

- 数组（Array）是相同数据类型元素的集合，用统一的数组名来表示, 数组中的每一个元素通过下标来区分。

数组元素又称为下标变量, 而以前所讲的变量称为简单变量

【例9-1】 改用数组来解决4.3节的例4-1, 确定谁是小偷 。a[0]、a[1]、a[2]、a[3] 分别表示A、B、C、D这4个人。通过分析每一个人的回答,得到确定谁是小偷的条件为:

```
a[1]+a[3]==1 &&  a[1]+a[2]==1 &&  a[0]+a[1]==1
&&  a[0]+a[1]+a[2]+a[3]==1
```

逻辑表达式可以简化为 :

```
a[1]+a[3]==1 && a[1]+a[2]==1 && a[0]+a[1]==1
```

最后写出C程序如下：

```
#include  <stdio.h>
main( )
{  int k, j, a[4];
   for (k=0; k<=3; k++)
   {   for (j=0; j<=3; j++)
       {  if (j==k) a[j]=1;     /* 假设a[k]为小偷 */
          else a[j]=0;          /* 其余3人都不是小偷 */
       }
       if (a[1]+a[3]==1 && a[1]+a[2]==1 && a[0]+a[1]==1)
          printf("The thief  is %c.\n",  k+'A');
   }
}
```

程序运行结果为：

The thief is B. （B是小偷）

用数组表示的方法对于小偷人数n的值很大时尤其重要，而且具有通用性，程序结构不随n而改变。无法想象如何写一个成百上千重循环的程序。

9.2 数组的定义与引用

9.2.1 一维数组

一般形式

类型说明符 数组名[常量表达式];

其中类型说明符是定义数组中各元素的数据类型,

常量表达式是说明数组的大小（即数组中元素的个数）

提醒

- 数组名的命名规则与变量名相同。
- 说明数组大小的常量表达式必须为整型，并且要用方括号括起来(不能用圆括号)。
- 说明数组大小的常量表达式中可以包含符号常量,但不能是变量。

int n; scanf("%d", &n); int a[n]; 是错误的!

因为n是变量, 不能用来定义数组。

- 在C语言中,数组元素的下标总是从0开始
- 若定义数组长度为N，数组元素的下标只能到N-1
- 在C语言中,只能逐个引用数组元素,不能一次引用数组中的全部元素

【例9-2】 下面的程序说明了如何对数组定义和引用数组元素：

只能逐个给数组元素赋值,也只能逐个打印数组元素,而不能试图通过：

```
scanf("%d", a);
printf("%d", a);
```

整体一起输入输出。

```
#include   <stdio.h>
#define  N  5
main( )
{  int  i, a[N];
   for (i=0; i<N; i++)
       a[i]=i;
   for (i=0; i<N; i++)
       printf("%5d",a[i]);
   printf("\n");
}
```

9.2.2 二维数组

一般形式

类型说明符 数组名[常量表达式1][常量表达式2];

例如,说明语句 **double a[3][4],b[5][10];**

定义了两个二维数组：3行4列的双精度实型数组a, 共有12个元素；5行10列的双精度实型数组b, 共有50个元素。

注意：在C语言中,二维数组在计算机中的存储顺序是以行为主的,即第一维的下标变化慢,第二维的下标变化快。

例如,由说明语句

double a[3][4];

数组在计算机内存中逐行存储的顺序如下：

a[0][0]⟶a[0][1]⟶a[0][2]⟶a[0][3]⟶

a[1][0]⟶a[1][1]⟶a[1][2]⟶a[1][3]⟶

a[2][0]⟶a[2][1]⟶a[2][2]⟶a[2][3]

9.2.3 数组的初始化

1. 一维数组的初始化

方法

① 利用赋值语句逐个对数组中的元素进行赋值。

② 利用输入函数为数组中的各个元素逐个输入值。

③ 初始化,即在定义数组时直接为各个元素赋初值。

说明

① 可以只给数组的前若干个元素赋初值, 此时后面的元素均将自动赋初值0。

需要指出的是,外部和静态数组在对程序进行编译连接的时候就给予分配存储空间,并自动进行赋初值(0或者你给的初值)。

而函数中定义的局部数组是在运行时才分配存储空间,如果不赋初值，其中各元素的初值是随机的。

例如：

```
#include <stdio.h>
void main( )
{    int a[10]={1,2,3,4,5,6,7,8,9,10}, i;
     for (i=0; i<10; i++)
          printf("%d, ", a[i]);
}
```

运行结果： 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,按任意键继续…

程序改为：

```
#include <stdio.h>
void main( )
{    int a[10]={1,2,3,4,5},  i;     /* 后5个元素自动赋初值为0 */
     for (i=0; i<10; i++)
          printf("%d, ", a[i]);
}
```

运行结果： 1, 2, 3, 4, 5, 0, 0, 0, 0, 0,按任意键继续…

说明

② 在对全部元素赋初值时,说明语句中可以不指定数组长度，其长度默认为与初值表中数据的个数相同。

如果不是对全部元素赋初值，则在说明语句中必须说明数组的长度。

【例9-3】分析下列程序的输出结果：

```
#include  <stdio.h>
main( )
{ int k,x[5];
  static int y[5];
  int z[5]={0};
  for (k=0; k<5; k++)
     printf("%5d%5d%5d\n",x[k],y[k],z[k]);
}
```

局部数组

用static说明的局部静态数组

局部数组,但在该说明语句中有为数组赋初值的部分

程序的运行结果为：

```
-858993460    0    0
-858993460    0    0
-858993460    0    0
-858993460    0    0
-858993460    0    0
```

如果程序改为：

```
#include  <stdio.h>
int k, x[5]; /* 外部数组变量，会自动初始化为0 */
main( )
{  static int y[5];  /*用static说明的局部静态数组会自动初始化为0 */
   int z[ ]={0,0,0,0,0};  /* 定义不写长度,由初值个数确定数组长度 */
   for (k=0; k<5; k++)
       printf("%5d%5d%5d\n", x[k], y[k], z[k]);
}
```

程序的运行结果为：

```
    0    0    0
    0    0    0
    0    0    0
    0    0    0
    0    0    0
```

【例9-4】从键盘输入年、月、日,计算并输出该日是该年的第几天。

```
#include   <stdio.h>
main( )
{ int year,month,day,k,sum;
   int t[ ]={31,28,31,30,31,30,31,31,30,31,30,31};
   printf("input year,month,day:");
   scanf("%d%d%d", &year, &month, &day);
   if ((year%4==0 && year%100!=0)||year%400==0)
       t[1]++;   /* 如果闰年,二月加1天 */
   sum=day;
   for (k=0; k<month-1; k++)
       sum = sum + t[k];
   printf("Days=%d\n", sum);
}
```

思考：此题能否修改为：输入任何的年月日,告知是星期几?

2．二维数组的初始化

注意：

① 在分行给二维数组赋初值时，对于每一行都可以只对前几个元素赋初值，后面未赋初值的元素系统将自动赋初值0，并且，还可以只对前几行元素赋初值。

② 在给全部元素赋初值时，说明语句中可以省略第一维的长度说明（但一对方括号不能省略）。

③ 在分行赋初值时，也可以省略第一维的长度说明。

```
int a[3][4] ={1,2,3,4,5,6,7,8,9,10,11,12};
int a[3][4] ={{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
int a[3][4] ={{1,2}, {5}, {9,10,11}};
int a[ ][4] ={1,2,3,4,5,6,7,8,9,10,11,12,13};
                    /* 等价于 int a[4][4]，最后3个元素值为0 */
int a[ ][4] ={{1,2}, {5}, {9,10,11}};
                    /*等价于 int a[3][4]，缺省者值为0 */
```

【例9-5】 求两个矩阵的乘积矩阵$C_{mn}= A_{mk}B_{kn}$

```
#include   <stdio.h>
main( )
{  int  i,j,k,c[2][3];
   int  a[2][4]={1,2,3,4,5,6,7,8};
   int  b[4][3]={1,2,3,4,5,6,7,8,9,10,11,12};
   for (i=0; i<2; i++)      /*矩阵相乘*/
     for (j=0; j<3; j++)
     {     c[i][j]=0;  /* 赋初值0，准备累加 */
           for (k=0; k<4; k++)
                c[i][j] +=  a[i][k]*b[k][j];
     }
   for (i=0; i<2; i++)       /*输出乘积矩阵*/
   {   for (j=0; j<3; j++)
             printf("%6d", c[i][j]);
       printf("\n");
   }
}
```

运行结果：

```
    70    80    90
   158   184   210
```

9.3 字符数组与字符串

- 用于存放字符型数据的数组称为字符数组。
- 在C语言中,字符数组中的一个元素只能存放一个字符。

9.3.1 字符数组的定义与初始化

- 一般形式

```
char  数组名[常量表达式];                    一维字符数组
char  数组名[常量表达式1][常量表达式2];   二维字符数组
```

例如： **char c[14];**

c[0]='G'; c[1]='o'; c[2]='o'; c[3]='d'; c[4]='b'; c[5]='y'; c[6]='e';

c[7]='!'; c[8]='\0';

定义了具有14个字符元素的字符数组c。字符数组元素经上述赋值语句赋值后，在计算机内存中的存放形式如下（其中后5个数组元素未赋值为随机字符）：

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]
G	o	o	d	b	y	e	!	\0

初始化

(1) 当对字符数组中所有元素赋初值时,数组的长度说明可以省略。

```
char a[14 ]={'h', 'o', 'w', ' ', 'd', 'o', ' ', 'y', 'o', 'u', ' ', 'd', 'o', '?', '\0'};
char a[ ]={'h', 'o', 'w', ' ', 'd', 'o', ' ', 'y', 'o', 'u', ' ', 'd', 'o', '?', '\0'};
```

(2) 可以只对前若干元素赋初值。

例如：　char b[10]={'A', 'B', ' ', 'C', 'D'};

此时后5个元素自动为'\0'（ASCII码值为0的字符）。

下列两个外部说明语句等价，结果所有元素全为'\0'：

```
unsigned char b[10]={'\0'};
unsigned char b[10]; /* 外部字符数组的全部元素自动赋初值为0 */
```

下列两个局部说明语句的效果是不同的：

static char a[10];

char a[10];

其中：

第一个局部说明语句不仅定义了一个长度为**10**的字符型静态数组**a**，而且还同时给其中的每一个元素（**a[0]～a[9]**）都赋了**'\0'**。

第二个局部说明语句只定义了一个长度为**10**的字符型数组**a**，在没有给其中的元素赋值以前其初值是随机的（不一定是**0**）。

9.3.2 字符串

C语言规定

- 字符串常量（简称字符串）要用一对双撇号括起来

例如, "how do you do?"是一个长度为14的字符串常量。但必须注意, 在一个字符串常量中,最后还隐含包括一个结束符'\0', 即在上面这个字符串常量中,实际包含15个字符,最后一个为结束符'\0'。

- C语言允许用字符串常量对字符数组进行初始化

例如,下列四个语句是等价的：

```
char a[15]={"how do you do?"};
char a[15]="how do you do?";
char a[ ]="how do you do?";
char a[ ]={'h', 'o', 'w', ' ', 'd', 'o', ' ', 'y', 'o', 'u', ' ', 'd', 'o', '?', '\0'};
```

而 `char a[ ]={104,111,119,32,100,111,32,121,111,117,32,100,111,63, 0};`

与上面四个语句也是等价的！

下列三个语句等价(后10个字符都为'\0')：

```
char  b[15]= "China";
char  b[15]={ 'C', 'h', 'i', 'n', 'a', '\0'};
char  b[15]={ 'C', 'h', 'i', 'n', 'a'};
```

下列两个语句不等价：

```
char  b[15]="China";        数组b长度为15
char  b[ ]="China";         数组b长度为6
```

- 字符串的长度与字符数组的长度是不相同的。

例如，说明语句

```
char  b[15]="China";
```

sizeof(b)与strlen(b)是不同的，strlen( )函数求字符串长度(不包括结束符\0)。字符数组b的长度为15，该数组内存放的字符串"China"的长度为5，但该字符串中实际有6个字符。即通常所说的字符串的长度不包括字符串结束符'\0'。

- 特别需要指出的是，利用字符串常量可以对字符数组进行初始化，但不能用字符串常量为字符数组赋值。

例如：下列初始化语句是允许的：

```
char  b[15]="China";
```

但下面的写法都是错误的：

```
char  b[15];
b= "China";
```

```
char  b[15];
b[15]= "China";
```

9.3.3 字符数组与字符串的输入与输出

字符数组的输入与输出

- 可以对数组中的每一个字符元素逐个进行输入或输出，也可以将数组中的所有字符作为一个字符串进行输入或输出。

格式说明符

(1) 格式符 %c用于输入输出一个字符

(2) 格式符 %s用于输入输出一个字符串

1. 输入输出一个字符（格式说明符为%c）

在用于输入时,输入项为数组元素地址。在具体输入时,各字符之间不需要分隔符分隔,字符也不要用单撇号括起来。在用于输出时,输出项为数组元素。

【例9-6】在下列C程序中,首先分别为字符数组元素a[1]与a[2]读入字符,然后输出数组元素a[2]中的字符。

```
#include   <stdio.h>
main( )
{  char  a[5];
   scanf("%c%c", &a[1], &a[2]);
   a[0]='a'; a[3]='d'; a[4]='\0';
   printf("%c\n",a[2]);
}
```

在运行上述程序时,如果从键盘输入　bc

则输出结果为:　　c

2. 输入输出一个字符串（格式说明符为 %s）

在用格式说明符 %s进行输入输出时,其输入输出项均为数组名。但在输入时，相邻两个字符串之间要用空格(回车)分隔，系统将自动地在所读入的字符串最后加结束符'\0'。在输出时， 遇结束符'\0'作为输出结束标志。

例9-6输出若改为： printf("%s\n",a); 输出结果将是：abcd

【例9-7】下面的C程序是对数组进行输入与输出操作：

```
#include   <stdio.h>
main( )
{ char  a[6],b[6];
  scanf("%s%s",a,b);
  printf("a=%s,b=%s\n",a,b);
}
```

① 如果键盘输入　ab cd

输出结果为　a=ab,b=cd

② 如果键盘输入 abcd efghkmn

输出结果为　a=abcd,b=efghkmn

但随后弹出了致命错误窗口，提示：

Run-time Check Failure #2 – Stack around the variable 'b' was corrupted.

这是数组b越界导致的致命错误。

③ 如果键盘输入 abcdefgh kmnp

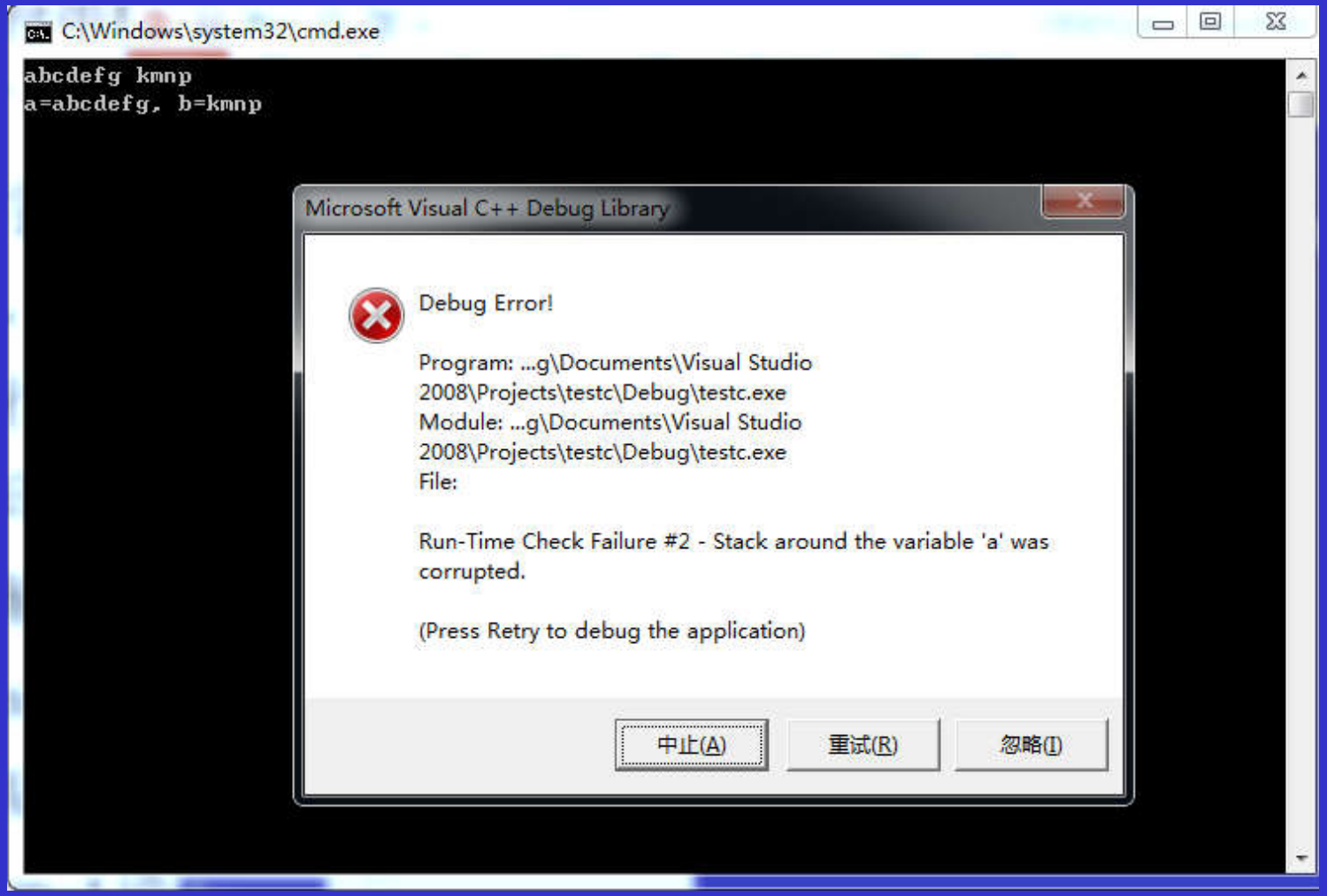
输出结果为 a=abcdefgh,b=kmnp　/* 结果可能不确定！ */

随后也弹出了致命错误窗口，提示：

Run-time Check Failure #2 – Stack around the variable 'a' was corrupted.

这是数组a越界导致的致命错误。

一旦出现数组越界,结果难以预料！随时可能出现致命错误！



需要说明的是，如果数组越界，对于上面的程序，不同编译系统的出错信息和运行结果可能会不同。

说明

① 在用格式说明符%s为字符型数组输入数据时，字符串的分隔符是空格符，因此，如果在输入的字符串中包括空格符时，只截取空格前的部分作为字符串赋给字符数组。

【例9-8】

```
#include  <stdio.h>
main( )
{ char str1[ ]= "how do you do";
  char str2[20];
  scanf("%s",str2);
  printf("%s\n",str2);
  printf("%s\n",str1);
}
```

运行时输入：

HOW DO YOU DO

输出：

HOW

how do you do

说明只读入了HOW给str2

运行时输入：

"HOW DO YOU DO"

输出：

"HOW

how do you do

运行时输入：

HOW\ DO\ YOU\ DO

输出：

HOW

how do you do

说明

② 在为字符型数组输入字符串时，输入字符串的长度不能大于数组的长度。特别要注意的是，字符串中还有一个字符串结束符，它虽然不计入字符串的长度中，但它实际需要占一个字节空间（即占一个字符元素的空间）。

9.3.4 常用字符串处理函数

头文件
<string.h>

1. puts(字符数组名)

功能：输出一个字符串到屏幕上，并在串尾自动添加一个回车符

例如：

```
#include  <stdio.h>
#include  <string.h>
main( )
{ char  str[ ]="China\nBeijing";
   puts(str);
}
```

字符串中'\n'为换行符。

输出结果为:

```
China
Beijing
请按任意键继续. . .
```

9.3.4 常用字符串处理函数

头文件
<string.h>

2. gets(字符数组名)

功能: 从键盘读入一行字符到字符数组，包括空格、制表符也会被一并读入，直到遇到回车符，但不会把回车符读入到字符串中。函数返回值为字符数组的首地址。

例如：

```
#include  <stdio.h>
#include  <string.h>
main( )
{   char  s[80];
    gets(s);
    puts(s);
}
```

如果输入:　　ab cdef
则输出结果为:　ab cdef

从键盘输入一行字符"ab cdef"到字符数组s中，包括空格。

3. strcat(str1, str2)

功能: 将字符串str2(字符串2)连接到字符串str1(字符数组1)的后面，并返回字符串str1的串首地址。

例如有程序段:

```
char  s1[20]= "abcd";
char  s2[ ]="cdef";
char  s3[20]= "\0";
strcat(s1, s2);
printf("%s\n", s1);
strcat(strcat(s3, s2), s1);
printf("%s\n", s3);
```

则输出结果为:

```
abcdcdef

cdefabcdcdef

请按任意键继续. . .
```

注意

① 字符数组1的长度必须足够大,以便能容纳被连接的字符串2。

② 连接后系统将自动取消字符串str1后面的结束符'\0'。

③ 字符串2可以是字符数组名，也可以是字符串常量，如

strcat(s1, "cdef"); 但不能： strcat("abcd", "cdef");

因为字符串常量不允许被修改，且串尾也没空间容纳新串。

4. strncat(str1, str2 , n)

功能：将字符串str2最多前n个字符连接到字符串str1的后面，并返回字符串str1的串首地址。

例如有程序段：

```
char  s1[20]= "abcd";
char  s2[ ]="cdef";
char  s3[20]= "\0";
strncat(s1, s2, 2);
printf("%s\n", s1);
strncat(s1, s2, 5);
printf("%s\n", s1);
```

则输出结果为：

```
abcdcd

abcdcdcdef

请按任意键继续. . .
```

注意

① 字符串str1所在的字符数组的长度必须足够大，以便能容纳被连接的字符串str2。

② 如果字符串str2的字符数大于n，那么该函数只将str2的前n个字符附加在字符串str1串尾。如果str2中的字符数小于n，该字符会将str2的所有字符附加在str1串尾。无论哪种情况，都会在新字符串的串尾添加结束符'\0'。

5. strcpy(字符数组1, str2)

功能：将字符串str2复制到字符数组1中，覆盖原字符串中的字符

例如，
```
char  s1[10]="abcdefg";
char  s2[ ]="efgh";
strcpy(s1,s2);     /* 不能 s1=s2; */
printf("%s\n",s1);
```

则输出结果为：　　efgh
但输出结果不会是：efghefg

注意

① 字符数组1的长度必须足够大，以便能容纳字符串str2。
② str2可以是字符数组名，也可以是字符串常量，如
　strcpy(s1, "efgh");
③ 字符串只能用复制函数，不能用赋值语句进行赋值。但单个字符可以用赋值语句赋给字符变量或字符数组元素。

6. strncpy(字符数组1, str2, n)

功能：将字符串str2前n个字符复制到字符数组1中

例如: char s2[ ]="abcde";
char s1[10];
strncpy(s1, s2, 3);
s1[3]= '\0';
printf("%s\n", s1);

则输出结果为:
abc

注意

① 如果字符串str2的字符个数超过n个，strncpy函数只复制str2字符串的前n个字符到字符数组1中，不复制也不自动添加字符串结束符'\0'，需要编程者自己去添加字符串结束符'\0'。

② 如果字符串str2的字符个数少于n个，strncpy函数将复制整个str2字符串到字符数组1中，包括字符串结束符'\0'。

上面的例子中，如果去掉s1[3]= '\0'; 则输出结果为:
abc烫烫烫烫烫烫烫烫蘥bcde

7. strcmp(字符串1,字符串2)

功能：按照字典序比较两个字符串大小

这个函数的返回值如下：
(1) 若字符串1 = 字符串2, 则返回值为0;
(2) 若字符串1 > 字符串2, 则返回值为1;
(3) 若字符串1 < 字符串2, 则返回值为-1。

注意

① 执行这个函数时，自左到右逐个比较两个字符串对应位置字符的ASCII码值，直到发现了不同字符或字符串结束符'\0'为止。并以最后一个不相同字符的ASCII码值的大小决定两个字符串的大小。
② 对字符串不能直接用关系运算符 == != > >= < <= 进行比较。若a和b是字符串， if (a>b) 是错误的。
③ “字符串1”与“字符串2”可以是字符数组名，也可以是字符串常量。

例如：

```
#include   <stdio.h>
#include   <string.h>
main( )
{  char  a[6]="abc", b[6]="abcd";
   if (strcmp(a, b)>0)
         printf("%s,%s\n", a, b);
   else
         printf("%s,%s\n", b, a);
}
```

输出结果：

abcd,abc

因为 '\0' < 'd'

8. strlen(字符串)

功能：求字符串长度(不包括结束符 '\0')

例如， char s[10]= "abcde";

printf("%d\n",strlen(s));

则输出结果为： 5

在使用这个函数时, strlen(字符串) 中的“字符串”可以是字符数组名，也可以是字符串常量。

printf("%d, %d\n",sizeof(s), strlen(s));
结果是： 10, 5
再次提醒：注意 sizeof(s)与strlen(s)的结果不相同。

9. 大小写转换函数

strlwr(字符串) 将字符串中大写字母转换成小写字母

strupr(字符串) 将字符串中小写字母转换成大写字母

例如有程序：

```
#include  <stdio.h>
#include  <string.h>
main( )
{  char s1[]="AbCDefgH123";
   char s2[]="AbCDefgH123";
   strlwr(s1);
   printf("%s\n",s1);
   strupr(s2);
   printf("%s\n",s2);
}
```

输出结果：

abcdefgh123

ABCDEFGH123

10. sprintf(字符数组名, "输出格式", 变量列表)

功能：结果输出到字符数组中，对应printf 的输出到屏幕上。

例如：

```
#include  <stdio.h>
#include  <string.h>
main( )
{ char  str[50];
   int  k=20;
   double f=123.4;
   sprintf(str, "k=%4d f=%8.3f", k, f);
   puts(str);   /* 输出字符串str */
}
```

输出结果为：

k= 20 f= 123.400

说明sprintf执行后，

str="k= 20 f= 123.400"

11. sscanf(字符数组名, "输入格式", 变量列表)

功能：从字符数组中读入数据。对应scanf从键盘上读入数据

例如：

```
#include  <stdio.h>
#include  <string.h>
main( )
{ char  str[50];
   int  k=20,m;
   double f=123.4, d;
   sprintf(str, "k=%4d f=%8.3f", k, f);
   sscanf(str, "k=%d f=%lf", &m, &d);
   printf("m=%4d d=%8.3f\n", m, d);
}
```

输出结果为:

m= 20 d= 123.400

9.4 数组作为函数参数

- C语言规定,数组名可以作为函数的形参

9.4.1 形参数组与实参数组的结合

【例9-9】十个小孩围成一圈分糖果。首先， 老师分给第一个小孩10块，第二个小孩2块，第三个小孩8块，第四个小孩22块，第五个小孩16块，第六个小孩4块，第七个小孩10块，第八个小孩6块，第九个小孩14块，第十个小孩20块。然后按如下方法将每个小孩手中的糖果进行调整：所有的小孩检查自己手中的糖块数，如果糖块数为奇数，则向老师再要一块，再同时将自己手中的糖分一半给下一个小孩。问需要多少次调整后，每个小孩手中的糖块数都相等？每人各有多少块糖？

三个模块

- 每次调整后检查每个小孩手中的糖果数是否相等，相等返回0。函数为:

```
int flag(int a[ ],  int n)
{   int k;
    for (k=1; k<n; k++)
       if (a[0] != a[k])
          return 1;
    return 0;
}
```

- 每次调整后,输出调整次数以及当前每个小孩手中的糖果数。函数为:

```
void pr(int k,int b[ ],int n)
{   int j;
    printf("  %2d  ",k);
    for (j=0; j<n; j++)
        printf("%4d", b[j]);
    printf("\n");
}
```

● 编写一个主函数,在主函数中首先初始化每个小孩手中的糖果数, 输出表头以及开始时每个小孩手中的糖果数。然后调用函数**flag( )**来检查每个小孩手中的糖果数是否相等,若不等则作如下调整：

① 每个小孩将自己的糖果分出一半（若是偶数块,则直接分出一半; 若不是偶数块，则向老师要一块后再分出一半）；

② 每个小孩将分出的一半给下一个小孩;

③ 调整次数加**1**;

④ 调用函数**pr( )**输出调整次数以及当前每个小孩手中的糖果数。

以上调整过程循环直到每个小孩手中的糖果数相等为止。

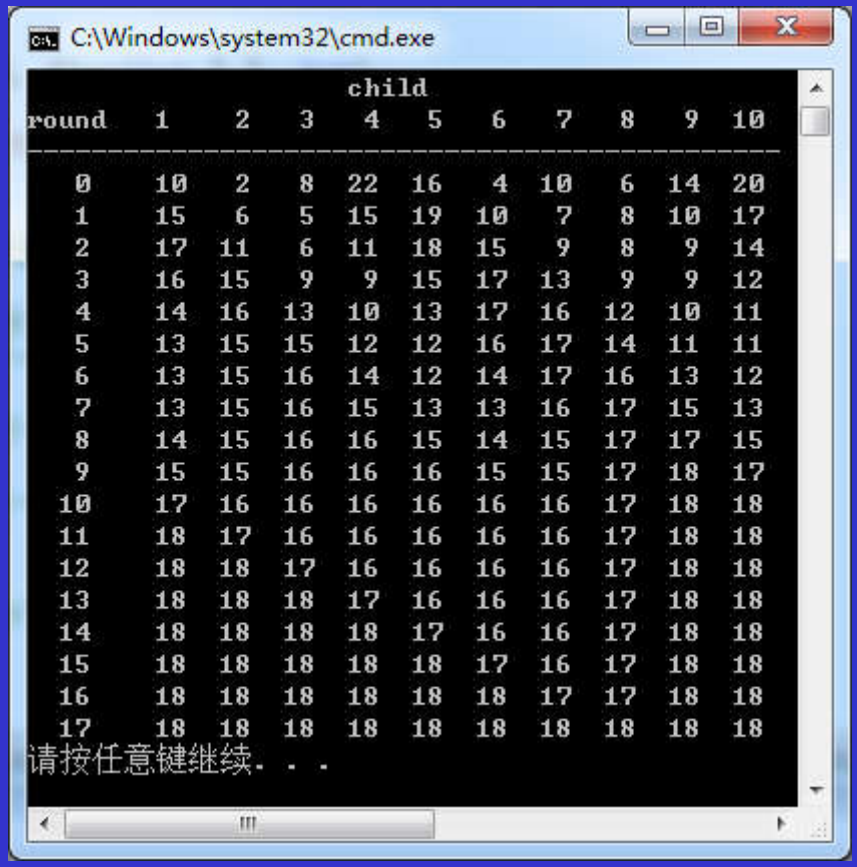
完整的C程序:

```
#include   <stdio.h>
main( )
{   int s[10]={10, 2, 8, 22, 16, 4, 10, 6, 14, 20}, k, t[10], n=0, flag(int [ ], int);
    void pr(int, int [ ], int);
    printf("                  child\nround   1    2   3   4   5   6   7   8   9  10\n");
    printf("-----------------------------------------\n");
    pr(n, s, 10); /*开始时每个小孩手中的糖果数 */
    while(flag(s, 10)) /*检查每个小孩手中糖果数是否相等,若不等则继续调整*/
    {   for (k=0; k<10; k++)  /*每个小孩将自己的糖果分出一半 */
        {  if (s[k]%2==0)  t[k] = s[k]/2;   /*若是偶数块,则直接分出一半 */
           else        t[k] = (s[k]+1)/2;  /*不是偶数,向老师要一块后再分出一半*/
        }
        for (k=0; k<9; k++)      /* 每个小孩将分出的一半给下一个小孩 */
            s[k+1] = t[k+1] + t[k];
       s[0] = t[0] + t[9];      /* 最后一个小孩将分出的一半给第一个小孩 */
       n = n+1;   /*调整次数加1 */
       pr(n, s, 10);
   }  /*调用函数pr( )输出调整次数以及当前每个小孩手中的糖果数 */
}
```

运行结果：

	child									
round	1	2	3	4	5	6	7	8	9	10
0	10	2	8	22	16	4	10	6	14	20
1	15	6	5	15	19	10	7	8	10	17
2	17	11	6	11	18	15	9	8	9	14
3	16	15	9	9	15	17	13	9	9	12
4	14	16	13	10	13	17	16	12	10	11
5	13	15	15	12	12	16	17	14	11	11
6	13	15	16	14	12	14	17	16	13	12
7	13	15	16	15	13	13	16	17	15	13
8	14	15	16	16	15	14	15	17	17	15
9	15	15	16	16	16	15	15	17	18	17
10	17	16	16	16	16	16	16	17	18	18
11	18	17	16	16	16	16	16	17	18	18
12	18	18	17	16	16	16	16	17	18	18
13	18	18	18	17	16	16	16	17	18	18
14	18	18	18	18	17	16	16	17	18	18
15	18	18	18	18	18	17	16	17	18	18
16	18	18	18	18	18	18	17	17	18	18
17	18	18	18	18	18	18	18	18	18	18

请按任意键继续...



形参数组与实参数组的结合要注意以下几点：

(1) 调用函数与被调用函数中分别定义实参和形参数组，其数组名可以不同，但类型必须一致。

(2) 在C语言中，形参变量与实参变量之间的结合是采用数值结合的，因此，如果在被调用函数中改变了形参的值是不会改变实参值的。但是，形参数组与实参数组的结合是采用地址结合的，从而可以实现数据的双向传递。在被调用函数中改变了形参数组元素的值，实际上就改变了实参数组元素的值。

(3) 实参数组与形参数组的大小可以一致也可以不一致， C编译系统对形参数组的大小不作检查，调用时只将实参数组的首地址传给形参数组。为了通用性，函数中的一维形参数组通常不指定大小。

(4) 虽然函数中的形参数组一般不指定大小，但为了控制形参数组的正确使用范围， 一般要在函数中另设一个传送形参数组元素个数的形参变量， 如函数**flag( )**与**pr( )**中的形参**n**。

(5) 在对被调用函数进行向前引用说明时，可以直接采用函数原型进行说明：

```
int flag(int a[ ], int n);
void pr(int k, int b[ ], int n);
```

也可以省略其中的形参变量名，但要注意的是，表示数组的[ ]不能省略：

```
int flag(int [ ], int);
void pr(int, int [ ], int);
```

9.4.2 二维数组作为函数参数

【例9-10】编写一个函数求两个矩阵的乘积矩阵。

```
#include  <stdio.h>
main( )
{ int  i, j, c[2][3];
  int  a[2][4]={1, 2, 3, 4, 5, 6, 7, 8};
  int  b[4][3]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
  void matmul(int [ ][4], int [ ][3], int [ ][3], int, int, int);
  matmul(a, b, c, 2, 4, 3);
  for (i=0; i<2; i++)
  {   for (j=0; j<3; j++)
          printf("%5d", c[i][j]);
      printf("\n");
  }
  printf("\n");
}
```

```
void matmul(int a[2][4], int b[4][3], int c[2][3], int m, int n, int k)
{   int  i, j, t;
     for (i=0; i<m; i++)
        for (j=0; j<k; j++)
        {     c[i][j]=0;
              for (t=0; t<n; t++)
                   c[i][j] += a[i][t] * b[t][j];
        }
   return;
}
```

运行结果为:

```
   70    80    90
  158  184  210
```

按任意键继续…

需要说明的是，在用二维数组名作为函数参数时，在被调用函数中对形参数组说明时，可以指定每一维的大小，也可以省略第一维的大小说明。例如，在例9.10的程序中，函数matmul( )中定义了三个二维形参数组a，b，c：

```
int  a[2][4], b[4][3], c[2][3];
```

在这个定义中，对三个形参数组都指定了每一维的大小，它们分别与实参数组的大小相同。但也可以省略其中第一维的大小，即可以用下列方式进行定义：

```
int  a[ ][4], b[ ][3], c[ ][3];
```

但在定义形参数组时不能省略第二维的大小。

例如，下列两种定义都是错误的：

```
int  a[2][ ], b[4][ ], c[2][ ];
```

与

```
int  a[ ][ ], b[ ][ ], c[ ][ ];
```

- 在被调用函数中，不能用形参变量来定义二维数组中的各维大小。以下对形参数组的定义是错误的：

void matmul(int a[m][n],int b[n][k],int c[m][k],int m,int n,int k);

- 当实参数组为二维数组时,虽然其形参也是二维数组,但像

void matmul(int a[2][4], int b[4][3], int c[2][3], int m, int n, int k)

这种定义二维形参数组的方法的函数是没有通用性的。

- 即使实参是二维数组，而形参是一维数组，它们的结合还是地址结合。因此，可以将二维的实参数组的存储空间看成是一个元素个数相同的一维数组的存储空间, 在用二维数组作为形参时, 可以转化为一维数组来处理。将例9-10中的程序改成：

- 一般情况下,如果一个二维数组（矩阵）的列数为n（即一行中有n个元素）,则该二维数组中行标为i、列标为j的元素所对应的一维数组元素的下标为i*n+j。

```
#include   <stdio.h>
main( )
{ int  i, j, c[2][3];    int  a[2][4]={1,2,3,4,5,6,7,8};
   int  b[4][3]={1,2,3,4,5,6,7,8,9,10,11,12};
   void matmul(int a[ ],int b[ ],int c[ ], int, int, int); /* 函数向前引用说明 */

   matmul((int *)a, (int *)b, (int *)c, 2, 4, 3);  /* 函数调用 */
   for (i=0; i<2; i++)
   {   for (j=0; j<3; j++)
          printf("%5d",c[i][j]);
       printf("\n");
   }
   printf("\n");
}
void matmul(int a[ ], int b[ ], int c[ ], int m, int n, int k) /* 函数定义 */
{     int  i, j, t;
      for (i=0; i<m; i++)
        for (j=0; j<k; j++)
        {   c[i*k+j]=0;
            for (t=0; t<n; t++)
                  c[i*k+j] += a[i*n+t]*b[t*k+j];
        }
}
```

- 在上述程序中，主函数与原来的一样，但函数matmul( )中的三个矩阵（包括乘积矩阵）均定义为一维的形参数组，在实际引用时，根据二维数组中的元素以行为主存储的原则，将二维数组元素中的两个下标（行标与列标）转换成一维数组元素的下标，从而实现一维数组元素与二维数组元素的对应。这样写出的矩阵相乘函数：

void matmul(int a[ ], int b[ ], int c[ ], int m, int n, int k)

可以实现任意A_{mk}与B_{km}的乘积，函数具有通用性。

- 但函数调用时需要用强制类型转换(int *)，把二维数组强制转换为一维数组，否则编译时，编译系统会给出类型不一致的警告错误信息。例如，如果将语句：

matmul((int *)a, (int *)b, (int *)c, 2, 4, 3);

改为：

matmul(a, b, c, 2, 4, 3);

编译结果是：

正在编译...

ex9-10.c

ex9-10.c(6) : warning C4047: “函数”: “int *”与“int [2][4]”的间接级别不同

ex9-10.c(6) : warning C4024: “matmul”: 形参和实参 1 的类型不同

ex9-10.c(6) : warning C4047: “函数”: “int *”与“int [4][3]”的间接级别不同

ex9-10.c(6) : warning C4024: “matmul”: 形参和实参 2 的类型不同

ex9-10.c(6) : warning C4047: “函数”: “int *”与“int [2][3]”的间接级别不同

ex9-10.c(6) : warning C4024: “matmul”: 形参和实参 3 的类型不同

ex9-10 - 0 个错误，6 个警告

【例9-11】从键盘输入6行6列二维整型数组的数据,求每行元素的均值。

```
#include  <stdio.h>
main( )
{    int a[6][6], i, j;
     void avg(int s[ ], int, double t[ ]);
     double b[6];
     printf("input MAT a:");  /* 输入前的提示 */
     for (i=0; i<6; i++)
        for (j=0; j<6; j++)
           scanf("%d", &a[i][j]);  /*  逐行输入二维数组a的各元素 */
     avg((int *)a, 6, b);        /* 计算二维数组a每一行元素的平均值,
                                    顺序存放在一维数组b中 */
     for (i=0; i<6; i++)
     {    for (j=0; j<6; j++)  /* 输出数组a中的一行元素 */
             printf("%7d", a[i][j]);
          printf("%15e\n", b[i]);  /* 输出数组a中一行元素的平均值*/
     }
 }
```

```
void avg(int s[ ],int n,double t[ ])
/*计算形参数组s中每一行元素的平均值,顺序存放在形参数组t中*/
{    int i, j;
     for (i=0; i<n; i++)
     {   t[i]=0.0;
         for (j=0; j<n; j++)   /* 每一行元素的和 */
             t[i] += s[i*n+j];
         t[i] /= n;   /* 每一行元素的均值 */
      }
}
```

注意：

调用函数 avg(a, 6, b);时,需要把二维数组a强制转换为一维数组：

```
avg( (int *)a, 6, b);
```

运行结果为：

```
input MAT a:
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 18 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
    1    2    3    4    5    6  3.500000e+000
    7    8    9   10   11   12  9.500000e+000
   13   14   15   18   17   18  1.583333e+001
   19   20   21   22   23   24  2.150000e+001
   25   26   27   28   29   30  2.750000e+001
   31   32   33   34   35   36  3.350000e+001
```

9.5 算法举例

1. 有序表的二分查找

指线性表中的元素按值非递减排列(即从小到大,但允许相邻元素值相等)

二分查找只适用于顺序存储的有序表。

设有序线性表的长度为n,被查元素为x,则二分查找的方法如下:

(1) 将x与线性表的中间项进行比较:

(2) 若中间项的值等于x，则说明查到，查找结束;

(3) 若x小于中间项的值，则在线性表的前半部分（即中间项以前的部分）以相同的方法进行查找;

(4) 若x大于中间项的值，则在线性表的后半部分（即中间项以后的部分）以相同的方法进行查找。

这个过程一直进行到查找成功或子表长度为0（说明线性表中没有这个元素）为止。

注意：只有当有序线性表为顺序存储时才能采用二分查找(当有序线性表采用链式或其他存储形式时，二分查找也不适用)。

/*函数返回被查找元素x在线性表中的序号(即一维数组下标), 如果在线性表中不存在元素值x,则返回 -1 */

```
int  bsearch(ET v[ ], int n, ET x)
{   int  i, j, k;
    i=0; j=n-1;
    while (i<=j)
    {   k=(i+j)/2;
        if (x==v[k])  return(k);
        if (x<v[k])  j=k-1;
        else  i=k+1;
    }
    return(-1);
}
```

可以证明,对于长度为n的有序线性表,在最坏情况下,二分查找只需要比较$\log_2 n$次,而顺序查找需要比较n次。因此, 二分查找的效率要比顺序查找高得多。

ET表示可以是任何数值类型标识符（char, short, int, long, float, double等），它根据线性表中实际的元素类型来确定。例如，如果线性表为整型，则应为 int；如果线性表为实型，则应为 float 或 double 。

2.冒泡排序

双向冒泡排序的基本过程如下：

- 首先, 从表头开始往后扫描线性表,在扫描过程中逐次比较相邻两个元素的大小。若相邻两个元素中,前面的元素大于后面的元素, 则将它们互换,称之为消去了一个逆序。显然，在扫描过程中，不断地将两相邻元素中的大者往后移动，最后就将线性表中的最大者换到了表的最后,这也是线性表中最大元素应有的位置。
- 然后从后向前扫描剩下的线性表，同样，在扫描过程中逐次比较相邻两个元素的大小。若相邻两个元素中，后面的元素小于前面的元素，则将它们互换，这样就又消去了一个逆序。显然，在扫描过程中，不断地将两相邻元素中的小者往前移动，最后就将剩下线性表中的最小者换到了表的最前面，这也是线性表中最小元素应有的位置。
- 对剩下的线性表重复上述过程，直到剩下的线性表变空为止，此时的线性表已经变为有序。

原序列	5	1	7	3	1	6	9	4	2	8	6
第1遍(从前往后)	5↔1		7↔3	↔1	↔6		9↔4	↔2	↔8	↔6	
结果	1	5	3	1	6	7	4	2	8	6	9
(从后往前)	1	5↔3	↔1	6↔7	↔4	↔2	8↔6		9		
结果	1	1	5	3	2	6	7	4	6	8	9
第2遍(从前往后)	1	1	5↔3	↔2	6	7↔4	↔6	8	9		
结果	1	1	3	2	5	6	4	6	7	8	9
(从后往前)	1	1	3↔2	5↔6	↔4	6	7	8	9		
结果	1	1	2	3	4	5	6	6	7	8	9
第3遍(从前往后)	1	1	2	3	4	5	6	6	7	8	9
最后结果	1	1	2	3	4	5	6	6	7	8	9

图9-1 (双向)冒泡排序过程示意图

冒泡排序的C函数如下：

假设线性表的长度为n，则在最坏情况下，冒泡排序需要经过n/2遍的从前往后扫描和n/2遍从后往前的扫描，需要的比较次数为n(n-1)/2。但这个工作量不是必需的，一般情况下要小于这个工作量。

```
void  bubsort(ET p[ ] , int n)
{ int m,k,j,i;
  ET d;
  k=0; m=n-1;
  while (k<m)  /*子表未空*/
  {  j=m-1; m=0;
    for (i=k; i<=j; i++)  /*从前往后扫描子表*/
      if (p[i]>p[i+1])  /*发现逆序进行交换*/
      { d=p[i]; p[i]=p[i+1]; p[i+1]=d; m=i; }
    j=k+1; k=0;
    for (i=m; i>=j; i--)  /*从后往前扫描子表*/
      if (p[i-1] >p[i])  /*发现逆序进行交换*/
      { d=p[i]; p[i]=p[i-1]; p[i-1]=d; k=i; }
  }
  return;
}
```

3.选择排序

基本思想

- 扫描整个线性表，从中选出最小的元素，将它交换到表的最前面(这是它应有的位置)；然后对剩下的子表采用同样的方法，直到子表空为止。
- 对于长度为n的序列，选择排序需要扫描n-1遍，每一遍扫描均从剩下的子表中选出最小的元素，然后将该最小的元素与子表中的第一个元素进行交换。图9-2是这种排序的示意图，图中有方框的元素是刚被选出来还未交换的最小元素。

原序列	89	21	56	48	85	16	19	47
第1遍选择	16	21	56	48	85	89	19	47
第2遍选择	16	19	56	48	85	89	21	47
第3遍选择	16	19	21	48	85	89	56	47
第4遍选择	16	19	21	47	85	89	56	48
第5遍选择	16	19	21	47	48	89	56	85
第6遍选择	16	19	21	47	48	56	89	85
第7遍选择	16	19	21	47	48	56	85	89

选择排序的C函数如下：

```
void selesort(ET  p[ ] , int  n)
{  int  i,j,k;
   ET  d;
   for (i=0; i <= n-2; i++)
   {   k=i;
       for (j=i+1; j<=n-1; j++)
           if (p[j]<p[k])
               k=j;
       if (k != i)
       { d=p[i]; p[i]=p[k]; p[k]=d; }
   }
   return;
}
```

选择排序在最坏情况下需要比较n(n-1)/2次。

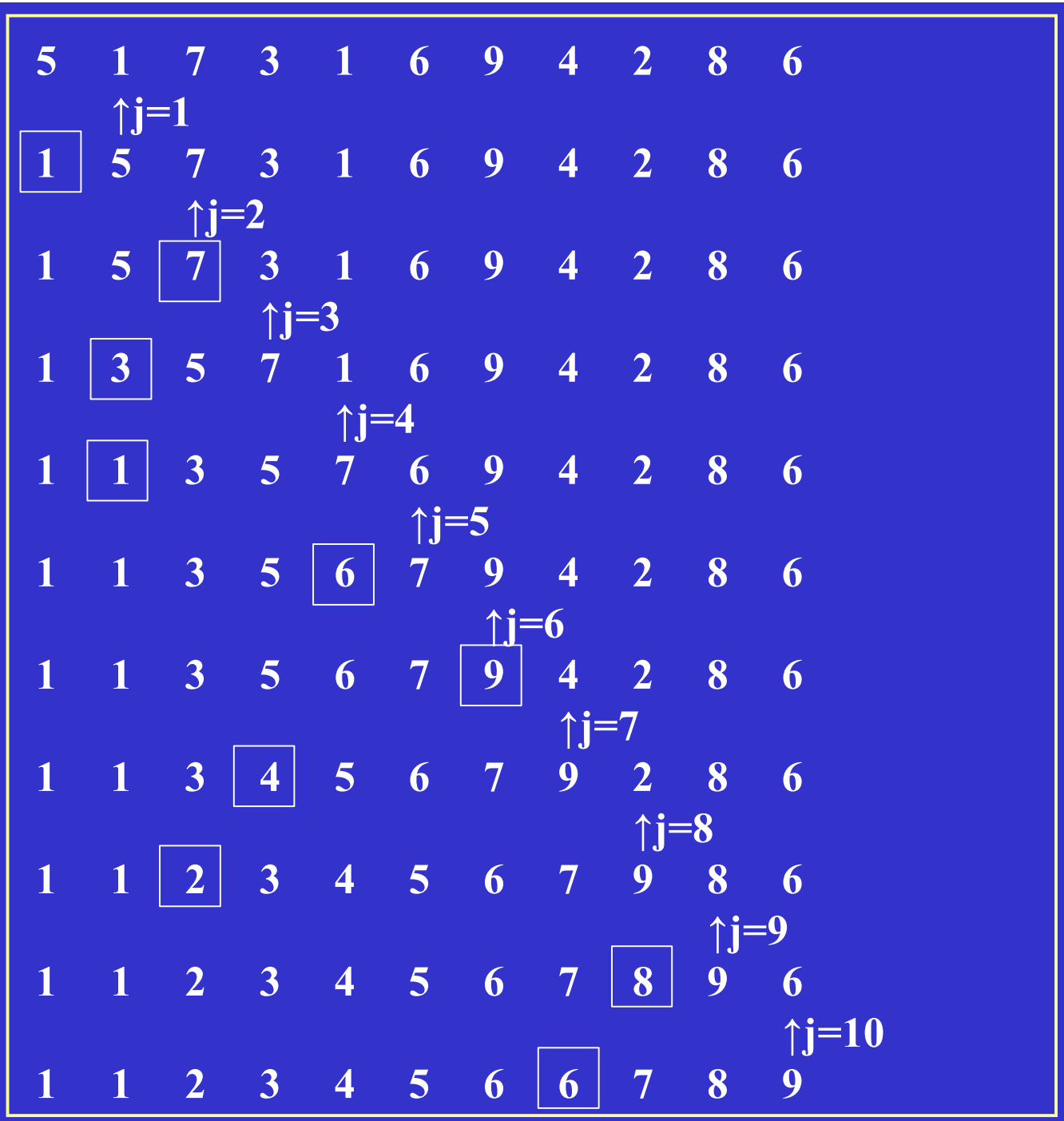
4. 插入排序

插入过程如下：

- 首先将第j个元素放到一个变量T中。然后从有序子表的最后一个元素（即线性表中第j-1个元素）开始，往前逐个与T进行比较，将大于T的元素均依次向后移动一个位置，直到发现一个元素不大于T为止，此时就将T（即原线性表中的第j个元素）插入到刚移出的空位置上，有序子表的长度就变为j了。

```
void insort(ET  p[ ] , int  n)
{ int  j,k;
  ET  t;
  for (j=1; j<n; j++)
  { t=p[j]; k=j-1;
    while ((k>=0)&&(p[k]>t))
    {  p[k+1]=p[k];  k=k-1;  }
    p[k+1]=t; /* 插入 */
  }
  return;
}
```

这种排序方法的效率与冒泡排序法相同。在最坏情况下，插入排序需要n(n-1)/2次比较。


5 1 7 3 1 6 9 4 2 8 6
↑j=1
1 5 7 3 1 6 9 4 2 8 6
↑j=2
1 5 7 3 1 6 9 4 2 8 6
↑j=3
1 3 5 7 1 6 9 4 2 8 6
↑j=4
1 1 3 5 7 6 9 4 2 8 6
↑j=5
1 1 3 5 6 7 9 4 2 8 6
↑j=6
1 1 3 5 6 7 9 4 2 8 6
↑j=7
1 1 3 4 5 6 7 9 2 8 6
↑j=8
1 1 2 3 4 5 6 7 9 8 6
↑j=9
1 1 2 3 4 5 6 7 8 9 6
↑j=10
1 1 2 3 4 5 6 6 7 8 9

5. 数值与数组元素下标的映射

【例9-12】人口普查时，需要统计各个年龄段的人数，共分为11个年龄段：0~9岁，10~19岁，20~29岁，30~39岁，40~49岁，50~59岁，60~69岁，70~79岁，80~89岁，90~99岁，100岁及以上。现有n个人的年龄在a数组中，请编程统计各年龄段的人数，统计结果存入数组c[11]。

```
#include   <stdio.h>                    /* 方法1 */
void count1(int a[ ], int n, int c[ ])
{       int i=0;
        for (i=0; i<11; i++) c[i] = 0;
        for (i=0; i<n; i++)
        {       if (a[i] >=0 && a[i] <=9)     c[0]++;
                if (a[i] >=10 && a[i] <=19) c[1]++;
                if (a[i] >=20 && a[i] <=29) c[2]++;
                if (a[i] >=30 && a[i] <=39) c[3]++;
                if (a[i] >=40 && a[i] <=49) c[4]++;
                if (a[i] >=50 && a[i] <=59) c[5]++;
                if (a[i] >=60 && a[i] <=69) c[6]++;
                if (a[i] >=70 && a[i] <=79) c[7]++;
                if (a[i] >=80 && a[i] <=89) c[8]++;
                if (a[i] >=90 && a[i] <=99) c[9]++;
                if (a[i] >=100) c[10]++;
        }
}
```

```
#include   <stdio.h>                    /* 方法1另一种写法 */
void count1A(int a[ ], int n, int c[ ])
{       int i=0;
        for (i=0; i<11; i++) c[i] = 0;
        for (i=0; i<n; i++)
        {       if (a[i] <=9) c[0]++;
                else if (a[i] <=19) c[1]++;
                else if (a[i] <=29) c[2]++;
                else if (a[i] <=39) c[3]++;
                else if (a[i] <=49) c[4]++;
                else if (a[i] <=59) c[5]++;
                else if (a[i] <=69) c[6]++;
                else if (a[i] <=79) c[7]++;
                else if (a[i] <=89) c[8]++;
                else if (a[i] <=99) c[9]++;
                else if (a[i] >=100) c[10]++;
        }
}
```

```
#include   <stdio.h>                          /*方法1 又一种写法 */
void count1B(int a[ ], int n, int c[ ])
{          int i=0;
           for (i=0; i<11; i++) c[i] = 0;
           for (i=0; i<n; i++)
           {     if (a[i] <=59)
                      if (a[i] <=29)
                           if (a[i] <=19)
                                if (a[i] <=9) c[0]++;
                                else c[1]++;
                           else c[2]++;
                      else if (a[i] <=39) c[3]++;
                                else if (a[i] <=49) c[4]++;
                                     else c[5]++;
                 else if (a[i] <=89)
                           if (a[i] <=69) c[6]++;
                           else if (a[i] <=79) c[7]++;
                                else  c[8]++;
                      else if (a[i] <=99) c[9]++;
                           else c[10]++;
           }
 }
```

```
#include   <stdio.h>                          /* 方法2 */
void count1C(int a[ ], int n, int c[ ])
{          int i=0;
           for (i=0; i<11; i++) c[i] = 0;
           for (i=0; i<n; i++)
           {    switch (a[i]/10)
                {
                  case 0:     c[0]++; break;
                  case 1:     c[1]++; break;
                  case 2:     c[2]++; break;
                  case 3:     c[3]++; break;
                  case 4:     c[4]++; break;
                  case 5:     c[5]++; break;
                  case 6:     c[6]++; break;
                  case 7:     c[7]++; break;
                  case 8:     c[8]++; break;
                  case 9:     c[9]++; break;
                  default:    c[10]++;
                }
           }
}
```

```
void count2(int a[ ], int n, int c[ ])    /* 方法3 */
{       int i=0, p;
        for (i=0; i<11; i++) c[i] = 0;
        for (i=0; i<n; i++)
        {       p = a[i] /10;  /*数值与数组元素下标的映射 */
                if (p > 10)
                        p = 10;
                c[p]++;  /* 相应年龄段的计数器加1 */
        }
}
```

补充例题。现有一个字符串s，请编程统计其中各个字母出现的次数，统计结果存入数组c[26]。统计时，字母不区分大小写。

```
#include<stdio.h> /* 方法1 */
#include<string.h>
void count1(char a[ ], int c[ ])
{ int i=0,  n=strlen(a);
   for (i=0; i<26; i++) c[i] = 0;
   for (i=0; i<n; i++)
    if ( (a[i] >='a' && a[i] <='z') ||
          (a[i] >='A' && a[i] <='Z'))
     switch (a[i])
    { case 'a': case 'A':  c[0]++; break;
      case 'b': case 'B':  c[1]++; break;
      case 'c': case 'C':  c[2]++; break;
      case 'd': case 'D':  c[3]++; break;
      case 'e': case 'E':   c[4]++; break;
      case 'f': case 'F':   c[5]++; break;
      case 'g': case 'G':  c[6]++; break;
      case 'h': case 'H':  c[7]++; break;
      case 'i': case 'I':    c[8]++; break;
      case 'j': case 'J':   c[9]++; break;
      case 'k': case 'K':  c[10]++; break;
      case 'l': case 'L':   c[11]++; break;
      case 'm':case 'M': c[12]++; break;
      case 'n': case 'N':  c[13]++; break;
      case 'o': case 'O':  c[14]++; break;
      case 'p': case 'P':  c[15]++; break;
      case 'q': case 'Q':  c[16]++; break;
      case 'r': case 'R':  c[17]++; break;
      case 's': case 'S':   c[18]++; break;
      case 't': case 'T':   c[19]++; break;
      case 'u': case 'U':  c[20]++; break;
      case 'v': case 'V':  c[21]++; break;
      case 'w':case 'W': c[22]++; break;
      case 'x': case 'X':  c[23]++; break;
      case 'y': case 'Y':  c[24]++; break;
      case 'z': case 'Z':  c[25]++; break;
    }
}
```

```
void count2(char a[ ], int c[ ]) /* 方法2 */
{       int i=0,p, n=strlen(a);
        for (i=0; i<26; i++)
            c[i] = 0;
        for (i=0; i<n; i++)
        {  if (a[i] >='a' && a[i] <='z')
                p = a[i] - 'a'; /*数值与数组元素下标的映射 */
           else if (a[i] >='A' && a[i] <='Z')
                p = a[i] - 'A'; /*数值与数组元素下标的映射 */
           else
                continue;   /* 不是字母，不统计 */
           c[p]++;   /* 相应字母的计数器加1 */
        }
}
```

6. 用递归方式在数组中找最大值

递归思想：如果能先在前n-1个元素中找出最大值t，则只需在t和a[n-1]中求最大值返回即可。

```
ET  FindMax(ET a[ ], int n)
{   ET  t;
    if (n > 1)
    {   t = FindMax(a, n-1);
        return  t>a[n-1] ? t : a[n-1];
    }
    else  return  a[0];
}
```

7. 用数组存超长整数并计算

用 a[0],a[1],a[2]…… 分别表示超长整数的个位、十位、百位……，只要电脑内存足够大，就基本对整数的长度没有限制。

【例9-13】 编程计算并输出m！，m从键盘上输入，要求结果精确到个位。

a[N-1],a[N-2],…… a[n],a[n-1], ……a[2],a[1],a[0]

初值：　a[0]=1;

外循环：　k=1,2,……m

　　a[0]=a[0]*k;

内循环：　n=1,2,……　/* 向前进位 */

　　a[n] = a[n]*k + a[n-1]/10;

　　a[n-1] %= 10;

```
#include <stdio.h>
#define  N 10000
main( )
{ int a[N]={1};  /* a[0]=1 */
   int n, k, m;
   scanf("%d", &m);
   for (k=2; k<=m; k++)
   {   a[0]=a[0]*k;
       for (n=1;n<N; n++)
       {    a[n] = a[n]*k + a[n-1]/10;
            a[n-1] %= 10;
       }
   }
   n=N-1;
   while(a[n]==0)
     n--; /*从最高位找第一个非零数*/
   printf("%d! = ", m);
   for (;n>=0;n--)
       printf("%d", a[n]);
   printf("\n");
}
```

运行结果：

```
50
50! =
30414093201713378043612608166064768844377641568960512000000000000
请按任意键继续. . .

100
100! =
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
请按任意键继续. . .
```

500
500! =
12201368259911100687012387854230469262535743428031928421924135883
8584537315388199760549644750220328186301361647714820358416337872
2078177200480785205159329285477907571939330603772960859086270429
1745478824249127263443056701732707694610628023104526442188787894
6575477714986349436778103764427403382736539747138647787849543848
9595537537990423241061271326984327745715546309977202781014561081
1883737095310163563244329870295638966289116589747695720879269288
7128178007026517450776841071962439039432253642260523494585012991
8571501248706961568141625359056693423813008856249246891564126775
6544818865065938479517753608940057452389403357984763639449053130
6232374906644504882466507594673586207463792518420045936969298102
2263971952597190945217823331756934581508552332820762820023402626
90789834245171200620771464097945611161276291459512372299133401695
5236385094288559201872743379517301458635757082835578015873543276
8888680120399882384702151467605445407663535984174430480128938313
8968816394874696588175045069263653381750554781286400000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
请按任意键继续. . .

(1135位）

【补充例题】

8. 有序表中数组元素的插入

方法1：有序表的插入,先找到合适的位置,向后移动元素, 插入。

```
void  insert1(ET v[ ], int n, ET x)
{   int  i, j;
    for(i=0; i<n; i++)
       if (v[i] < x)  break;
    if (i < n)
      for (j=n-1; j>=i; j--)
        v[j+1] = v[j]; /*从最后一个元素开始,向后移动元素,腾出位置 */
    v[i] = x;  /* 插入元素 */
}
```

8. 有序表中数组元素的插入

方法2：有序表的插入,边找合适的位置边向后移动元素, 插入。

```
void  insert2(ET v[ ], int n, ET x)
{ int  j;
   for (j=n-1; j>=0 && x< v[j]; j--)
       v[j+1] = v[j]; /* 从最后一个元素开始,向后移动元素,腾出位置 */
   v[j+1] = x;  /* 插入元素 */
}
```

9. 有序表中数组元素的删除

有序表元素的删除，先找到合适的位置，向前移动元素。

```
void  delete(ET v[ ], int n, ET x)
{   int  i, j;
    for(i=0; i<n; i++)
       if (v[i] == x)  break;
    if (i < n-1)
       for (j=i; j<n-1; j++)
           v[j] = v[j+1];  /* 从后一个元素开始,向前移动元素,补充空出的位置 */
}
```

注意：因为一维数组的元素是连续存放的，如果要插入或删除元素，不得不向后或向前移动其他元素。

10. 一般表的顺序查找

无序的一般表，只能顺序逐个查找。

```
int  search(ET v[ ],int n, ET x)
{   int  i;
    for(i=0; i<n; i++)
        if (v[i]==x)  return  i;
    return(-1);
 }
```

这个过程一直进行到查找成功或全表都找过,但没有找到。

找到的平均查找次数： (1+2+3+…….+n)/n = (n+1)/2

最坏情况下，找不到的查找次数：n

11. 用递归方式进行数组元素的排序

递归思想：如果前n-1个元素能先排好序，则只需在把最后一个元素a[n-1]插入到长度为n-1的a数组中即可。

```
void  RecurSort(ET a[ ], int n)
{  if (n > 1)
    {    RecurSort(a, n-1);
         insert2(a,  n-1, a[n-1]); /*有序表的插入*/
    }
}
```

```
#include <stdio.h>
void  insert2(int v[ ], int n, int x)
{   int  j;
    for (j=n-1; j>=0 && x< v[j]; j--)
          v[j+1] = v[j];
    v[j+1] = x;
}
void  RecurSort(int a[ ], int n)
{   if (n > 1)
    {   RecurSort(a, n-1);
        insert2(a,  n-1, a[n-1]);
    }
}
main( )
{ int a[10]={2,1,10,9,6,7,3,4,8,5},i;
  for (i=0; i<10; i++)
          printf("%d ", a[i]);
  printf("\n");
  RecurSort(a, 10);
  for (i=0; i<10; i++)
          printf("%d ", a[i]);
  printf("\n");
}
```

运行结果：

2 1 10 9 6 7 3 4 8 5

1 2 3 4 5 6 7 8 9 10

按任意键继续…

12. 黑色星期五问题

如果某个月的13号正好是星期五，这天被称为“黑色星期五”。

1. 试编写程序，统计出在某个年份中，出现了多少次“黑色星期五”，并给出这一年出现“黑色星期五”的月份。

2. 探究自公元1年以来，一年中最多会出现几次黑色星期五、最少会出现几次黑色星期五，是否可能某一年没有黑色星期五？

分析：若年份为n，首先计算前n-1年有多少天days：

```
leap = (n-1)/4 -(n-1)/100 + (n-1)/400;  /* 闰年数 */
days = (n-1)*365+leap;
```

用一个数组存储每月前一个月的天数：

```
int month[12]={0,31,28,31,30,31,30, 31,31, 30,31, 30};
```

若当年为闰年，则二月份多一天：

```
if (n%4==0 && n%100 !=0 || n%400==0)
        month[2] +=1;
```

设一个计数器mm=0统计黑色星期五个数，循环计算一年12个月每月的13号是不是黑色星期五，并用数组c存月份：

```
for (i=1; i<=12; i++)
{   days += month[i-1];
    if ((days +13) % 7 == 5)
            c[mm++] = i;
}
```

程序为：

```c
#include  <stdio.h>
int fun(int n, int c[ ])
{  int i, mm=0, leap, days;
   int month[12]={0,31,28,31,30,31,30,31,31,30,31,30};
   leap = (n-1)/4 -(n-1)/100 + (n-1)/400;
   days = 365*(n-1) + leap;
   if (n%4==0 && n%100 !=0 || n%400==0)
        month[2] +=1;
   for (i=1; i<=12; i++)
   {   days += month[i-1];
       if ((days +13) % 7 == 5)
            c[mm++] = i;
   }
   return mm;
}
main( )
{   int n,c[12], m=0, i;
    printf("Please input a year:");
    scanf("%d", &n);
    m = fun(n, c);
    printf("%d年黑色星期五的个数是:%d\n",n, m);
    printf("黑色星期五的月份是：\n");
    for (i=0; i<m; i++)
        printf("%d\n", c[i]);
}
```

```
Please input a year:2015
2015年黑色星期五的个数是:3
黑色星期五的月份是：
2
3
11
请按任意键继续. . .

Please input a year:2017
2017年黑色星期五的个数是:2
黑色星期五的月份是：
1
10
请按任意键继续. . .

Please input a year:2019
2019年黑色星期五的个数是:2
黑色星期五的月份是：
9
12
请按任意键继续. . .

Please input a year:2020
2020年黑色星期五的个数是:2
黑色星期五的月份是：
3
11
请按任意键继续. . .

Please input a year:2021
2021年黑色星期五的个数是:1
黑色星期五的月份是：
8
请按任意键继续. . .

Please input a year:2022
2022年黑色星期五的个数是:1
黑色星期五的月份是：
5
请按任意键继续. . .
```

设数组int dd[13]={0};

利用前面的fun函数，循环计算自公元1年到2022年每年出现黑色星期五个数m; 并做统计：

```
dd[m]++;
```

最后dd[i]就是出现i个黑色星期五的个数。i=0,1,2,…,12

程序为：

```
#include  <stdio.h>
int fun(int n, int c[ ])
{  int i, mm=0, leap, days;
   int month[12]={0,31,28,31,30,31,30,31,31,30,31,30};
   leap = (n-1)/4 -(n-1)/100 + (n-1)/400;
   days = 365*(n-1) + leap;
   if (n%4==0 && n%100 !=0 || n%400==0)
       month[2] +=1;
   for (i=1; i<=12; i++)
   {   days += month[i-1];
       if ((days + 13) % 7 == 5)
       {  c[i-1]++;  mm++; }
   }
   return mm;
}
main( )
{   int  cc[12]={0}, dd[13]={0},m, i;
    for (i=1; i<=2022; i++)
    {    m=fun(i, cc);
         dd[m]++;
    }
    for (i=0; i<=12; i++)
        printf("%d\n", dd[i]);
    for (i=0; i<12; i++)
        printf("%d,", cc[i]);
}
```

运行结果：
0
864
860
298
0
0
0
0
0
0
0
0
0
293,288,293,294,294,288,294,283,288,282,293,288,请按任意键继续. . .

结论：每年最多3个，最少1个黑色星期五。

各月份黑色星期五出现的次数：
293,288,293,294,294,288,294,283,288,282,293,288

第8次作业（第9章 练习9）

p.223 习题 1, 4, 5, 10, 12

思考探究题：编程实现从键盘上读入两个超长正整数（可能几百上千位），计算它们的乘积并输出。