第3章 结构化程序开发

●要求：

1）熟悉C语言的程序结构框图；

2）熟练掌握C语言的二分支结构、多分支结构和循环结构；

3）熟练掌握结构化程序设计方法。

3.1 结构化程序设计

●结构化程序设计的基本思想是：

任何程序都由顺序结构、分支结构和循环结构这三种基本结构组成。

●结构化程序设计方法：

♦自顶向下，逐步细化

♦模块化设计

♦结构化编码

●结构化程序设计特点

♦一个程序单元由顺序、选择分支、循环这3种基本结构组成。3种基本结构经过反复组合、嵌套构成的程序，可以表示任何复杂的算法 。

♦一个大的程序由若干个不同功能的小模块组成。

♦每个小模块具有单一入口和单一出口。

♦程序中不能有无穷循环（死循环）。

♦程序中不能有在任何条件下都执行不到的语句（死语句）

♦用结构化思想设计出来的计算机程序，具有清晰的模块界面，因此，在书写程序时，我们应根据逻辑结构和层次深度的不同，采用缩进对齐的方式，将程序模块写在不同的位置，这样可以提高程序的可读性，有助于调试程序，找出程序的逻辑错误。

●C语言的语句概述：

♦C程序对数据的处理是通过"语句"的执行来实现的。

♦一条语句完成一项操作（或功能）。

♦一个为实现特定目的的程序应包含若干条语句。

●结构化C语言程序的组成

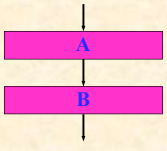
注意：1）在复合语句中的"内部数据描述语句"中定义的变量，是局部变量，仅在复合语句中有效。

2）复合语句结束的"}"之后，不需要分号。

3）复合语句可以出现在任何数据操作语句可以出现的地方。

3.2 顺序结构

顺序结构即按照语句书写顺序执行的程序结构。



顺序结构

例如：

```
#include <stdio.h>//p39_16
void main()
{    int x,y;
     scanf("%d,%d",&x,&y);
     printf("x < y = %d\n",x<y);
     printf("x > y = %d\n",x>y);
     printf("x == y = %d\n",x==y);
     printf("x != y = %d\n",x!=y);
     printf("x / y = %d\n",x/y);
}
```

3.3 选择结构

选择结构又称为分支结构，是指有条件地选择要执行的程序段。

3.3.1 if语句

if条件语句是我们最常用的一种分支语句。它符合人们通常的语言习惯和思维习惯。比如：

if（如果）绿灯亮是真，then(那么)本就可以通行。

else(否则)本辆要等待。

下面我们将介绍几种if语句的使用方法。

♦if(e)

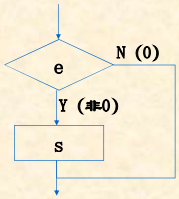
　　s;

e为测试表达式，s为执行语句。

如果e为真，则执行s；否则，结束if语句。

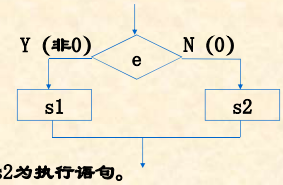
例如：

```
if(x>y)
   printf("max=%d",x);
```



♦ if-else语句

```
if(e)
      s1;
else
      s2;
```



其中，e为测试表达式，s1和s2为执行语句。

如果e为真，则执行s1；否则，执行s2。

if语句的无论条件表达式的值为何值，只能执行语句s1或语句s2中的一个，即二者中取一。

例如：

```
if(x>y)
      printf("max=%d",x);
else
      printf("max=%d",y);
```

注意：

如果s中的语句大于一条，则必须写成复合语句的形式。

【例3.2】输入三条边长，计算三角形面积。

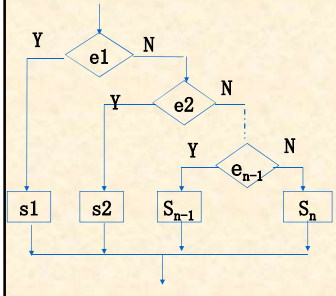
```
#include <stdio.h>
#include <math.h> /*使用数学函数库*/
void main()
{ float a, b, c, s, area;
  scanf("%f%f%f", &a, &b, &c);
  if ( a + b > c && b + c > a && c + a > b )
  { /*用一对花括号括起来的一条或多条语句来构成复合语句。无论包括多少条语句，复合语句从逻辑上讲，被看成是一条语句。*/
     s = ( a + b + c ) / 2.0;
     area = sqrt(s*(s-a)*(s-b)*(s-c));
  }
   else   area = 0.0;
   printf("area: %f\n", area);
}
```

也可以area放前面，这样可以省去else

♦ if … else if … else语句

实际应用中常常面对更多的选择，这时，将if … else扩展一下，就得到if … else if … else结构，其一般形式为：

对应的流程图：




```
if(e1)
    s1;
  else if (e2)
    s2;
...
    else
      sn;
```

其中，e1、e2、…为测试表达式，s1、s2、…、sn 为执行语句。

如果 e1 为真，则执行s1；

否则，如果 e2 为真，则执行s2；

……否则（以上条件都不成立），执行sn。

【例3.7】求一元二次方程的根

设一元二次方程为$ax^2+bx+c=0$，对输入的系数a、b、c，有以下几种情况需要考虑：

(1)a≠0：方程有两个根。

(2)a=0，b≠0：方程退化为bx+c=0，方程根为：

　　-c/b。

(3)a=0，b=0：方程或为同义反复(c=0)，或矛盾(c≠0)。

由以上分析得到以下程序结构：

```
{ (1)输入方程系数a, b, c;
  if ( a != 0.0 ) (2)求两个根;
  else if ( b != 0.0 )(3) 输出方程根 -c/b;
  else if ( c == 0 )  (4) 输出方程同义反复字样;
  else (5)输出方程矛盾字样;
}
```

分析1：对步骤(1)、(3)、(4)、(5)，用C语言描述非常简单。

分析2：对步骤(2)，根据判别式：

△=b^2-4ac＞0，有两个实根

△=b^2-4ac＜0，有两个复根

对于两个复根情况，可分别计算它们的实部和虚部。对于实根，也可根据上面给出的公式计算两个根。但是考虑到 b^2 >> 4ac 时，有一个根就非常接近零。数值计算中，两个非常接近的数执行减法求出的值的精度很低。为此先求出一个绝对值大的根root1。然后，利用根与系数的关系：

root1*root2 = c/a

即 root2=c/(a*root1)

求出root2。

```
#include <stdio.h>
#include <math.h>    /* 使用数学库函数 */
void main()
{
  double  a,b,c,delta,re,im,root1,root2;
  printf("输入方程系数a,b,c:");
  scanf("%lf,%lf,%lf",&a,&b,&c);
  if (a != 0.0){  /* 有两个根 */
    delta = b*b-4.0*a*c;  re = -b/(2.0*a);
    im = sqrt(fabs(delta))/(2.0*a);
    if (delta>= 0.0){/* 两个实根，先求绝对值大的根*/
     root1 = re + (b < 0.0 ? im : -im);
     root2 = c/(a*root1);
     printf("两实根是:%7.5f,%7.5f\n",root1,root2);
    }
```

```
  else
    printf("两复根是 %7.5f+%7.5fI, %7.5f-%7.5fI\n",re,
   fabs(im), re, fabs(im));
 }
 else  /* a = 0.0 */
 if (b != 0.0)
      printf("重根 %7.5f\n", -c/b);
 else if (!c)
       printf("方程同义反复.\n");
 else printf("方程矛盾。\n");
}
```

运行结果如下：

```
输入方程系数a,b,c:2,5,3
两实根是:-1.50000,-1.00000
输入方程系数a,b,c:0,0,3
方程矛盾。
输入方程系数a,b,c:1,2,3
两复根是 -1.00000+1.41421I, -1.00000-1.41421I
```

例　计算分段函数的值：

$$y=\begin{cases} x & x<0 \\ e^x & 0\le x<1 \\ \log_{10} x & 1\le x<10 \\ \sin x & x\ge 10 \end{cases}$$

```
#include <stdio.h>
#include <math.h>
void main( ){  float x,y;
      scanf("%f",&x);
      if (x<0)          y=x;
      else if (x < 1)   y=exp(x);
      else if (x<10)    y=log10(x);
      else              y=sin(x);
      printf("Y=%6.2f\n",y);
}
```

◆ if语句的嵌套

在一个if 语句中可以又出现另一个if语句，这称为if语句的嵌套或多重if语句：特别应注意else与if的对应关系

嵌套的一般形式

```
if (expr1)
    if (expr2)
        statement1
    else
        statement2
```

内嵌if

```
if (expr1)
    if (expr2)  statement1
    else        statement2
else
    if(expr3)   statement3
    else        statement4
```

内嵌if

```
if (expr1)
    statement1
else
    if(expr3)
        statement3
    else
        statement4
```

内嵌if

- if ~ else 配对原则：缺省{ }时，为避免不同理解，C语言约定else总是与它前面最接近的if对应。

```
if(...)
    if(...)
        if(...)
            else...
    else...
else...
```

示例：计算函数

$$y=\begin{cases}1 & x>0\\ 0 & x=0\\ -1 & x<0\end{cases}$$

```
void main()
{  float x,y;
   printf("input x=");
   scanf ( "%f" , &x ) ;
   if (x>=0)
      if (x>0)y = 1 ;
      else   y = 0 ;
   else   y = - 1 ;
   printf ( " y = % 4.0f\n " , y ) ;
}
```

对多重if，最容易犯的错误是if与else配对错误，例如，写成如下形式：

```
y = 0 ;
if (x>=0)
   if (x>0)y = 1 ;
else   y = - 1 ;
```

从缩排上可以看出，作者希望else是与if x>=0配对，但是C语言规定else总是与离它最近的上一个if配对，结果，上述算法完全违背了设计者的初衷。改进的办法是使用复合语句，将上述程序段改写如下：

修改：

```
y = 0 ;
if （x > = 0 ）
   { if （x > 0 ） y = 1 ; }
else
y = - 1 ;
```

实现if～else 正确配对方法：加{}

```
if (e1)
      if (e2)
            s1;
      else
            s2;
```

⇔

```
if (e1)
{     if (e2)
            s1;
} else
      s2;
```

关于对非零值的测试

if(a==0)可简写成if(!a)

if(a!=0)可简写成if(a)

从而

if(a==0 && b==0)

可改为

if(!a && !b)

而

if(a!=0 && b!=0)

可改为

if(a && b)

♦ switch语句（开关语句）

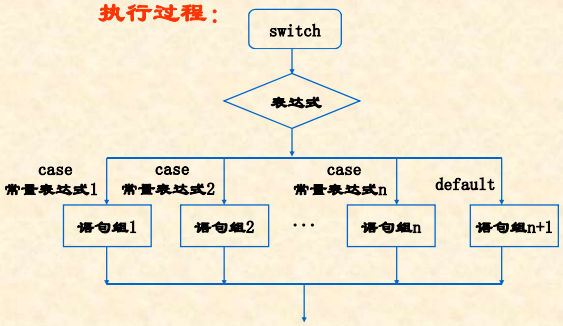
if 语句能处理从两者间选择之一，当要实现几种可能之一时，就要用 if … else 甚至多重的嵌套if来实现，当分支较多时，则嵌套的if语句层数就越多，程序不但冗长而且理解也较困难，可读性降低。因此C语言又提供了一种专门用于处理多分支结构的条件选择语句，称为switch语句，又称开关语句。C语言提供了switch开关语句专门处理多路分支的情形，使程序变得简洁。

switch语句（开关语句）的一般形式为：

```
switch(表达式)
{
   case  常量表达式1:   s 1;  [ break;]
   case  常量表达式2:   s 2;  [ break;]
         .......
   case  常量表达式n:   s n;   [break;]
   [default:            sn+1 ; [break;]]
 }
```

其中常量表达式的值必须是整型，字符型或者枚举类型，各语句序列允许有多条语句，例如：7+5，20%3，4*9，都是常量表达式。而7+x，y%3，a不是常量达式。s1、s2、…、sn+1可能是一个语句或者是语句组。

执行过程：



【例 3.8】输入选择,输出相应的名称

```
#include <stdio.h>
void main()
{ char choice;
  printf("Enter choice !(A, B, C, ...\n");
  scanf("%c", &choice);
  switch (choice)
  {
     case 'A' : printf(" A chosen!\n");
     case 'B' : printf(" B chosen!\n");
     case 'C' : printf(" C chosen!\n");
     default  : printf("default chosen!\n");
   }
}
```

执行上述示例代码时，如输入字符'B'，则程序的执行将输出：

B chosen!
C chosen!
default chosen!

由于choice的值为'B'，进入switch语句后，表达式与情况前缀'B'匹配，进入情况前缀'B'的语句序列执行。由于没有提供转出手段，因此执行将穿过case 'C'和default情况前缀(不再进行判别)，而顺序执行这些前缀之后的语句序列，产生以上所述的输出结果。

如果要使各种情况互相排斥，仅执行各case所对应的语句序列，最常用的办法是使用break语句。

【例 3.9】根据天气情况，安排活动：

```
int w_con;  /* 天气情况变量定义 */
printf("天气如何? [1:晴天, 2:多云, 3:下雨] ");
scanf("%d",&w_con);
switch (w_con)
{  case 1 : printf("上街购物!\n");  break;
   case 2 : printf("去游泳!\n");    break;
   case 3 : printf("在家看电视!\n");break;
   default: printf("错误选择!\n");
}
```

特殊情况下，如switch表达式的多个值都需要执行相同的语句：

【例3.10】计算

$$y(x)=\begin{cases}\sin(x) & 0.5<x<1.5\\ \ln(x) & 1.5<x<4.5\\ \exp(x) & 4.5<x<7.5\end{cases}$$

为了能用switch语句描述y(x)的计算，要把实型变量x的值映射到整型。即对于x的每个区间用1个或多个整数值表示。例如，将x加上0.5后取整，让区间[0.5,1.5)映射到1，区间[1.5,4.5)映射到2、3、4，区间[4.5,7.5)映射到5、6、7。

x范围	函数值	(int)(x+0.5)
0.5 < x < 1.5	sin(x)	1
1.5 < x < 4.5	ln(x)	2、3或4
4.5 < x < 7.5	exp(x)	5、6或7

由于switch语句的表达式不允许是实型，应用于实型值选择情况时，通常需作适当处理并转换为整型。

```
#include <stdio.h>
#include <math.h>
void main(){ int x;double y;
       printf("输入整数x!\n");
       scanf("%d",&x);
       switch ((int)(x+0.5)) {
       case 1: y = sin(x);
            printf("sin(%d)=%lf\n",x,y); break;
       case 2:
       case 3:
       case 4: y = ln(x);
          printf("log(%d)=%lf\n",x,y); break;
       case 5:
       case 6:
       case 7: y = exp(x);
            printf("exp(%d)=%lf\n",x,y); break;
       default :
            printf("自变量x值超出范围\n");
            break;
       }
}
```

●总结：

☑ switch后面的表达式的值类型可以是整型、字符型或枚举型。若用实型值选择时，通常将实表达式通过适当的计算，将实型的值映照到一个较小的范围上，然后再将它转换到整型。见【例3.10】

☑ 当表达式的值与某个case中的常量表达式的值相等时，就执行相应的case后的语句序列，直到遇到break语句或到达switch结构末尾。

☑ 多个连续的case语句可以共用一个语句序列。

☑ case后的不同常量表达式的值不能相等。

☑ break的作用是改变程序在switch结构中的执行流程，将程序流程跳出switch语句，转到switch语句后的下一条语句去执行。

☑ switch语句中允许嵌套switch语句。

3.4 循环结构

引例： 求s=1+2+3+...+100之和。

解题思路：设变量s存储累加和，其初值为0，变量n作为循环变量，其值由1变化到100，将n的每一个值累加到s变量，则可以实现上述算法。（利用目前所学知识编写如下程序）

```
#include <stdio.h>
void main( )
{ int s=0,i=0;
  i=i+1;
  s=s+i;
  i=i+1;
  s=s+i;
  ...
  printf("s=%d\n",s);
}
```

在例中，i=i+1; 和s=s+i; 两语句会在程序中反复出现100次，使程序变得很长。为解决这一问题，C语言引入了循环结构。C语言中实现循环结构的语句有：while语句、do~while语句和for语句。

循环控制结构（又称重复结构）是程序中的另一个基本结构。在实际问题中，常常需要进行大量的重复处理，循环结构可以使我们只写很少的语句，而让计算机反复执行，从而完成大量类同的计算。

C语言提供了while语句，do_while语句，for语句实现循环结构。

3.4.1 while语句

while语句是当型循环控制语句

while语句的一般形式为：

```
while (e)
  s
```

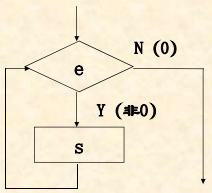
其中：e是逻辑表达式

s是循环体中的语句组，称为循环体，当需要执行多条语句时，应使用复合语句。

其特点是先判断，后执行，若条件不成立，有可能一次也不执行。

执行过程：首先计算和判断e值，如果逻辑表达式e为真，则执行 s，继续循环；否则，结束循环。

while语句是一种顶测试的循环控制结构



如前例：

```
int n,s;
i=1;    s=0;
while(i<=100)
{   s=s+i;
    i++;
}
```

```
i=1;    s=0;
while(i<=100)s+=i;i++;
```

例子中，while语句的循环体由两个语句组成，当用while语句控制的循环计算需要由多个语句组成时，必须把它们书写成复合语句形式。如上例去掉花括弧，将仅对"s += i;"作循环控制，将会出现死循环，显然是错误的。

若写成以下形式，循环体只是一个简单语句，就可不必写成复合语句。

```
s = 0;  i = 0;
while ( i < 100 )
   s += ++i;
```

如用以下代码实现跳过输入的空白类字符：

```
while((c = getchar())==' '||c=='\n'||c=='\t');
```

代码说明当循环体为空时，用空语句代替。

为使while语句的执行能正常结束，控制循环条件的变量应在循环体内被更新，使表达式的值会变为0 。

有时，很难写出循环条件，可用1代之，而在循环体中有当条件满足时，执行break语句，跳出循环。如：

```
while (1){
   ...
   if (表达式) break;
   ...
}
```

break语句

有时，我们需要在循环体中提前跳出循环，或者在满足某种条件下，不执行循环中剩下的语句而立即从头开始新的一轮循环，这时就要用到break和continue语句。在前面学习switch语句时，我们已经接触到break语句，在case子句执行完后，通过break语句使控制立即跳出switch结构。在循环语句中，break语句的作用是在循环体中测试到应立即结束循环时，使控制立即跳出循环结构，转而执行循环语句后的语句。

形式：break;

break语句经常放在循环语句的循环体中，且通常和if语句一起使用。

作用：在满足一定条件时，提前退出本层循环（不管循环控制条件是否成立），使程序流程转向该循环结构后的下一条语句执行。

```
while(e)
{
        …;
    if(exp)break;
    …;
}
    …;
```

例：打印半径为1到10的圆的面积，若面积超过100，则不予打印。

```
#include <stdio.h>
#include <math.h>
void main()
{
  int r=1;
  float area;
  while( r < 10)
  {
    area = 3.141593 * r * r ;
    r++ ;
    if (area > 100.0 )break ;
      printf ("square=%f\n ",area ) ;
  }
  printf("now r=%d\n",r ) ;
}
```

运行结果

```
square = 3.141593
square = 12.566372
square = 28.274338
square = 50.265488
square = 78.539825
now r=7
```

当break处于嵌套结构中时，它将只跳出最内层结构，而对外层结构无影响。

continue语句

形式： continue;

continue语句只能用于循环结构中。

作用：是结束本次循环，即跳过循环体中某些还没有被执行的语句，开始新的一次循环。

与break比较： continue仅结束本次循环，而break是结束整个循环语句的执行。

```
while(e)
{
        …;
        …;
 if (exp) continue;
        …;
}
```

例：计算半径为1到15的圆的面积，仅打印出超过50的圆面积

```
# include <stdio.h>
void main( )
{
  int r=1;
  float area;
  while(r<=5)
  {
     area = 3.141593 * r * r ;
     r++;
    if ( area < 50.0 )continue ;
      printf("square =%f\n",area ) ;
  }
}
```

运行结果

```
square =50.265488
square =78.539825
```

同break一样， continue语句也仅仅影响该语句本身所处的循环层，而对外层循环没有影响。

程序应用举例

【例3.13】统计输入字符行中，空白类字符、数字符和其它字符的个数。

```
#include <stdio.h>
void main()
{ int c, nwhite, nother, ndigit;
  nwhite=nother=ndigit=0; printf("输入字符行\n");
  while ((c = getchar()) != '\n')
   switch ( c ) {
    case '0': case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9':
            ndigit++; break;
    case ' ':  case '\t': nwhite++; break;
    default: nother++; break;
   }
   printf("digit = %d\twhite space = %d\tother = %d\n",
          ndigit, nwhite, nother);
}
```

3.4.2 do-while语句

在C语句中，直到型循环的语句是do...while，它的一般形式为：

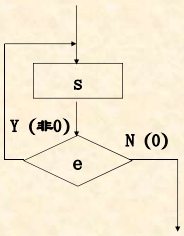
```
do  { s; }while (e);
```

其中，e是逻辑表达式，s是循环体中的语句组，称为循环体，当需要执行多条语句时，应使用复合语句。

do...while语句的流程图见右图，其基本特点是：先执行s，再判e，如果逻辑表达式e为真，则继续循环；否则，结束循环。因此s至少执行一次。

do-while语句是一种底测试的循环控制结构。

注意，分号是do-while语句的结束符，决不能省略。



【第二版 例3.16】寻找一个最小整数，该整数满足以下条件：被3除余2，被5除余3，被7除余4。

题解一：穷举法

假定该未知数为x，可从x＝4开始，逐步加1，直到x满足条件为止。代码片段如下：

```
  x = 4;
  do
    x++;
  while(!(x%3==2 && x%5==3 && x%7==4));
printf("x=%d\n", x);
}
```

题解二：分步求解法

假定该未知数为x，可从x＝2(被3除余2)开始，寻找被5除余3的解，然后寻找被7除余4的解。这样做能简化循环条件，并能加快找解的速度。

```
#include <stdio.h>
void main()
{ int i = 2; /* 初值i被3除余2 */
  do i += 3; while (i % 5 != 3);
  while (i % 7 != 4) i += 15;
  printf("被3除余2,5除余3,7除余4的最小数是%d\n\n",i);
}
```

分步求解法是在保证x被3除余2的情况下，逐步加3直至找到被3除余2而且被5除余3的x。然后在保证这样的情况下，逐步加15(3和5的积)，最后获得问题的解。而题解一中对x是逐步加1。显然分步求解的办法减少了寻找x的次数。

3.4.3 for语句

for语句是循环控制结构中使用最广泛的一种循环控制语句，特别适合已知循环次数的情况。它的一般形式为：

```
for(表达式1；表达式2；表达式3)
        s;
```

for语句很好地体现了正确表达循环结构应注意的三个问题：

1) 控制变量的初始化。
2) 循环的条件。
3) 循环控制变量的更新。

表达式1：一般为赋值表达式，给控制变量赋初值；
表达式2：关系表达式或逻辑表达式，循环控制条件；
表达式3：一般为赋值表达式，给控制变量增量或减量。
s：循环体，当有多条语句时，必须使用复合语句。

for语句的执行过程

(1) 计算表达式1；
(2) 计算表达式2，若其值为非0，则执行第3步；若为0，则转向第6步执行；
(3) 执行循环体；
(4) 计算表达式3；
(5) 跳转到第2步继续执行；
(6) 终止循环，执行for语句后的下一条语句。

图 for 循环的流程图

例 :求取m 个数列之和 $\sum_{n=1}^{m} n$
即 s = 1 + 2 + … + m

```
#include <stdio.h>
void  main( )
{
       int s,i;
       for(s=0,i=1; i<= m ; i++)
            s=s+i;
       printf("s=%d\n",s);
}
```

说明：

(1)在for语句中，若表达式1缺省，则必须将表达式1作为语句安排在for语句之前。

```
#include <stdio.h>
void  main( )
{     int s,n;
       s=0; n=1;
       for( ; n<=100; n++)
       s=s+n;
       printf(" s=%d\n ",s);
}
```

(2)在for语句中，若表达式2缺省，则系统默认循环控制条件为真（非0值），此时，如果不在循环体中加其它语句进行控制，循环将无限制进行下去，即出现死循环。

```
#include <stdio.h>
void main( )
{ int s,n;
  for(s=0,n=1; ; n++)
  {   if(n>100)
      break;
      s=s+n;
  }
 printf("s=%d\n",s);
}
```

(3)在for语句中，若表达式3缺省，可将它的语句放在循环体的最后。否则会陷入"死循环"；

```
#include <stdio.h>
void main( )
{  int s,n;
   for(s=0,n=1; n<=100; )
   {
       s=s+n;
       n++;
   }
   printf("s=%d\n",s);
}
```

(4)在for语句中三个表达式都可以缺省，但其中的两个分号不可省，相当于while(1)，无条件循环，

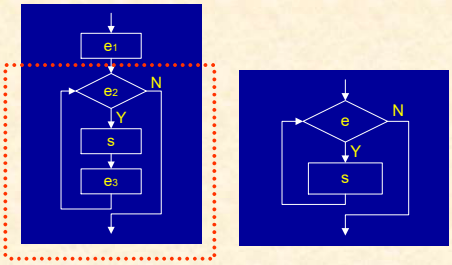
```
#include <stdio.h>
void  main( )
{ int s=0,n=1;
  for(;;)
 {  s=s+n;
    n=n+1;
   if(n>100) break;
 }
 printf("s=%d\n",s);
}
```

(5)循环体可以为空语句，但必须有分号（即循环为空语句）

```
#include <stdio.h>
void main( )
{
   int s,n;

for(s=0,n=1;n<=100;s=s+n,n++);
    printf("s=%d\n",s);
}
```

for and while



3.4.4 循环语句比较

while、do-while、和for语句都是循环控制语句，可以相互转换，但是实际使用时可以根据具体问题的需要，采用不同的思维方式来选择其中的某一个：

- 采用while的情况：

  当某个条件成立时，循环执行某个计算；特点是先判后做，循环体可能一次也没有执行。

- 采用do-while的情况：

  循环执行某个计算，直至条件不成立时结束循环；特点是先做后判，循环体至少执行一次。

- 采用for的情况：

  某个变量从初值开始循环变化，当某个条件成立时，循环执行某个计算。特点和while语句相同。

【例3.16】p = 1 - 1/3 + 1/5 - 1/7 + ...

要求：当某项的绝对值小于0.000001时程序结束。

一、使用while循环语句

对当前项t的绝对值不小于0.000001时循环求和，并修正变量d（作为分母）和t，其代码为：

```
p = 0.0; d = 1; t = 1.0;
while (fabs(t) >= 1.0e-6) {
  p += t;
  d += 2; /* 后一项的分母递增2 */
  t = ((d-1)%4) ? -1.0/d : 1.0/d;
} /* (d-1)能除尽4则当前项为正，否则为负 */
```

二、使用 do-while 循环语句

循环求和，并修正变量d（作为分母）和t（作为当前项），直至新的t的绝对值小于0.000001时结束，其代码为：

```
// p = 1 - 1/3 + 1/5 - 1/7 + ...
  p = 0.0; d = 1; t = 1.0;
  do {
     p += t;
     d += 2; /* 后一项的分母递增2 */
     t = ((d-1)%4) ? -1.0/d : 1.0/d;
  } while (fabs(t) >= 1.0e-6);
```

三、使用 for 循环语句

从变量d（作为分母）初值为1、t （作为当前项）初值为1.0开始，当t的绝对值不小于0.000001时循环求和，每次循环后相应地修正变量d和t，其代码为：

```
// p = 1 - 1/3 + 1/5 - 1/7 + ...
   for(p = 0.0, d = 1, t = 1.0;//表达式1
       fabs(t) >= 1.0e-6; //表达式2
       d += 2, //表达式3：对d和t的修正
       t =  ((d-1)%4) ? -1.0/d : 1.0/d)
       p += t;//循环体
```

注意死循环问题：dead loop

可能原因：

循环结束条件永远不满足

```
p = 1;
for(i=1; i <10;i--)
{
   p*=i; i++;
}
```

循环条件永远满足

```
p = 1;i = 1;
while( i = 10 )
{
   p*=i; i++;
}
```

3.4.5嵌套的循环结构

- 循环嵌套：指一个循环完全包含在另一个循环的循环体中。不允许循环体交叉。
- while循环、do－while循环和for循环都可以互相嵌套。
- 二重循环的执行过程是外循环执行一次，内循环执行一遍，直至内循环结束后，才能再进行一次外循环，如此反复，直到外循环结束。
- 内、外循环控制变量一般不能相同.

在编写有循环嵌套的程序时，要注意各层次上的控制循环的变量的变化规律。

【例3.23】输入整数n，输出由2*n+1行、2*n+1列，以下形式(n = 2)的图案。

```
    *
  * * *
* * * * *
  * * *
    *
```

分析：图案上面由n+1行，下面有n行。同一行上的两个星号字符之间有一个空格符。对于上半部分，设第一行的星号字符位于屏幕的中间，则后行图案的起始位置比前行起始位置提前两个位置。而对于下半部，第一行的起始位置比上半部最后一行起始位置前进两个字符位置，以后各行也相继进两个位置。

```
#include <stdio.h>
void main()
{  int n, j, k;
   printf("Enter n:"); scanf("%d", &n);
   for(j = 0; j <= n; j++) /* 图案上半部分 */
   { printf("%*c", 40 - 2*j, ' ');
     /* 输出40-2*j个空格 */
     for(k = 1; k <= 2*j+1; k++) printf(" *");
     printf("\n");
   }
   for(j = n-1; j >= 0; j--) /* 图案下半部分 */
   { printf("%*c", 40 - 2*j, ' ');
     for(k = 1; k <= 2*j+1; k++) printf(" *");
     printf("\n");
   }
}
```

【第二版 例3.23】试输入整数n(<10)，输出以下形式的数字排列图案。所示图案假定n=3。

```
      1
    1 2 1
  1 2 3 2 1
    1 2 1
      1
```

分析：将图案分成上下两部分，并对每行的开始前导空格符作按行变化的控制。

利用每行的对称性，分两个循环组织输出。如果上半部的行从1开始至n编号，对于j行，前一半输出1至j，后一半输出j-1至1。如果下半部的行从n-1至1编号，则每行结构与上半部相同。

```
#include <stdio.h>
void main()
{ int n, j, k, space;
  printf("输入 n!\n");
  scanf("%d", &n);
  space = 40; //准备输出上半图
  for(j = 1; j <= n; j++, space -= 2) {
    printf("%*c", space, ' ');
    for(k = 1; k <= j; k++)//顺序输出1至j
        printf("%2d", k);
   for(k = j-1; k >= 1; k--)//顺序输出j-1至1
        printf("%2d", k);
   printf("\n");//一行输出结束，换行
  }
```

【例3.27】输入一整数，输出小于等于该整数的全部质数的程序

设输入整数为m，程序首先输出质数2，之后对指定范围内的奇数n(3<=n<=m)判其是否是质数，若n是质数，则将数n输出。

判奇数n(>2)是否是一个质数。让整数变量k自3开始，每次增2，直至k的平方超过n为止。如果其中某个k能整除n，则n不是质数，结束测试循环。如果所有这样的k都不能整除n，因k*k大于n结束测试循环，n是质数。

```
#include  <stdio.h>
void main()
{  int m, n, k ,j; /* j用于控制每行输出10个质数*/
   printf("输入整数(>2)\n");  scanf("%ld", &m);
   printf("%6d", 2);  j = 1;
   for(n = 3L; n <= m; n += 2) {
      for(k = 3L; k*k <= n; k += 2L)
         if (n % k == 0)
            break;//测试k对n的整除性，若能整除结束测试
         if(k*k > n){//所有的k都不能整除n，则n是质数
         if (j++ % 10 == 0) printf("\n");
         printf("%6ld", n);
       }
   }
   printf("\n");
}
```

【例3.29】输入x，求级数s(x)的近似值。约定求和的精度为0.000001。

$$s(x) = x - \frac{x^3}{3*1!} + \frac{x^5}{5*2!} - \frac{x^7}{7*3!} + \cdots$$

一般地，设级数为

$$s(x) = t_0 + t_1 + t_2 + \cdots + t_k$$

求级数部分和的算法可描述如下：

```
{ s = 0;  /* 级数的部分和变量s置初值0 */
  t = 首项值; /* 置通项变量t为级数的首项值 */
  k = 0; /* 置项序号变量k为0 */
  while (fabs(t) >= Epsilon) {
    s += t;  /* 累计当前项tk到部分和 */
    t = f(t,k);/* 由当前项t和k计算下一个当前项的值 */
    k++;     /* 项序号增1 */
  }
}
```

对于本题，首项值为x，级数第k(>=0)项t_k的算式为

$(-1)^k*x^{(2*k+1)}/((2*k+1)*k!)$

k+1项t_{k+1}与k项t_k有关系

$$t_{k+1} = -t_k*x*x*(2*k+1)/((2*k+3)*(k+1))$$

t_k是通项t的当前项值，t_{k+1}是通项t的下一个当前项值。由当前项t和k计算t的下一个当前项值，可用以下表达式实现：

$$t = -t*x*x*(2.0*k+1.0)/((2.0*k+3)*(k+1))$$

把以上式子代入上述算法，并令x的值由输入给定，写出程序如下：

【例3.29】

```
#include  <stdio.h>
#include  <math.h>
#define  Epsilon  0.000001
void main()
{  int k; double s, x, t;
   printf("Enter x.\n");  scanf("%lf", &x);
    s = 0.0;  /* 级数的部分和变量s置初值0 */
    t = x; /* 置通项变量t为级数的首项值 */
    k = 0; /* 置项序号变量k为0 */
    while (fabs(t) >= Epsilon) {
      s += t; t = -t*x*x*(2.0*k+1)/((2.0*k+3)*(k+1));
      k++;      /* 项序号增1 */
    }
    printf("s(%f) = %f\n", x, s);
}
```

3.5 其他语句

● exit语句

在main函数中执行return语句是终止程序的一种方法，另一种方法是调用exit函数，此函数属于<stdlib.h>头文件中。格式：exit(表达式)；例如：exit(0); //正常退出

另外：C语言允许用 EXIT_SUCCESS来代替0（效果是相同的）：

```
exit(EXIT_SUCCESS);   /* normal termination */
```

也可用EXIT_FAILURE 表示异常终止：

```
exit(EXIT_FAILURE);  /* abnormal termination */
```

EXIT_SUCCESS和 EXIT_FAILURE都是定义在<stdlib.h>中的宏。通常分别是0和1。

作为终止程序的方法，return语句和exit函数关系紧密。事实上，main函数中的语句

return表达式;

等价于

exit(表达式);

return语句和exit函数之间的差异是：

不管哪个函数调用exit函数都会导致程序终止，而return语句仅当由main函数调用时才会导致程序终止。

● goto 语句

格式如下所示：

goto 语句标号;

其中，goto是关键字，语句标号是一种标识符，按标识符的规则来写出语句标号。语句标号是用来标识一条语句的，这种标识专门给goto转向语句使用的，即指明goto语句所要转到的语句。语句标号出现在语句的前面，用冒号（:）与语句分隔。

其格式为：

语句标号:语句

下面面通过程序实例说明got.语句的应用。

使用goto语句与if语句构成循环计算1至100自然数之和。

```
void main()
{
   int i=1,sum=0;
loop: if(i<101)
        {
     sum+=i++;
     goto loop;
        }
  printf ("%d\n",sum);
}
```

由于C语言中对goto语句采取限制使用的方法，限制goto语句转向只能在本函数体内。因此语句标号要求在一个函数体内是唯一的，不同函数体可以相同，所以，语句标号的作用范围也被限制在本函数体内。

在C语言程序中尽量要少用goto语句，最好不用goto语句，因为它会破坏结构化，影响可读性。goto.语句最常见的用法：一是用来与if语句构成循环结构，二是可以在多重循环的最内层一次退到最外边。在使用goto语句时，要注意在转向时越过循环语句的循环头和说明语句部分时，可能会出现错误，请要小心慎重。

注意：goto和break差别

3.6. 常用算法小结

●穷举法

穷举法是在很多个可能的解中，进行逐一枚举和筛选，从中找出符合要求的解。

◆猜名次 （教材p58, 例3.21）

◆水仙花数 （习题p64-4）

【例3.21】甲，乙，丙三位球迷分别预测已进入半决赛的四队A、B、C、D 的名次如下：

甲预测：A第一名、B第二名；

乙预测：C第一名、D第三名；

丙预测： D第二名、A第三名。

设比赛结果，四队名次互不相同，且甲、乙、丙的预测各对一半，求A、B、C、D四队的名次。

令变量a、b、c、d分别表记四队的名次，采用穷举法对四队所有可能的名次组合作循环测试，就能找到解。

题解分析：

1）根据三人的预测各猜对一半，可列出各人预测结果的表达式：

```
A==1 && B!=2 || A!=1 && B==2
C==1 && D!=3 || C!=1 && D==3
D==2 && A!=3 || D!=2 && A==3
```

表达式:A==1 && B!=2 || A!=1 && B==2

也可以写成：

(A==1) != (B==2)

同理：(c==1)!=(d==3)

(d==2)!=(a==3)

2）根据题意，每个队可能的名次是1 到4，但各队的名次不可相同，应该将这些相同情况排除掉。同时，4 队的名次总和为10，因此只需要三重循环来测试以上的预测。

程序如下：

```
#include <stdio.h>
void main()
{ int a, b, c, d, t;
  for(a = 1; a <= 4; a++)
    for(b = 1; b <= 4; b++) {
      if(b == a) continue; // 结束本次循环，取下一个B
      for(c = 1; c <= 4; c++) {
        if(c == a || c == b)
            continue; //结束本次循环，取下一个C
        d = 10-a-b-c;  // 四队名次之和为10
        t = ((a==1)!=(b==2)) && ((c==1)!=(d==3))
                && ((d==2)!=(a==3));
        if(t) printf("A=%d,B=%d,C=%d,D=%d\n",a,b,c,d);
      }
    }
}
```

运行结果

A=3, B=2, C=1, D=4

【习题p64-4】水仙花数(narcissus number)

◉ 要求打印出所有的"水仙花数"。

"水仙花数"指一个三位数，其各位数字立方和等于该数本身。

例如：153是一个水仙花数，因为153=1^3+5^3+3^3。

方法1：

最简单的是用三重循环

方法1：

```
#include <stdio.h>
void main()
{   int i,j,k;
    for(i=1;i<10;i++)
      for(j=0;j<10;j++)
        for(k=0;k<10;k++)
            if((i*i*i+j*j*j+k*k*k)==i*100+j*10+k)
                printf("%d%d%d\n",i,j,k);
}
```

运行结果：

153
370
371
407

```
#include <stdio.h>
void main()//方法2:
{       int a,b,c,x;
        for(x=101;x<1000;x++){
            a=x%10;
            b=x/10%10;
            c=x/100;
            if(a*a*a+b*b*b+c*c*c==x)
            printf("x=%d\n",x);
        }
 }
```

◉ 算法:

对于任意一个101到999之间的三位数x，先求出各位数字a、b、c，然后判断x和$a^3+b^3+c^3$是否相等。

◉ 算法关键：对于一个三位数x，如何用适当的表达式来表示其个位数、十位数以及百位数？

●牛顿迭代法

牛顿迭代法一般用于求解方程的根或近似解。

设方程为f(x)=0，用某种数学方法导出等价的形式

x=g(x)，然后按以下步骤执行：

1. 选一个方程的近似根，赋给变量x0；
2. 将x0的值保存为x1，然后计算g(x1)，并将结果存于变量x0；
3. 当x0与x1差的绝对值不小于某个精度要求时，重复步骤2的计算。

(教材P53, 例3.15)

【例3.15】求方程 $f(x)=3x^3+4x^2-2x+5$ 的实根

说明：

1. 用牛顿迭代方法求方程f(x)=0的根的近似解：

$x_{k+1}=x_k-f(x_k)/f'(x_k)$， k=0, 1, …

2. 当修正量$d_k=f(x_k)/f'(x_k)$ 的绝对值小于某个很小数 ε 时，x_{k+1}就作为方程的近似解。

3. 下面的程序取迭代初值为-2， ε = 1.0e-6。

```
#include <stdio.h>
#include <math.h>  /* 引用数学函数 */
#define Epsilon   1.0e-6
void main()
{ double x, d;
  x = -2.0;
  do {
        d = (((3.0*x+4.0)*x-2.0)*x+5.0)/
        ((9.0*x+8.0)*x-2.0); /* f(x)/f'(x) */
        x = x-d;
    } while (fabs(d) > Epsilon);
  printf("The root is %.6f\n", x);
} //结果为： The root is -2.053319
```

●递推法

递推法是利用问题本身所具备的某种递推关系求解的一种方法。假设问题的规模为N，要求N为1时，能够较快地得到解，而递推性质则要求当已得到问题规模为i-1的解之后，能够从已知问题规模为1, 2, …, i-1的一系列解，构造出问题规模为i的解。这样，程序可从i=0或i=1出发，通过递推的方式，得到问题规模为N的解。

Fibonacci数列 (教材P62, 例3.28)

猴子吃桃 (习题p65-22)

【例3.28】求Fibonacci 数列的前40个数，并且每行输出2 个数，前7个数列是0, 1, 1, 2, 3, 5, 8….有以下公式：

$$F_n=\begin{cases}1 & (n=1)\\ 1 & (n=2)\\ F_{n-1}+F_{n-2} & (n\ge 3)\end{cases}$$

有两种方法：
(1) 递推方法
(2) 公式计算法
见习题p65-18公式

(1) 递推方法

根据定义可得

$f_0=0$,
$f_1=1$,
$f_2=f_1+f_0=1$,
$f_3=f_2+f_1=2$,
$f_4=f_3+f_2=3$,
$f_5=f_4+f_3=5$,
……

除了f_0和f_1之外，每个数都是根据前两个数的值而获得的，即可以根据f_{i-1}和f_{i-2}，来推导出f_i，这种方法称为递推法。

```
#include <stdio.h>
void main( )
{
    int  i,f1 = 0,  f2 = 1; //初始值
    for(i=1;i<=20;i++)
    {
        printf("%d,%d\n",f1,f2);
        f1 = f1+f2;//由第1,2项求出第3项
        f2 = f1+f2;//由第2,3项求出第4项
     }
}
```

(2)公式计算法（自学）

$fy^i=(int)(C^i/\sqrt{5}+0.5)$ //0.5是补偿误差

其中，$c=(1.0+\sqrt{5})/2.0$，要求对$C^i/\sqrt{5}$

实施四舍五入，表示对fy的值舍去小数部分而取整。

对于i从0到某个n-1时，成立fxi == fyi，而当i > n-1后，fxi ≠ fyi。

要求找出第一个用两种不同方法计算结果不相等的项，也就是要求找出这个n，即存在

fxn-1 == fyn-1， fxn ≠ fyn。

根据$fxi=C^i/\sqrt{5}+0.5$， $\sqrt{5}=2.236$， $c=(1.0+\sqrt{5})/2.0=1.618$

可得：

$C^0=1$， $fy^0=C^0/\sqrt{5}+0.5=0.447+0.5=0$

$C^1=1.618$， $fy^1=C^1/\sqrt{5}+0.5=0.723+0.5=1$

$C^2=2.618$， $fy^2=C^2/\sqrt{5}+0.5=1.170+0.5=1$

$C^3=4.236$， $fy^3=C^3/\sqrt{5}+0.5=1.894+0.5=2$

$C^4=6.854$， $fy^4=C^4/\sqrt{5}+0.5=3.065+0.5=3$

$C^5=11.090$， $fy^5=C^5/\sqrt{5}+0.5=4.960+0.5=5$

以此类推。

```
#include <stdio.h>
#include <math.h>
#define SQRT5 sqrt(5.0)
void main()
{       int f0=0,fxi=1,f2,fyi;
        int i=1;
        double sqrt5 =SQRT5,c;
        c=(1.0+sqrt5)/2.0;
        do{     i++;
                f2=fxi+f0;f0=fxi;fxi=f2;//第一种方法
                c=c*(1.0+sqrt5)/2.0;
                fyi=(int)(c/sqrt5+0.5);/*0.5是补偿计算误差*/
                printf("i=%d,fai=%d,fbi=%d\n",i,fxi,fyi);
        }while(fxi==fyi);
}
```

运行结果：

```
i=2,fai=1,fbi=1
i=3,fai=2,fbi=2
i=4,fai=3,fbi=3
i=5,fai=5,fbi=5
i=6,fai=8,fbi=8
i=7,fai=13,fbi=13
i=8,fai=21,fbi=21
i=9,fai=34,fbi=34
i=10,fai=55,fbi=55
i=11,fai=89,fbi=89
i=12,fai=144,fbi=144
i=13,fai=233,fbi=233
i=14,fai=377,fbi=377
i=15,fai=610,fbi=610
```

```
i=59,fai=-1055680967,fbi=-1055680967
i=60,fai=1820529360,fbi=1820529360
i=61,fai=764848393,fbi=764848393
i=62,fai=-1709589543,fbi=-1709589543
i=63,fai=-944741150,fbi=-944741150
i=64,fai=1640636603,fbi=1640636603
i=65,fai=695895453,fbi=695895453
i=66,fai=-1958435240,fbi=-1958435240
i=67,fai=-1262539787,fbi=-1262539787
i=68,fai=1073992269,fbi=1073992269
i=69,fai=-188547518,fbi=-188547518
i=70,fai=885444751,fbi=885444751
i=71,fai=696897233,fbi=696897234
```

P65-22　循环语句：猴子吃桃问题(逆推法)

```
#include <stdio.h>
void main()
{  int i , a0,a1=0;//第5天的桃子全吃完，a1的初值
   for(i=5;i>=1;i--)
   {
      a0=2 * ( a1+ 1 );//计算当天的桃子数
      a1= a0;//将当天的桃子数作为下一次计算的初值
      printf("第%d吃了%d个桃子\n",i,a0/2+1);
   }
   printf("总共有%d个桃子\n",a0);
}
```

设计程序时应遵循的基本原则

- 正确性
- 可靠性
- 简单性
- 有效性
- 可维护性
- 可移植性
- 存储空间少
- 执行时间短

程序编码的风格

随着软件规模和复杂性的增加，人们体会到在整个软件生命周期中，程序不仅仅要被计算机理解和执行，还要经常被人阅读。

程序编码的风格是指程序的易读性、易理解性、易修改性、程序的资料文档化等。

通过这三章的学习，对于某个实际的编程问题，现在我们可以用不同的语句来编写出很多种程序来解决同一问题。所以，对我们初学者来说，关键是要多学、勤练习，熟则能生巧，到时，编写程序来就游刃有余了！

同学们，加油吧！！！