

计算机程序设计基础(1)

--- C语言程序设计(7)

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
2022.10.

第7章 循环结构

7.1 当型循环与直到型循环

7.2 while语句

7.3 do-while语句

7.4 对键盘输入的讨论

7.5 for语句

7.6 循环的嵌套与其他有关语句

7.6.1 循环的嵌套

7.6.2 break 语句

7.6.3 continue 语句

7.7 算法举例

1. 列举法

2. 试探法

3. 密码问题

4. 方程求根

5. 级数求和

6. 四叶玫瑰数

7. 整数分解

7.1 当型循环与直到型循环

● 当型循环结构

先判断条件，当条件满足(即逻辑表达式的值为真)时，执行循环体中所包含的操作，当循环体执行完后，将再次判断条件，直到条件不满足(即逻辑表达式的值为假)为止，从而退出循环结构。

如果在开始执行这个循环结构时条件就不满足，则当型循环结构中的循环体一次也不执行。

当条件满足

循环体

● 直到型循环结构

首先执行循环体，然后判断条件（即计算逻辑表达式），如果条件满足（即逻辑表达式值为真），则退出循环结构；如果条件不满足（即逻辑表达式值为假），则继续执行循环体。

由于首先执行循环体，然后再判断条件，因此，其循环体至少要执行一次。这是直到型循环结构与当型循环结构最明显的区别。

循环体

直到条件满足

7.2 while语句

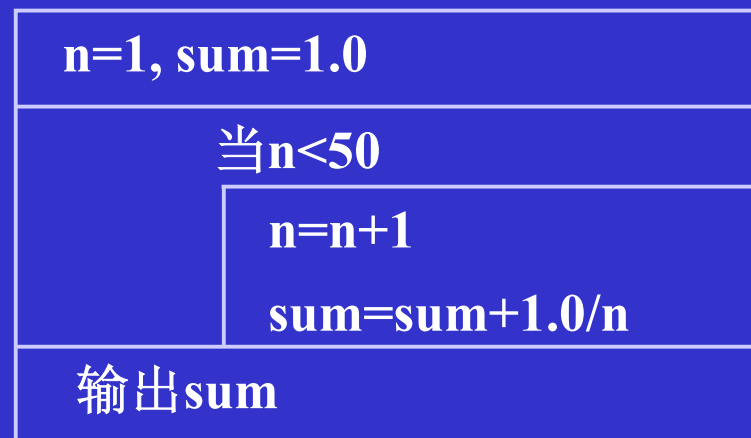
while (表达式) 循环体语句

首先计算表达式的值，当表达式值 $\neq 0$ （为真）时，执行循环体语句，执行完循环体中所有的语句后，再次计算表达式的值。只有当表达式值 $= 0$ （为假）时才退出循环体，执行循环结构后面的语句。

【例7.1】计算并输出下列级数和：

$$\text{sum} = 1 + 1/2 + 1/3 + 1/4 + \dots + 1/50$$

● 在用while语句构成循环结构的时候，在循环体内一定要有改变“表达式”（循环条件）值的语句，否则将造成死循环（即表达式值恒为1）。



相应的C程序如下：

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

```
main( )
```

```
{ int n;
```

```
  double sum;
```

```
  sum=1.0; n=1;
```

```
  while(n<50)
```

```
  { n++;
```

```
    sum += 1.0/n;
```

```
  }
```

```
  printf("sum=%lf\n", sum);
```

```
}
```

sum=0.0; n=1;

```
while(n<=50)
{ sum += 1.0/n;
  n++;
}
```

sum=0.0; n=0;

运行结果是：

sum=4.499205

注意：累加变量、循环变量的边界值！

需要注意的是，在这个程序中，语句

sum += 1.0/n;

也可以写成

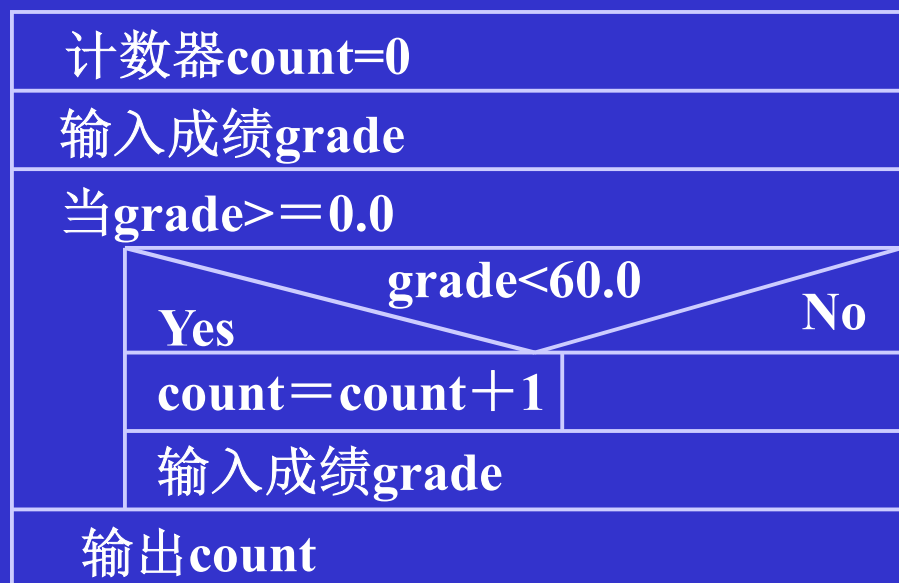
sum += 1/(double)n; 或 sum += (double)1/n;

但不能写成

sum += 1/n; 或 sum += (double)(1/n);

因为1和n都是整型量，当n>1时，1/n的值总为0。

【例7.2】从键盘输入若干学生成绩，并对成绩不及格（60分以下）的学生人数进行计数，直到输入的成绩为负为止，最后输出成绩不及格的学生人数。



```
#include <stdio.h>
main( )
{ int count;
  float grade;
  count=0;
  scanf("%f", &grade);
  while (grade>=0.0)
  { if (grade<60.0) count++;
    scanf("%f", &grade);
  }
  printf("count=%d\n",count);
}
```

注意：应先读入一个grade，然后才能开始while循环的条件判断！

7.3 do-while语句

do 循环体语句 **while**(表达式);

首先执行循环体语句，然后判断表达式值，若表达式值 $\neq 0$ （为真），则再次执行循环体语句，如此循环，直到表达式值 $= 0$ （为假）为止。

例7.1中的问题也可以用do-while语句来解决，其C程序：

sum=0.0; n=1;

sum+= 1.0/n; n++;

while(n<=50);

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

```
main( )
```

```
{ int n;
```

```
double sum;
```

```
sum=1.0; n=1;
```

```
do {
```

```
n++; sum+= 1.0/n;
```

```
} while(n<50);
```

```
printf("sum=%lf\n", sum);
```

```
}
```

sum=0.0;
n=0;

【例7.3】 计算并输出下列级数和： $\text{sum}=1-1/3+1/5-\dots+(-1)^k/(2k+1)$
直到某项的绝对值小于 10^{-4} 为止。

相应的C程序如下：

```
#include <stdio.h>
main()
{ int k, f;
  double sum, d;
  sum = 1.0; k = 0; f = 1;
  do
  { k++;
    f = -f;
    d=1.0/(2*k+1);
    sum = sum+f*d;
  } while(d >= 1.0e-4);
  printf("sum=%lf\n", sum);
}
```

sum=1.0, k=0, f=1	
	k=k+1, f= -f d=1.0/(2*k+1) sum=sum+f*d
直到d<10 ⁻⁴	
输出sum值	

其中f是一个开关量，值只取{1, -1}，
用于改变每一项的符号。

程序的运行结果为：

sum=0.785448

注意：这里并没有写为
fabs(d)，因为d始终是正数

● 用while语句与do-while语句所实现的两种循环结构的区别与联系:

● 在用while语句实现的循环结构中, 其循环体可以一次也不执行(即执行while循环结构一开始, 其条件就不满足)。而在用do-while语句实现的循环结构中, 其循环体至少要执行一次, 这是因为条件的判断是在执行循环体之后。

因此, 在有些问题中, 如果其重复的操作(即循环体)有可能一次也不执行(即开始时条件就不满足), 则要用while语句来处理, 一般不用do-while语句来处理。

【例7.4】下列C程序的功能是计算并输出n! (阶乘) 值, 其中n从键盘输入。

```
#include <stdio.h>
main()
{ int n, k;
  double s;
  printf("input n :");
  scanf("%d", &n);
  k=1; s=1.0;
  while (k<n)
  { k++; s=s*k; }
  printf("n!=%f\n", s);
}
```

while

```
#include <stdio.h>
main()
{ int n, k;
  double s;
  printf("input n :");
  scanf("%d", &n);
  k=1; s=1.0;
  do { k++; s=s*k;
    } while (k<n);
  printf("n!=%f\n", s);
}
```

do-while

当n为1时，do-while结果是错误的！把k的初值改为0才与while等价

● 不管是用while语句还是用do-while语句实现循环结构，在循环体内部必须要有能改变条件(即逻辑表达式值)的语句，否则将造成死循环。例如，如果将例7.1中的C程序改成如下：

```
#include <stdio.h>
main()
{ int n;
  double sum;
  sum=1.0; n=1;
  while(n<50) sum+=1.0/n;
  printf("sum=%lf\n", sum);
}
```

即在循环体中缺少了语句“n=n+1;”，则将造成死循环，因为，此时n的值没有改变，保持原来赋的值1，循环的条件“n<50”始终满足。

while(n++<50) sum+=1.0/n;

改为 while(n<50); { n++; sum+=1.0/n; } 为啥还是死循环？
因为while(n<50)之后多了一个分号，造成复合语句{ }与while无关！

● 有些问题既可以用while语句来处理，也可以用do-while语句来处理。例7.3这个问题也可以用while语句实现的循环结构来处理，其C程序如下：

```
#include <stdio.h>
main()
{ int k, f;
  double sum, d;
  sum=1.0; k=0; f=1; d=1.0;
  while(d>=1.0e-4)
  { k++; f = -f;
    d = 1.0/(2*k+1);
    sum += f*d;
  }
  printf("sum=%lf\n", sum);
}
```

sum=1.0, k=0, f=1	
d=1.0	
当 $d \geq 10^{-4}$	
	k=k+1, f=-f d=1.0/(2*k+1) sum=sum+f*d
输出sum值	

● 不管是用while语句还是用do-while语句实现循环结构，其循环体如果包含一个以上的语句，则应以复合语句形式出现。

7.4 对键盘输入的讨论

【例7.5】编写一个C程序实现如下功能：从键盘输入一个英文字母，如果输入的英文字母为'y'或'Y'，则输出"yes!"; 如果输入的英文字母为'n'或'N'，则输出"no!"。其C程序如下：

```
#include <stdio.h>
main()
{ char ch;
  printf("Please input(y/n)?");
    /*输入提示*/
  scanf("%c", &ch);
    /*输入一个字符*/
  if (ch=='y' || ch=='Y')
    printf("yes!\n");
  else printf("no!\n");
}
```

运行结果：

第1次 Please input(y/n)?y<回车>

yes!

第2次 Please input(y/n)?Y<回车>

yes!

第3次 Please input(y/n)?n<回车>

no!

第4次 Please input(y/n)?N<回车>

no!

第5次 Please input(y/n)?a<回车>

no!

这个显然不对！

为了使程序的输出结果与题意相符合，在程序中还需要增加判断输入合理性的环节，如果输入的字母既不是'y'或'Y'，也不是'n'或'N'，则应重新输入。程序修改为：

```
#include <stdio.h>
main( )
{ char ch;
  do
  { printf("Please input(y/n)?"); /*输入提示*/
    scanf("%c",&ch); /*输入一个字符*/
  } while( ch!='y' && ch!='n' && ch!='Y' && ch!='N');
  if (ch=='y' || ch=='Y') printf("yes!\n");
  else printf("no!\n");
}
```

运行结果（带下划线的为键盘输入）：

Please input(y/n)?a<回车>

Please input(y/n)?Please input(y/n)?ay<回车>

Please input(y/n)?yes!

如果当前的输入函数读不完从键盘输入的数据，即输入缓冲区中的数据还未取完，则将留给下一个输入函数使用。因此需要在读取输入缓冲区中的一个字符后，立即“吃”掉输入缓冲区中剩下的所有字符。程序修改为：

```
#include <stdio.h>
main( )
{ char ch;
  do { printf("Please input(y/n)?");
        scanf("%c",&ch);
        while(getchar( )!='\n'); /* 读空输入缓冲区 */
    } while(ch!='y' && ch!='n' && ch!='Y' && ch!='N');
    if (ch=='y' || ch=='Y') printf("yes!\n");
    else printf("no!\n");
}
```

运行结果（带下划线的为键盘输入）：

Please input(y/n)?asfdy<回车>

Please input(y/n)?y<回车>

yes!

Please input(y/n)?zxcvN<回车>

Please input(y/n)?Yabcde<回车>

yes!

程序最后修改为:

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

```
main( )
```

```
{ char ch,ch2;
```

```
do
```

```
{ printf("Please input(y/n)?");
```

```
scanf("%c", &ch);
```

```
scanf("%c", &ch2);
```

```
if ( ch2 != '\n')
```

```
while(getchar() != '\n'); /* 读空输入缓冲区 */
```

```
}while(ch2!='\n'||(ch!='y'&& ch!='n'&& ch!='Y'&& ch!='N'));
```

```
if (ch=='y' || ch=='Y')
```

```
printf("yes!\n");
```

```
else
```

```
printf("no!\n");
```

```
}
```

第1次运行结果:

Please input(y/n)?yasfd<回车>

Please input(y/n)?y<回车>

yes!

第2次运行结果:

Please input(y/n)?Nzxcv<回车>

Please input(y/n)?N<回车>

no!

注意: 增加一个ch2, 每次同时读入2个字符, 即使第一个字符是n或N或y或Y, 如果第二个字符不是回车符也是错误的, 重新输入。

利用**while**语句构成的循环

```
while(getchar() != '\n');
```

将输入缓冲区中的剩余数据全部读完，使输入缓冲区变为空。特别要指出的是，在这个循环结构中，循环体为空操作，它的功能只是通过字符输入函数**getchar()**不断地读取输入缓冲区中剩下的字符，直到读入<回车>（注：键盘上的<回车>键中包括了换行字符'\n'）为止。

关于清空输入缓冲区还有更方便简单的方法，我们会在第12章中讲到用 **fflush(stdin)** 清空标准输入缓冲区。

程序还可以修改为:

```
#include <stdio.h>
#include <conio.h>
main( )
{ char ch;
  do
  { printf("Please input(y/n)?");
    ch = _getche();
    putchar('\n');
  } while(ch!='y' && ch!='n' && ch!='Y' && ch!='N');
  if (ch=='y' || ch=='Y')
    printf("yes!\n");
  else
    printf("no!\n");
}
```

运行结果:

Please input(y/n)?a

Please input(y/n)?s

Please input(y/n)?b

Please input(y/n)?y

yes!

直接读键盘缓冲区，只要按下任何键程序就会相应给出相应结果。

7.5 for语句

提供循环的初始值

提供循环的条件

for(<表达式1>; <表达式2>; <表达式3>) <循环体>

【例7.6】计算并输出1到19之间各自然数的阶乘值。

改变循环的条件

等价于

```
#include <stdio.h>
main()
{ int n;
  double p;
  p=1.0;
  for (n=1; n<=19; n++)
  { p=p*n;
    printf("%d!=%f\n", n, p);
  }
}
```

<表达式1>;
while(<表达式2>)
{ <循环体>
 <表达式3>;
}

注意：由于double型有效位数的限制，这里循环求n!只能到19，19以后的结果是近似值。

运行结果为:

1!=1.000000

2!=2.000000

3!=6.000000

4!=24.000000

5!=120.000000

6!=720.000000

7!=5040.000000

8!=40320.000000

9!=362880.000000

10!=3628800.000000

11!=39916800.000000

12!=479001600.000000

13!=6227020800.000000

14!=87178291200.000000

15!=1307674368000.000000

16!=20922789888000.000000

17!=355687428096000.000000

18!=6402373705728000.000000

19!=121645100408832000.000000

也可以把p定义为超长整数:

```
#include <stdio.h>
main( )
{ int n;
  long long p;
  p=1;
  for (n=1; n<=19; n++)
  { p=p*n;
    printf("%d! =%lld\n", n, p);
  }
}
```

运行结果为:

```
1! =1
2! =2
3! =6
4! =24
5! =120
6! =720
7! =5040
8! =40320
9! =362880
10! =3628800
11! =39916800
12! =479001600
13! =6227020800
14! =87178291200
15! =1307674368000
16! =20922789888000
17! =355687428096000
18! =6402373705728000
19! =121645100408832000
请按任意键继续...
```

for 语句的一般格式是:

```
for (<表达式1>; <表达式2>; <表达式3>)  
    <循环体>
```

其中:

- (1) <表达式1>为赋初值, 只在进入时执行一次。
- (2) <表达式2>为条件表达式, 为真则执行一次<循环体>。
- (3) <表达式3>为后处理, 向循环结束的方向前进一步。

for循环的执行过程是:

- (1) 计算<表达式1>;
- (2) 计算<表达式2>; 若其值为真(非0), 转步骤(3);
 若其值为0, 转步骤(5);
- (3) 执行一次<循环体>;
- (4) 计算<表达式3>, 转步骤(2);
- (5) 结束循环。

- 在for语句中，<表达式1>与<表达式2>均可省略，但其中的两个“;”不能省略。

例如

下列四种循环形式是等价的：

① for (i=1; i<=100; i=i+1) <循环体>

② i=1;

for (; i<=100; i=i+1) <循环体>

③ i=1;

for (; i<=100;) {<循环体>; i=i+1; }

④ i=1;

while (i<=100) {<循环体>; i=i+1; }

- 在for语句中如果没有<表达式2>，则将构成死循环，除非<循环体>中另有出口强行终止循环。

例如

下列两个循环都是死循环：

for (<表达式1>; <表达式3>) <循环体>

for (; ;) <循环体>

注意：

- (1) 使用**for**语句一般解决事先知道循环的起始点和终止点及其循环次数的问题。
- (2) **for**语句中的任何部分可省略，但 ‘;’不能省。

for (; ;)省略的条件为恒真，构成死循环。

```
unsigned long k=0;
while(1)
{ k++;
  if (k>=0xFFFFFFFF)
    break;
}
```

```
unsigned long k=0;
for( ; ; )
{ k++;
  if (k>=0xFFFFFFFF)
    break;
}
```

你认为哪个程序执行效率更高？

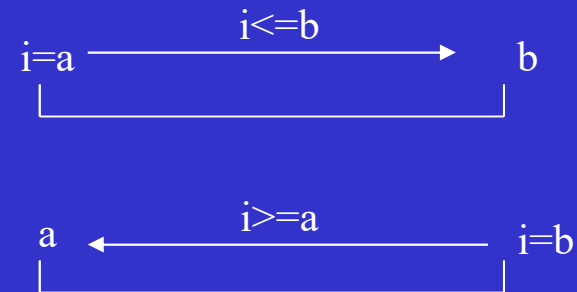
据说是**for**更高

- **for**循环本质上也是当型循环结构，只不过它对于事先可以确定循环次数的问题特别方便。

常用的**for**循环形式有以下两种：

for (i=a; i<=b; i=i+步长) 循环体

for (i=b; i>=a; i=i-步长) 循环体



- 在**for**循环中，循环体可以是复合语句，即用一对花括号{ }括起来的语句组。

下面是用for循环求解例7.1中问题的C程序：

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

```
main()
```

```
{ int n;
```

```
double sum;
```

```
sum=1.0;
```

```
for (n=2; n<=50; n++)
```

```
sum += 1.0/n;
```

```
printf("sum=%lf\n", sum);
```

```
}
```

sum=0.0;

for (n=1; n<=50; n++)

可以看到用for循环解决这个问题，写出的程序更清晰明了。

下面是用for循环来实现计算并输出n!(阶乘)值的C程序:

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

```
main()
```

```
{ int n, k;
```

```
    double s;
```

```
    printf("input n :");
```

```
    scanf("%d", &n);
```

```
    s=1.0;
```

```
    for (k=1; k<=n; k=k+1)
```

```
        s=s*k;
```

```
    printf("n!=%f\n", s);
```

```
}
```

注意: s=0.0; 会导致错误!

7.6 循环的嵌套与其他有关语句

● 循环的嵌套是指一个循环体内又包含了另一个完整的循环结构

7.6.1 循环的嵌套

【例7.7】 在马克思数学手稿中有这样一段话：有30个人，其中有男人、女人和小孩，在一家小饭馆里共花了50先令；每个男人花3先令，每个女人花2先令，每个小孩花1先令。问男人、女人和小孩各有多少？

设男人、女人和小孩数分别是p，q，r，则：

$$\begin{cases} p+q+r=30 \\ 3p+2q+r=50 \end{cases}$$

这是典型的不定整数方程组，一般会有多个解。

如何解此类问题？穷举！在可取值范围内逐个检验是否满足。

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

```
main( )
```

```
{ int  p, q, r;
```

```
  for (p=0; p<=30; p++)
```

```
    for (q=0; q<=30; q++)
```

```
      for (r=0; r<=30; r++)
```

```
        if (p+q+r == 30 && 3*p+2*q+r == 50)
```

```
          printf("%5d%5d%5d\n", p, q, r);
```

```
}
```

循环次数: $31*31*31=29791$

此程序能否优化?

运行结果:

0 20 10

1 18 11

2 16 12

3 14 13

4 12 14

5 10 15

6 8 16

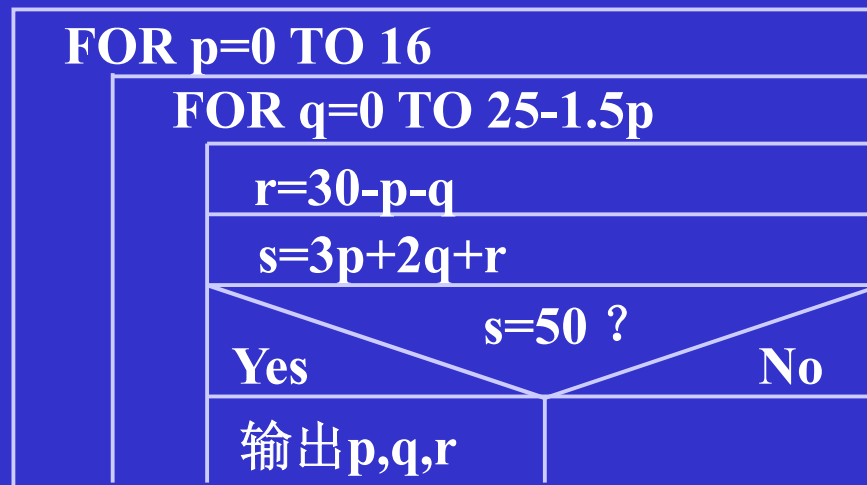
7 6 17

8 4 18

9 2 19

10 0 20

● 考虑到每个男人化3先令，50先令钱最多供16个男人吃饭；每个女人化2先令，50先令钱最多供25个女人吃饭。由于在考虑女人人数时已经有了p个男人，而1个男人所化的钱数相当于1.5个女人所化的钱，因此，在这种情况下，女人最多有 $25-1.5p$ 个。又由于已经考虑了p个男人和q个女人，因此，小孩只能有 $30-p-q$ 个，不必再循环找小孩人数。由此可以画出确定吃饭人数的流程图：



```
#include <stdio.h>
main()
{ int p, q, r, s;
  for (p=0; p<=16; p++)
    for (q=0; q<=25-1.5*p; q++)
    { r=30-p-q;
      s=3*p+2*q+r;
      if (s==50)
        printf("%5d%5d%5d\n",p,q,r);
    }
}
```

循环次数应小于 $17*25=425$
实际循环次数为: 234

类似的问题有“百钱百鸡”问题：

我国古代数学家张丘建在《算经》一书中提出了
“百钱百鸡”：鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一。百钱买百鸡，问鸡翁、鸡母、鸡雏各几何？

“百钱买百鸡”：公鸡每只5钱，母鸡每只3钱，
小鸡3只1钱。

设公鸡数为 x ，母鸡数为 y ，小鸡数为 z ，则可以得到
下面的整数不定方程组：

$$x + y + z = 100$$

$$5 * x + 3 * y + z / 3 = 100$$

大家可以自己试着编写一下这个程序，并验证结果是否是右边的结果，并考虑如何优化。

0	25	75
4	18	78
8	11	81
12	4	84


【例7.8】顺序输出3~100之间的所有素数。

```
#include <stdio.h>
#include <math.h>
main()
```

```
{ int j=0, n, k, i, flag;
  printf("\n");
  for (n=3; n<100; n=n+2)
  { k=(int)sqrt((double)n);
    i=2; flag=0;
    /*每次都需要重新赋值i和flag为2和0*/
    while ((i<=k)&&(flag==0))
    { if (n%i==0) flag=1;
      i=i+1;
    }
  }
```

用2到 \sqrt{N} 之间的所有整数K去除N,若所有的K均除不尽N,则N为素数,否则N不是素数。

```
    if (flag==0)
    { j=j+1;
      printf("%d ", n);
      if (j%10==0)
        printf("\n");
      /*每行打印10个*/
    }
  }
  printf("\n");
}
```



运行结果如下：

3 5 7 11 13 17 19 23 29 31

37 41 43 47 53 59 61 67 71 73

79 83 89 97

- 在这个例子中，for循环结构中嵌套了一个while循环结构和一个选择结构。
- 为了判断一个数是不是素数，引入了一个标记量flag，初值为0，循环中若n能被任意一个数整除，则置为1。循环结束后，判断flag是否为0，是则说明n不能被任何数整除是素数，打印n。
- 注意：对每个n，i都要重新赋值为2，flag都要重新赋值为0。

7.6.2 break 语句

功能

- 1) 跳出 switch 结构;
- 2) 退出当前循环结构,包括while、do...while和for循环结构

【例7.9】 找出3位数中最大的5个素数。

```
#include <stdio.h>
#include <math.h>
main()
{ int j=0, n, k, i, flag;
  for (n=999; n>=101; n=n-2)
  { k= (int) sqrt((double)n);
    i=2; flag=0;
    while ((i<=k)&&(flag==0))
    { if (n%i==0) flag=1;
      i=i+1;
    }
  }
```

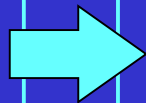
```
    if (flag==0)
    { j=j+1; printf("%d ", n); }
    /*对素数个数计数并输出该素数*/
    if (j==5) break;
    /*若已求出五个素数,退出循环*/
  }
  printf("\n");
}
```

运行结果为: 997 991 983 977 971

注意

- 在循环结构中的**break**语句只是退出当前循环结构。
- 循环结构中的**break**语句是一个非正常出口，理论上是不符合结构化程序设计原则的，建议在循环结构中尽量不用**break**语句退出，而利用正常的条件判断来退出。

```
for (k=1; k<20; k++)  
{ ...  
    while (n<k)  
    { ...  
        printf("input c: ")  
        scanf("%c", &c);  
        if (c=='\n') break;  
        ...  
    }  
    ...  
}
```



```
for (k=1; k<20; k++)  
{ ...  
    flag=0;  
    while (n<k && flag==0)  
    { ...  
        printf("input c: ")  
        scanf("%c", &c);  
        if (c=='\n') flag=1;  
        else ...  
    }  
}
```

在上面的程序段中，利用一个标志变量flag 退出while循环

不用break语句改写的【例7.9】输出3位数的最大的5个素数的程序。

```
#include <stdio.h>
#include <math.h>
main( )
{ int j=0, n, k, i, flag;
  for (n=999; n>=101 && j<5; n=n-2)
  { k= (int) sqrt((double)n);
    i=2; flag=0;
    while ((i<=k)&&(flag==0))
    { if (n%i==0) flag=1;
      i=i+1;
    }
    if (flag==0)
    { j=j+1;
      printf("%d ", n);
    }
  }
  printf("\n");
}
```

或改写为:

```
#include <stdio.h>
#include <math.h>
main()
{ int j, n, k, i, flag;
  for (n=999, j=0; n>=101 && j<5; n=n-2)
  { k= (int) sqrt((double)n);
    i=2; flag=0; /* 每次循环, 都重新为i和flag初始化 */
    while ((i<=k)&&(flag==0))
    { if (n%i==0) flag=1;
      i=i+1;
    }
    if (flag==0)
    { j=j+1;
      printf("%d ", n);
    }
  }
  printf("\n");
}
```

7.6.3 continue语句

● **continue**的功能是结束本次循环的执行，但不退出循环结构

【例7.10】 输出100~200之间所有能被7或9整除的自然数。

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

```
main( )
```

```
{ int n;
```

```
  for (n=100; n<=200; n++)
```

```
  { if ( (n%7!=0)&&(n%9!=0) )
```

```
      continue; /*结束本次循环，继续进行下次循环*/
```

```
      printf("%d\n", n);
```

```
  }
```

```
}
```

● 利用**continue**语句可以在循环体的任何位置上结束本次循环而开始下一次的循环，破坏了循环结构的正常执行顺序，因此，严格来说，它是一个不符合结构化原则的语句。

在7.4节中曾提到，一个for循环结构

```
for(<表达式1>; <表达式2>; <表达式3>)
    <循环体>
```

等价于下列的当型循环结构：

```
    <表达式1>;
    while(<表达式2>)
    { <循环体>
      <表达式3>;
    }
```

这种等价关系只是在循环结构符合结构化原则的前提下才成立。如果在循环体内包含有**continue**语句，这个等价关系就不成立了，并且还会导致死循环。例如，对于例7.10中的程序，如果按上面等价的方法，将其中的**for**循环用**while**循环来表示，即将程序改写成如下形式：


```
#include <stdio.h>
main()
{   int n;
    n=100;
    while(n<=200)
    {   if ((n%7!=0)&&(n%9!=0))
        continue; /*结束本次循环，继续进行下次循环*/
        printf("%d\n", n);
        n=n+1;
    }
}
```

● 在执行该程序时，如果遇到一个既不能被7整除也不能被9整除的数后，就在语句

if (n%7!=0)&&(n%9!=0)) continue;

中的**continue**语句处结束本次循环，但在执行下次循环时，由于循环控制变量**n**没有加1，**n**保持原来的值不变，即该**n**的值仍然既不能被7整除也不能被9整除，从而还在原来的地方结束本次循环。依此类推，程序将无限制地执行下去。 死循环！

但如果程序中不用**continue**语句，就可以将**for**循环等价地用**while**循环表示。例如，例7.10中的不用**continue**语句的**for**循环程序，可以等价地改写成**while**循环程序如下：

```
#include <stdio.h>
main( )
{  int n;
   n=100;
   while(n<=200)
   {  if ((n%7==0)||(n%9==0))
       printf("%d\n", n);
       n=n+1;
   }
}
```

7.7 算法举例

1. 列举法

列举法又称穷举法，基本思想是，根据提出的问题，列举所有可能的情况，并用问题中给定的条件检验哪些是需要的，哪些是不需要的。

列举法常用于解决“是否存在”或“有多少种可能”等类型的问题，例如求解不定方程的问题。

【例7.11】 某参观团按以下限制条件从A, B, C, D, E五个地方中选定若干参观点:

- 1) 如果去A地, 则必须去B地;
- 2) D和E两地中只能去一地;
- 3) B和C两地中只能去一地;
- 4) C和D两地要么都去, 要么都不去;
- 5) 如果去E地, 则必须去A和D地。问该参观团能去哪几个地方?

● 用a、b、c、d、e五个整型变量, 分别表示A、B、C、D、E是否去的状态。若变量值为1, 则表示去该地; 若变量值为0, 则表示不去该地。

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

```
main( )
```

```
{ int a, b, c, d, e;
```

```
  for (a=0; a<=1; a++)
```

```
    for (b=0; b<=1; b++)
```

```
      for (c=0; c<=1; c++)
```

```
        for (d=0; d<=1; d++)
```

```
          for (e=0; e<=1; e++)
```

```
            if ((a&&b || !a) && d+e==1 && b+c==1&&  
                (c+d==2 || c+d==0) && (e&& a+d==2 || !e))
```

```
              { printf("will %s go to A.\n", a? "": "not");
```

```
                printf("will %s go to B.\n", b? "": "not");
```

```
                printf("will %s go to C.\n", c? "": "not");
```

```
                printf("will %s go to D.\n", d? "": "not");
```

```
                printf("will %s go to E.\n", e? "": "not");
```

```
              }
```

```
            }
```

运行结果为:

will not go to A. (不去)

will not go to B. (不去)

will go to C. (去)

will go to D. (去)

will not go to E. (不去)

2. 试探法

在前面所讨论的列举算法中，一般总是知道列举量，其列举的情况总是有限的。而在另外一些问题中，可能其列举量事先并不知道，只能从初始情况开始，往后逐步进行试探，直到满足给定的条件为止。这就是逐步试探的方法。
简称：试探法

【例7.12】 某幼儿园按如下方法依次给A、B、C、D、E五个小孩发苹果。将全部苹果的一半再加二分之一一个苹果发给第一个小孩；将剩下苹果的三分之一再加三分之一一个苹果发给第二个小孩；将剩下苹果的四分之一再加四分之一一个苹果发给第三个小孩；将剩下苹果的五分之一再加五分之一一个苹果发给第四个小孩；将最后剩下的11个苹果发给第五个小孩。每个小孩得到的苹果数均为整数。

编制一个C程序，确定原来共有多少个苹果？每个小孩各得到多少个苹果？

设x是总苹果数，则：

$a=(x+1)/2;$ /* 第一个小孩分到的苹果数 */

$b=(x-a+1)/3;$ /* 第二个小孩分到的苹果数 */

$c=(x-a-b+1)/4;$ /* 第三个小孩分到的苹果数 */

$d=(x-a-b-c+1)/5;$ /* 第四个小孩分到的苹果数 */

$e=x-a-b-c-d=11;$ /* 第五个小孩分到的苹果数 */

可以得到：

$$\frac{n+1}{k+1} = \frac{n}{k+1} + \frac{1}{k+1} \quad k=1,2,3,4$$



```
#include <stdio.h>
main( )
{ int n, flag, k, x, a, b, c, d, e; n=11; /*试探初值*/
  flag=1;
  while(flag) /*进行试探*/
  { x=n; /* 保存当前试探值 */
    flag=0; /* 清标志量值 */
    for (k=1; k<=4 && flag==0; k++) /*模拟四次发放过程*/
      if ((n+1)%(k+1)==0) /*该小孩得到的是整数个苹果*/
        n=n-(n+1)/(k+1); /*计算余下的苹果数*/
      else flag=1; /*该小孩得到的不是整数个苹果，置标志值*/
    if (flag==0 && n!=11)
      flag=1; /*每次分配都得到整数个苹果,且最后剩下11个苹果*/
    n=x+1; /*下一次的试探值*/
  }
}
```




```
printf("Total number of apple=%d\n", x);
```

```
    /*输出总的苹果数*/
```

```
    a=(x+1)/2;        /* 第一个小孩分到的苹果数 */
```

```
    b=(x-a+1)/3;      /* 第二个小孩分到的苹果数 */
```

```
    c=(x-a-b+1)/4;    /* 第三个小孩分到的苹果数 */
```

```
    d=(x-a-b-c+1)/5; /* 第四个小孩分到的苹果数 */
```

```
    e=x-a-b-c-d;      /* 第五个小孩分到的苹果数 */
```

```
    printf("A=%d\n", a);
```

```
    printf("B=%d\n", b);
```

```
    printf("C=%d\n", c);
```

```
    printf("D=%d\n", d);
```

```
    printf("E=%d\n", e);
```

```
}
```

请思考：能否通过改变试探初值与步长，以便减少循环次数。

程序的运行结果如下：

Total number of apple=59

A=30

B=10

C=5

D=3

E=11

3. 密码问题

- 在报文通信中,为使报文保密,发报人往往要按一定规律将其加密,收报人再按约定的规律将其解密(即将其译回原文)。

- 最简单的加密方法是,将报文中的每一个英文字母转换为其后的第 k 个字母,而非英文字母不变。被称为凯撒密码。

例如,当 $k=5$ 时,字母a转换为f, B转换为G等。由此可以看出,这种转换是很方便的,只需将该字母的ASCII码加上5(k 的值)即可。

- 在转换过程中,如果某大写字母其后的第 k 个字母已经超出大写字母Z,或某小写字母其后的第 k 个字母已经超出小写字母z,则将循环到字母表的开始。

例如,大写字母V转换为A,大写字母Z转换为E,小写字母v转换为a,小写字母z转换为e等。

- 由加密的过程不难推出解密的方法。

下面举例说明上述这种加密和解密的方法。

【例7.13】 从键盘输入一行字符，将其中的英文字母进行加密输出（非英文字母不用加密）。

```
#include <stdio.h>
main()
{ char c;  int k;
  printf("input k:"); /*输入k的提示*/
  scanf("%d", &k);  /*输入k值*/
  getchar(); /*吃掉上次输入的回车符*/
  c=getchar(); /*输入一行字符，并读取第1个字符*/
  while(c != '\n') /*一行字符未读完*/
  { if ((c>='a'&&c<='z') || (c>='A'&&c<='Z')) /*对英文字母加密*/
    { c=c+k;
      if (c>'z' || (c>'Z' && c <= 'Z'+k))
        c=c-26;
    }
    printf("%c", c); /*依次输出加密后的字符*/
    c=getchar(); /*依次读取下一个字符*/
  }
}
```

运行结果：

input k:5

are you ready? （原文）

fwj dtz wjfid? （加密后）

【例7.14】 从键盘输入一行经加密过的字符，将其中的英文字母进行解密输出（非英文字母不用解密）。

```
#include <stdio.h>
main()
{ char c; int k;
  printf("input k:"); /*输入k的提示*/
  scanf("%d", &k); /*输入k值*/
  getchar(); /*吃掉上次输入的回车符*/
  c=getchar(); /*输入一行字符，并读取第1个字符*/
  while(c != '\n') /*一行字符未读完*/
  { if ((c>='a' && c<='z') || (c>='A' && c<='Z')) /*对英文字母解密*/
    { c=c-k;
      if ((c<'a' && c>='a'-k) || c<'A')
        c=c+26;
    }
    printf("%c", c); /*依次输出解密后的字符*/
    c=getchar(); /*依次读取下一个字符*/
  }
}
```

运行结果：

input k:5

fwj dtz wjfid? （密码）

are you ready? （解密后）

4. 方程求根

● 对分法求方程实根

假设已知非线性方程 $f(x)=0$ 的左端函数 $f(x)$ 在区间 $[a,b]$ 上连续, 并满足: $f(a)f(b)<0$, 则该非线性方程在区间 $[a,b]$ 上至少有一个实根。

基本思想 逐步缩小有根的区间, 当这个区间长度减小到一定程度时, 就取这个区间的中点作为根的近似值。

对分法的要点可以描述如下：

- (1) 取有根区间的中点，即令 $x=(a+b)/2$ 。
- (2) 若 $f(x)=0$ ，则 x 即为根，过程结束。
- (3) 若 $f(a)f(x)<0$ ，则说明实根在区间 $[a,x]$ 内，令 $b=x$ ；
若 $f(b)f(x)<0$ ，则说明实根在区间 $[x,b]$ 内，令 $a=x$ 。
- (4) 若 $|a-b|<e$ （ e 为预先给定的精度要求），则过程结束， $(a+b)/2$ 即为根的近似值（已满足精度要求）；否则从(1)开始重复执行。

如果在区间 $[a,b]$ 内有多个实根，则单独利用对分法只能得到其中的一个实根。在实际应用中，可以将逐步扫描与对分法结合起来使用，以便尽量搜索出给定区间内的所有实根。这种方法的要点如下：

- 从区间左端点 $x=a$ 开始，以 h 为步长，逐步往后进行搜索。
- 对于在搜索过程中遇到的每一个子区间 $[x_k, x_{k+1}]$
(其中 $x_{k+1}=x_k+h$)，作如下处理：
 - 若 $f(x_k)=0$ ，则 x_k 为一个实根，且从 $x_k+h/2$ 开始往后再搜索；
 - 若 $f(x_{k+1})=0$ ，则 x_{k+1} 为一个实根，且从 $x_{k+1}+h/2$ 开始往后再搜索；
 - 若 $f(x_k)f(x_{k+1})>0$ ，则说明在当前子区间内无实根或 h 选得过大，放弃当前子区间，从 x_{k+1} 开始往后再搜索；
 - 若 $f(x_k)f(x_{k+1})<0$ ，则说明在当前子区间内有实根，此时利用对分法，直到求得一个实根为止，然后从 x_{k+1} 开始往后再搜索。
 - 在进行根的搜索过程中，要合理选择步长，尽量避免根的丢失。

$x_1 = a, y_1 = f(x_1), x_2 = x_1 + h, y_2 = f(x_2)$				
当 $x_1 \leq b$				
Yes		$y_1 * y_2 > 0$ No		
$x_1 = x_2$ $y_1 = y_2$ $x_2 = x_1 + h$ $y_2 = f(x_2)$		flag=0		
		当flag=0		
		$x = (x_1 + x_2) / 2$		
		Yes		No
		输出x $x_1 = x + h / 2$ $y_1 = F(x_1)$ $x_2 = x_1 + h$ $y_2 = f(x_2)$ flag=1		$y = f(x)$
		$y_1 * y < 0$		
		Yes	No	
		$x_2 = x$ $y_2 = y$	$x_1 = x$ $y_1 = y$	

【例7.15】 用对分法求方程 $f(x)=x^2-6x-1=0$
在区间 $[-10, 10]$ 上的实根，即 $a=-10$ ， $b=10$ 。
取扫描步长 $h=0.1$ ，精度要求 $\varepsilon=10^{-6}$ 。

相应的C程序如下：

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

```
#include <math.h>
```

```
#define F(x) ((x)*(x)-6.0*(x)-1.0)
```

```
main( )
```

```
{ int flag;
```

```
double a=-10.0, b=10.0, h=0.1, x1 , y1 , x2 , y2 , x, y;
```

```
x1 =a; y1 =F(x1);
```

```
x2 = x1 +h; y2 =F(x2);
```

```
while (x1 <= b)
{   if (y1*y2>0.0) /*子区间两端点函数值同号, 无根*/
    {x1 =x2; y1 =y2; x2 = x1 + h; y2 =F(x2); }
    else /*子区间两端点函数值异号,
           在该子区间内用对分法求实根*/
    { flag=0;
      while (flag==0)
      {   x=(x1 + x2)/2; /*取子区间中点*/
          if (fabs(x2 - x1)<0.000001) /*满足精度要求*/
          {   printf("x=%11.7f\n", x); /*输出实根值*/
              x1 =x + 0.5*h; y1 =F(x1); /*搜索下一子区间*/
              x2 = x1 + h; y2 =F(x2);
              flag=1;
          }
      }
```

```
else /*不满足精度要求, 继续对分*/  
{ y=F(x);  
  if (y1*y<0.0) { x2 =x; y2 =y; }  
  else { x1 =x; y1 =y; }  
}  
} /* while (flag == 0) */  
} /* if (y1*y2>0.0) else */  
} /* while (x1 <= b) */  
}
```

程序运行结果为:

x= -0.1622780

x= 6.1622770

● 迭代法求方程实根

设非线性方程为 $f(x)=0$ ，用迭代法求一个实根的基本方法如下：首先将方程 $f(x)=0$ 改写成便于迭代的格式 $x = \Phi(x)$ 然后初步估计方程实根的一个初值 x_0 ，作迭代 $X_{n+1} = \Phi(X_n)$ 。直到满足条件 $|X_{n+1} - X_n| < \varepsilon$ 或者迭代了足够多的次数还不满足这个条件为止。其中 ε 为事先给定的精度要求。

给定初值 x , 精度要求 ε	
输入最大迭代次数 M	
	$x_0 = x$
	$x = \Phi(x_0)$
	$M = M - 1$
直到 $M=0$ 或 $ x - x_0 < \varepsilon$	
Yes	No
输出 "FAIL"	输出 x

【例7.16】 求非线性方程 $x-1-\arctan x=0$ 的一个实根。取初值 $x_0=1.0$ ，精度要求 $\varepsilon=0.000001$ 。并改写成如下迭代格式：

$$x_{n+1} = 1 + \arctan x_n$$

```
#include <stdio.h>
#include <math.h>
main()
{ int m;
  double x=1.0, eps=0.000001, x0 ;
  printf("input m: ");
  scanf("%d", &m);
  /*输入最大迭代次数*/
  do
  { x0 =x; x=1.0+atan(x0); m=m-1;
  } while ((m!=0)&&(fabs(x- x0)>=eps));
  if (m==0) printf("FAIL!\n ");
  else printf("x=%11.7f\n", x);
}
```

最后要指出的是，如果给定的最大迭代次数已经很大，但还满足不了精度要求，此时有可能初值选得不合适，或者改写成的迭代格式本身就不收敛。在这种情况下，要重新取初值，或改变迭代格式再试一试。

运行结果：

input m: 20

x= 2.1322676

● 牛顿法求方程实根

设非线性方程为 $f(x)=0$ 在选取一个初值 x_0 后，牛顿迭代格式为 $X_{n+1}=X_n-f(X_n)/f'(X_n)$ 实际上牛顿迭代格式是一种特殊的简单迭代（但牛顿法收敛更快），相当于 $\Phi(X_n)=X_n-f(X_n)/f'(X_n)$ 上述迭代过程一直进行到满足条件 $|X_{n+1}-X_n|<\varepsilon$ 或者迭代了足够多的次数还不满足这个条件为止。其中 ε 为事先给定的精度要求。

给定初值 x ，精度要求 ε	
输入最大迭代次数 M	
$x_0 = x$	
$x = x_0 - f(x_0)/f'(x_0)$	
$M=M-1$	
直到 $M=0$ 或 $ x-x_0 <\varepsilon$	
<div style="display: flex; justify-content: space-between; align-items: center;"> Yes $M=0$ No </div>	
输出 "FAIL"	输出 x

【例7.17】 求非线性方程 $x-1-\cos x=0$ 的一个实根。取初值 $x_0=1.0$ ，精度要求 $\varepsilon=0.000001$ 。

$$f(x)=x-1-\cos x$$

$$f'(x)=1+\sin x$$

牛顿迭代公式为：

$$x_{n+1} = x_n - \frac{x_n - 1 - \cos x_n}{1 + \sin x_n}$$

运行结果：

input m: 10

x= 1.2834287

```
#include <stdio.h>
#include <math.h>
main()
{ int m;
  double x=1.0, eps=0.000001, x0;
  printf("input m: ");
  scanf("%d", &m);
  /*输入最大迭代次数*/
  do
  { x0=x;
    x=x0-(x0-1-cos(x0))/(1.0+sin(x0));
    m=m-1;
  } while ((m!=0)&&(fabs(x-x0)>=eps));
  if (m==0) printf("FAIL!\n ");
  else printf("x=%11.7f\n", x);
}
```

5. 级数求和

【例7.18】 计算下列级数和：

$$S(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

直到 $\frac{x^n}{n!} < 10^{-6}$ 。n和x从键盘读入。

● 如何自己推导出递推式？

分析： 设级数的第n项为： $T_n = \frac{x^n}{n!}$

则第n+1项： $T_{n+1} = \frac{x^{n+1}}{(n+1)!} = \frac{x^n}{n!} \times \frac{x}{n+1} = T_n \times \frac{x}{n+1}$

由此得到递推式： $T_{n+1} = T_n \times \frac{x}{n+1}$ ， $T_0 = 1$ 。

由此递推式编写的程序为：


```

#include <stdio.h>
#include <math.h>
main( )
{ double s=1.0, t=1.0, x=2.0; /* S=T0, T0=1 */
  int n=1;
  do {
    t = t * x / n;    /* Tn+1=Tn × x/(n+1) */
    s += t;
    n++;
  } while (t > 1e-6);
  printf("%12.6f %12.6f\n", s, exp(2.0)); /* ex */
}

```

运行结果是: 7.389057 7.389056

6. 四叶玫瑰数

【例题7.19】编写程序打印出所有的“四叶玫瑰数”以及“四叶玫瑰数”之和。

所谓“四叶玫瑰数”是指一个四位数，其各位数字的四次方之和等于该数，又称为：自幂数。

例如：1634是一个“四叶玫瑰数”，因为

$$1634 = 1^4 + 6^4 + 3^4 + 4^4$$

三位自幂数：水仙花数

四位自幂数：四叶玫瑰数

五位自幂数：五角星数

六位自幂数：六合数

七位自幂数：北斗七星数

八位自幂数：八仙数

九位自幂数：九九重阳数

十位自幂数：十全十美数

```
#include <stdio.h>
#define F(x) ((x)*(x)*(x)*(x))
main( )
{ int sum=0,n,a,b,c,d;
  for (n=1000; n<=9999; n++) /* 穷举法 */
  { a=n%10; /* 取整数的个位上的数 */
    b=n/10%10; /* 取整数的十位上的数 */
    c=n/100%10; /* 取整数的百位上的数 */
    d=n/1000; /* 取整数的千位上的数 */
    if (F(a)+F(b)+F(c)+F(d)== n)
    { printf("%d\n", n);
      sum += n;
    }
  }
  printf("sum=%d\n", sum);
}
```

运行结果:

1634

8208

9474

sum=19316

请按任意键继续...

另一种实现方法:

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

```
#define F(x) ((x)*(x)*(x)*(x))
```

```
main( )
```

```
{ int sum=0,n,a,b,c,d;
```

```
  for (a=1; a<=9; a++)
```

```
    for (b=0; b<=9; b++)
```

```
      for (c=0; c<=9; c++)
```

```
        for (d=0; d<=9; d++)
```

```
          { n= a*1000 + b*100 + c*10 + d; /* 将4位数合并为一个整数*/
```

```
            if (F(a)+F(b)+F(c)+F(d)== n)
```

```
              { printf("%d\n", n);
```

```
                sum += n;
```

```
              }
```

```
            }
```

```
          printf("sum=%d\n", sum);
```

```
}
```

运行结果:

1634

8208

9474

sum=19316

请按任意键继续...

7. 整数分解

【例7.20】一个正整数有可能被表示为m个连续正整数之和，例如：

$$15=1+2+3+4+5$$

$$15=4+5+6$$

$$15=7+8$$

请编写程序，根据输入的正整数n，如果n能表示成m个连续正整数之和，则打印出符合这种要求的所有连续正整数序列；否则给出提示信息，说明此数不能分解为连续正整数之和。

● 如何设计算法？分析：

从整数15的求和结果：

$$15=1+2+3+4+5$$

$$15=4+5+6$$

$$15=7+8$$

可以设想，根据输入的正整数n，（设置一个标志量flag=0）

- (1) 设一个变量b从1开始循环，每次加1，直到n/2，转(6)；
- (2) 让m=n， c=b；
- (3) 当m>=c时循环执行： m=m-c， c=c+1；
- (4) 若m==0；说明n能分解为从b到c的整数之和，打印；
（同时将flag加1，表示此数n可以分解）
- (5) 转(1)继续循环，求下一个可能的整数序列之和；
- (6) 若flag为0，说明n不能分解为连续正整数之和，打印不可分解的信息。

```

#include <stdio.h>
main( )
{   int j, b,c,m, n, flag=0;
    printf("Input a number:");
    scanf("%d", &n);
    for (b=1; b<=n/2; b++) /* 从1开始试，直到n的一半*/
    {
        m = n;
        c = b; /* 从b开始，求连续正整数求和的最后一个整数*/
        while (m>=c) /* 若m仍能分解，继续循环 */
        {
            m = m - c;    c++;
        }
        c--; /* 多加了1，减去 */
        if ( m == 0) /* 能分解为正整数连加之和 */
        {
            printf("%d=", n); /*打印连续正整数之和 */
            for (j=b; j<c; j++) /* 注意: c先不打印 */
                printf("%d+", j);
            printf("%d\n", j); /* 打印c及回车符，一个序列之和完毕 */
            flag++; /* 每打印连加一次，加1 */
        }
    }
    if (!flag) /* 若未打印过连加和 */
        printf("%d can't be splitted!\n", n);
}

```


Input a number: 99

$$99=4+5+6+7+8+9+10+11+12+13+14$$

$$99=7+8+9+10+11+12+13+14+15$$

$$99=14+15+16+17+18+19$$

$$99=32+33+34$$

$$99=49+50$$

请按任意键继续...

Input a number: 100

$$100=9+10+11+12+13+14+15+16$$

$$100=18+19+20+21+22$$

请按任意键继续...

Input a number: 128

128 can't be splitted!

请按任意键继续...

($n>1$ 时, 2^n 都不能分解!)

第6次作业

p.158-160 习题 1,3,4,7,10,11,12

思考探究题：黑色星期五问题

在有些国家，如果某个月的13号正好是星期五，这天就被称为“黑色星期五(**Black Friday**)”，人们就会觉得不太吉利。你能否编写程序，告知在某个年份中，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。并探究自公元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？