第5章　循环结构程序设计

1. 请画出例 5.6 中给出的 3 个程序段的流程图。

解：下面分别是教材第 5 章例 5.6 给出的程序，据此画出流程图。

(1) 程序 1：

```
#include <stdio.h>
int main( )
  {
    int i,j,n=0;
    for (i=1;i<=4;i++)
      for (j=1;j<=5;j++,n++)                    //n 用来累计输出数据的个数
        { if (n%5==0) printf ("\n");             //控制在输出 5 个数据后换行
          printf ("%d\t",i*j);
        }
    printf("\n");
    return 0;
  }
```

其对应的流程图见图 5.1。

运行结果：

```
1       2       3       4       5
2       4       6       8       10
3       6       9       12      15
4       8       12      16      20
```

成惠资料订购链接

(2) 程序 2：

```
#include <stdio.h>
int main( )
  {
    int i,j,n=0;
    for (i=1;i<=4;i++)
      for (j=1;j<=5;j++,n++)
        { if(n%5==0) printf("\n");                //控制在输出 5 个数据后换行
          if (i==3 && j==1) break;                //遇到第 3 行第 1 列，结束内循环
          printf("%d\t",i*j);
        }
    printf("\n");
    return 0;
  }
```

其对应的流程图见图 5.2。

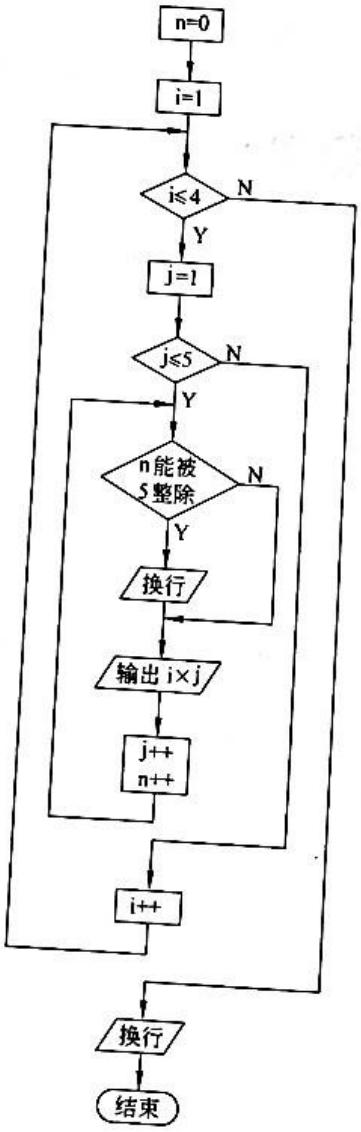

图 5.1

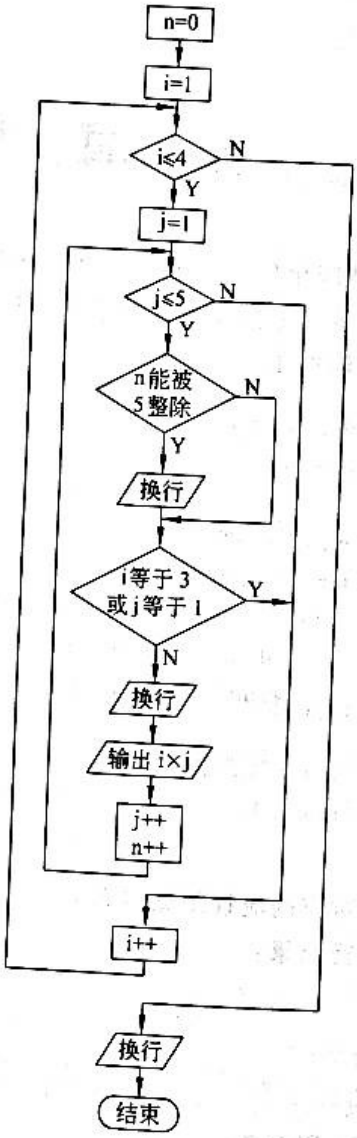

图 5.2

运行结果：

```
1    2    3    4    5
2    4    6    8    10
4    8    12   16   20
```

遇到第 3 行第 1 列时，执行 break，结束内循环，进行第 4 次外循环。

(3) 程序 3：

```
#include <stdio.h>
int main( )
  {
    int i,j,n=0;
    for (i=1;i<=4;i++)
      for (j=1;j<=5;j++,n++)
```

```
        { if(n%5==0)printf("\n");          //控制在输出 5 个数据后换行
          if (i==3 && j==1) continue;      //遇到第 3 行第 1 列,终止本次内循环
          printf("%d\t",i*j);
        }
    printf("\n");
    return 0;
}
```

其对应的流程图见图 5.3。

运行结果：

```
1       2       3       4       5
2       4       6       8       10
6       9       12      15
4       8       12      16      20
```

遇到第 3 行第 1 列时，执行 continue，只是提前结束本次内循环，不输出原来的第 3 行第 1 列的数 3，而进行下一次内循环，接着在该位置上输出原来的第 3 行第 2 列的数 6。

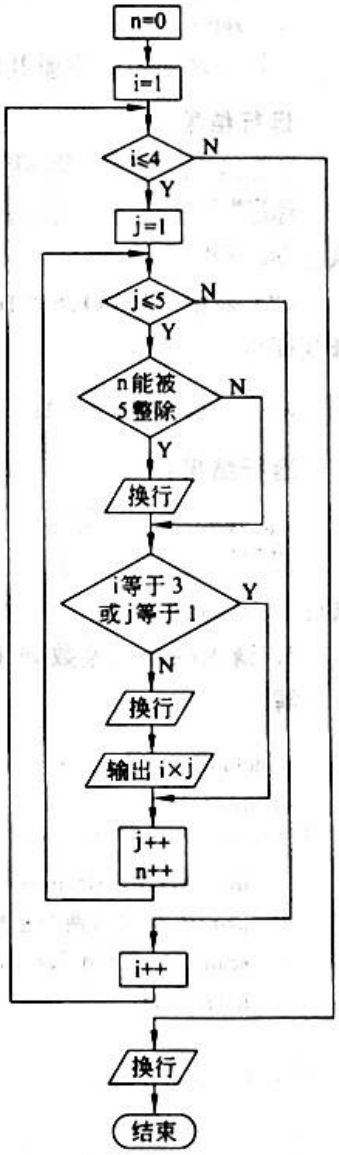

图　5.3

请仔细区分 break 语句和 continue 语句。

2. 请补充教材例 5.7 程序，分别统计当“fabs(t)>=1e−6”和“fabs(t)>=1e−8”时，执行循环体的次数。

解： 例 5.7 程序是用 $\frac{\pi}{4}\approx 1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$ 公式求 π 的近似值，直到发现某一项的绝对值小于 10^{-6} 为止。根据本题要求，分别统计当 fabs(t)>=1e−6 和 fabs(t)>=1e−8 时，执行循环体的次数。

(1) 采用 fabs(t)>=1e−6 作为循环终止条件的程序补充修改如下：

```
#include <stdio.h>
#include <math.h>
            //程序中用到数学函数 fabs,应包含头文件 math.h
int main( )
  {
    int sign=1,count=0;
                                    //sign 用来表示数值的符号,count 用来累计循环次数
    double pi=0.0,n=1.0,term=1.0;   //pi 开始代表多项式的值,最后代表 π 的值,n 代表分母,
                                    //term 代表当前项的值
    while(fabs(term)>=1e-6)     //检查当前项 term 的绝对值是否大于或等于 10 的(-6)次方
    {
     pi=pi+term;                    //把当前项 term 累加到 pi 中
     n=n+2;                         //n+2 是下一项的分母
     sign=-sign;                    //sign 代表符号,下一项的符号与上一项符号相反
     term=sign/n;                   //求出下一项的值 term
     count++;                       //count 累加 1
    }
```

```
    pi=pi*4;                          //多项式的和pi乘以4,才是π的近似值
    printf("pi=%10.8f\n",pi);         //输出π的近似值
    printf("count=%d\n",count);       //输出count的值
    return 0;
}
```

运行结果:

```
pi=3.14159065
count=500000
```

执行50万次循环。

(2) 采用fabs(t)>=1e-8作为循环终止条件的程序,只须把上面程序的第8行如下修改即可:

```
while(fabs(term)>=1e-8)
```

运行结果:

```
pi=3.14159263
count=50000000
```

执行5000万次循环。

3. 输入两个正整数 m 和 n,求其最大公约数和最小公倍数。

解:

```
#include <stdio.h>
int main( )
{
    int  p,r,n,m,temp;
    printf("请输入两个正整数n,m:");
    scanf("%d,%d",&n,&m);
    if (n<m)
      {
        temp=n;
        n=m;
        m=temp;
      }
    p=n*m;
    while(m!=0)
      {
        r=n%m;
        n=m;
        m=r;
      }
    printf("它们的最大公约数为:%d\n",n);
    printf("它们的最小公倍数为:%d\n",p/n);
    return 0;
}
```

运行结果：

```
请输入两个正整数n,m:35,49
它们的最大公约数为:7
它们的最小公倍数为:245
```

4. 输入一行字符，分别统计出其中英文字母、空格、数字和其他字符的个数。

解：

```
#include <stdio.h>
int main( )
  {
    char c;
    int letters=0,space=0,digit=0,other=0;
    printf("请输入一行字符:\n");
    while((c=getchar( ))!='\n')
     {
       if (c>='a' && c<='z' || c>='A' && c<='Z')
          letters++;
       else if (c==' ')
          space++;
       else if (c>='0' && c<='9')
          digit++;
       else
          other++;
     }
    printf("字母数:%d\n 空格数:%d\n 数字数:%d\n 其他字符数: %d\n",letters,space,digit,other);
    return 0;
  }
```

运行结果：

```
请输入一行字符:
I am a student.
字母数:11
空格数:3
数字数:0
其他字符数:1
```

5. 求 $S_n=a+aa+aaa+\cdots+\overbrace{aa\cdots a}^{n个a}$ 之值，其中 a 是一个数字，n 表示 a 的位数，n 由键盘输入，例如：2+22+222+2222+22222(此时 $n=5$)。

解：

```
#include <stdio.h>
int main( )
  {
    int  a,n,i=1,sn=0,tn=0;
    printf("a,n=:");
    scanf("%d,%d",&a,&n);
      while (i<=n)
```

```
    {
      tn=tn+a;                          //赋值后的 tn 为 i 个 a 组成数的值
      sn=sn+tn;                         //赋值后的 sn 为多项式前 i 项之和
      a=a*10;
      ++i;
    }
    printf("a+aa+aaa+...=%d\n",sn);
    return 0;
  }
```

运行结果：

```
a,n=:2,5
a+aa+aaa+...=24690
```

6. 求 $\sum_{n=1}^{20} n!$（即求 1!+2!+3!+4!+…+20!）。

解：

```
#include <stdio.h>
int main( )
  {double s=0,t=1;
    int n;
    for (n=1;n<=20;n++)
     {
      t=t*n;
      s=s+t;
     }
    printf("1!+2!+...+20!=%22.15e\n",s);
    return 0;
  }
```

运行结果：

```
1!+2!+...+20!=2.561327494111820e+018
```

请注意：s 不应定义为 int 型或 long 型，因为在用 Turbo C 或 Turbo C++ 等编译系统时，int 型数据在内存占 2 个字节，整数的范围为－32768～32767，long 数据在内存占 4 个字节，整数的范围为－21 亿～21 亿。用 Visual C++ 6.0 时，int 型和 long 型数据在内存都占 4 个字节，数据的范围为－21 亿～21 亿。无法容纳求得的结果。今将 s 定义为 double 型，以得到更多的精度。在输出时，用 22.15e 格式，使数据宽度为 22，数字部分中小数位数为 15 位。

7. 求 $\sum_{k=1}^{100} k+\sum_{k=1}^{50} k^2+\sum_{k=1}^{10} \frac{1}{k}$。

解：

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

```
int main( )
  {
    int n1=100,n2=50,n3=10;
    double k,s1=0,s2=0,s3=0;
    for (k=1;k<=n1;k++)                    //计算 1～100 的和
      {s1=s1+k;}
    for (k=1;k<=n2;k++)                    //计算 1～50 各数的平方和
      {s2=s2+k*k;}
    for (k=1;k<=n3;k++)                    //计算 1～10 的各倒数和
      {s3=s3+1/k;}
    printf("sum=%15.6f\n",s1+s2+s3);
    return 0;
  }
```

运行结果：

```
sum=   47977.928968
```

8. 输出所有的“水仙花数”。所谓“水仙花数”是指一个 3 位数，其各位数字立方和等于该数本身。例如，153 是一水仙花数，因为 $153=1^3+5^3+3^3$。

解：

```
#include <stdio.h>
int main( )
  {
    int i,j,k,n;
    printf("parcissus numbers are ");
    for (n=100;n<1000;n++)
     {
      i=n/100;
      j=n/10-i*10;
      k=n%10;
      if (n==i*i*i+j*j*j+k*k*k)
        printf("%d ",n);
     }
    printf("\n");
    return 0;
  }
```

运行结果：

```
parcissus numbers are 153 370 371 407
```

9. 一个数如果恰好等于它的因子之和，这个数就称为“完数”。例如，6 的因子为 1，2，3，而 6=1+2+3，因此 6 是“完数”。编程序找出 1000 之内的所有完数，并按下面格式输出其因子：

```
6 its factors are 1  2  3
```

解：方法一。

程序如下：

```
#define M 1000                          //定义寻找范围
#include <stdio.h>
int main( )
  {
    int k1,k2,k3,k4,k5,k6,k7,k8,k9,k10;
    int i,a,n,s;
    for (a=2;a<=M;a++)                  //a 是 2～1000 的整数,检查它是否完数
     {n=0;                              //n 用来累计 a 的因子的个数
      s=a;                              //s 用来存放尚未求出的因子之和,开始时等于 a
       for (i=1;i<a;i++)                //检查 i 是否 a 的因子
         if (a%i==0)                    //如果 i 是 a 的因子
           {n++;                        //n 加 1,表示新找到一个因子
            s=s-i;                      //s 减去已找到的因子,s 的新值是尚未求出的因子之和
            switch(n)                   //将找到的因子赋给 k1～k9,或 k10
             {case 1:
                 k1=i;  break;          //找出的第 1 个因子赋给 k1
              case 2:
                 k2=i;  break;          //找出的第 2 个因子赋给 k2
              case 3:
                 k3=i;  break;          //找出的第 3 个因子赋给 k3
              case 4:
                 k4=i;  break;          //找出的第 4 个因子赋给 k4
              case 5:
                 k5=i;  break;          //找出的第 5 个因子赋给 k5
              case 6:
                 k6=i;  break;          //找出的第 6 个因子赋给 k6
              case 7:
                 k7=i;  break;          //找出的第 7 个因子赋给 k7
              case 8:
                 k8=i;  break;          //找出的第 8 个因子赋给 k8
              case 9:
                 k9=i;  break;          //找出的第 9 个因子赋给 k9
              case 10:
                 k10=i;  break;         //找出的第 10 个因子赋给 k10
             }
           }
      if (s==0)
        {
          printf("%d ,Its factors are ",a);
          if (n>1)   printf("%d,%d",k1,k2);//n>1 表示 a 至少有 2 个因子
          if (n>2)   printf(",%d",k3);    //n>2 表示至少有 3 个因子,故应再输出一个因子
          if (n>3)   printf(",%d",k4);    //n>3 表示至少有 4 个因子,故应再输出一个因子
```

```
            if (n>4)   printf(",%d",k5);    //以下类似
            if (n>5)   printf(",%d",k6);
            if (n>6)   printf(",%d",k7);
            if (n>7)   printf(",%d",k8);
            if (n>8)   printf(",%d",k9);
            if (n>9)   printf(",%d",k10);
            printf("\n");
          }
      }
      return 0;
  }
```

运行结果：

```
6 ,Its factors are 1,2,3
28 ,Its factors are 1,2,4,7,14
496 ,Its factors are 1,2,4,8,16,31,62,124,248
```

方法二。

程序如下：

```
#include <stdio.h>
int main( )
  {int m,s,i;
    for (m=2;m<1000;m++)
      {s=0;
       for (i=1;i<m;i++)
         if ((m%i)==0) s=s+i;
       if(s==m)
        {printf("%d,its factors are ",m);
         for (i=1;i<m;i++)
           if (m%i==0)   printf("%d ",i);
         printf("\n");
         }
      }
    return 0;
  }
```

运行结果：

```
6,its factors are 1 2 3
28,its factors are 1 2 4 7 14
496,its factors are 1 2 4 8 16 31 62 124 248
```

10. 有一个分数序列：

$$\frac{2}{1},\frac{3}{2},\frac{5}{3},\frac{8}{5},\frac{13}{8},\frac{21}{13},\cdots$$

求出这个数列的前 20 项之和。

解：

```
#include <stdio.h>
int main( )
  {
    int i,n=20;
    double a=2,b=1,s=0,t;
    for (i=1;i<=n;i++)
      {
       s=s+a/b;
       t=a;
       a=a+b;
       b=t;
      }
     printf("sum=%16.10f\n",s);
     return 0;
  }
```

运行结果：

```
sum=   32.6602607986
```

11. 一个球从100m高度自由落下，每次落地后反跳回原高度的一半，再落下，再反弹。求它在第10次落地时，共经过多少米，第10次反弹多高。

解：

```
#include <stdio.h>
int main( )
  {
    double sn=100,hn=sn/2;
    int n;
    for (n=2;n<=10;n++)
     {
       sn=sn+2*hn;                    //第n次落地时共经过的米数
       hn=hn/2;                       //第n次反跳高度
      }
     printf("第10次落地时共经过%f米\n",sn);
     printf("第10次反弹%f米\n",hn);
     return 0;
  }
```

运行结果：

```
第10次落地时共经过299.609375米
第10次反弹0.097656米
```

12. 猴子吃桃问题。猴子第1天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个。第2天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第10天早上想再吃时，就只剩一个桃子了。求第1天共摘了多少个

桃子。

解：

```
#include <stdio.h>
int main( )
  {
    int day,x1,x2;
    day=9;
    x2=1;
    while(day>0)
      {x1=(x2+1)*2;                    //第 1 天的桃子数是第 2 天桃子数加 1 后的 2 倍
       x2=x1;
       day--;
      }
    printf("total=%d\n",x1);
    return 0;
  }
```

运行结果：

```
total=1534
```

13. 用迭代法求 $x=\sqrt{a}$。求平方根的迭代公式为

$$x_{n+1}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right)$$

要求前后两次求出的 x 的差的绝对值小于 10^{-5}。

解：用迭代法求平方根的算法如下：

(1) 设定一个 x 的初值 x_0；

(2) 用以上公式求出 x 的下一个值 x_1；

(3) 再将 x_1 代入以上公式右侧的 x_n，求出 x 的下一个值 x_2；

(4) 如此继续下去，直到前后两次求出的 x 值(x_n 和 x_{n+1})满足以下关系：

$$|x_{n+1}-x_n|<10^{-5}$$

为了便于程序处理，今只用 x_0 和 x_1，先令 x 的初值 $x_0=a/2$(也可以是另外的值)，求出 x_1；如果此时 $|x_1-x_0|\geqslant 10^{-5}$，就使 $x_1\Rightarrow x_0$，然后用这个新的 x_0 求出下一个 x_1；如此反复，直到 $|x_1-x_0|<10^{-5}$ 为止。

程序如下：

```
#include <stdio.h>
#include <math.h>
int main( )
  {
    float a,x0,x1;
    printf("enter a positive number:");
    scanf("%f",&a);
    x0=a/2;
```

```
    x1=(x0+a/x0)/2;
    do
     {x0=x1;
      x1=(x0+a/x0)/2;
     }while(fabs(x0-x1)>=1e-5);
    printf("The square root of %5.2f  is %8.5f\n",a,x1);
    return 0;
}
```

运行结果：

```
enter a positive number:2
The square root of  2.00  is  1.41421
```

14. 用牛顿迭代法求下面方程在 1.5 附近的根：

$$2x^3-4x^2+3x-6=0$$

解：牛顿迭代法又称**牛顿切线法**，它采用以下的方法求根：先任意设定一个与真实的根接近的值 x_0 作为第 1 次近似根，由 x_0 求出 $f(x_0)$，过$(x_0,f(x_0))$点做 $f(x)$的切线，交 x 轴于 x_1，把 x_1 作为第 2 次近似根，再由 x_1 求出 $f(x_1)$，过$(x_1,f(x_1))$点做 $f(x)$的切线，交 x 轴于 x_2，再求出 $f(x_2)$，再作切线……如此继续下去，直到足够接近真正的根 x^* 为止，见图 5.4。

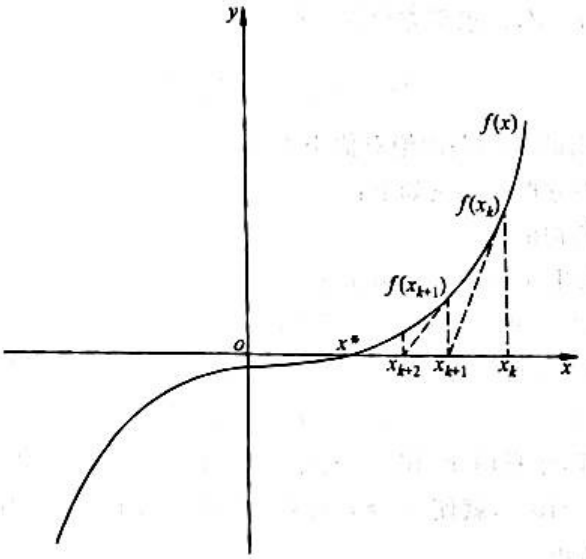

图 5.4

从图 5.4 可以看出：

$$f'(x_0)=\frac{f(x_0)}{x_1-x_0}$$

因此

$$x_1=x_0-\frac{f(x_0)}{f'(x_0)}$$

这就是牛顿迭代公式。可以利用它由 x_0 求出 x_1，然后由 x_1 求出 x_2……

在本题中

$$f(x) = 2x^3 - 4x^2 + 3x - 6$$

可以写成以下形式：

$$f(x) = ((2x - 4)x + 3)x - 6$$

同样，$f'(x)$可写成

$$f'(x) = 6x^2 - 8x + 3 = (6x - 8)x + 3$$

用这种方法表示的表达式在运算时可节省时间。例如，求 $f(x)$只需要进行 3 次乘法和 3 次加法，而原来的表达式要经过多次指数运算、对数运算和乘法、加法运算，花费时间较多。

但是由于计算机的运算速度越来越快，这点时间开销是微不足道的。这是以前计算机的运算速度较慢时所提出的问题。由于过去编写的程序往往采用了这种形式，所以在此也顺便介绍一下，以便在阅读别人所写的程序时知其所以然。

程序如下：

```
#include <stdio.h>
#include <math.h>
  int   main( )
  {double x1,x0,f,f1;
   x1=1.5;
   do
      {x0=x1;
       f=((2*x0-4)*x0+3)*x0-6;
       f1=(6*x0-8)*x0+3;
       x1=x0-f/f1;
      }while(fabs(x1-x0)>=1e-5);
   printf("The root of equation is %5.2f\n",x1);
   return 0;
  }
```

运行结果：

```
The root of equation is 2.00
```

为了便于循环处理，程序中只设了变量 x0 和 x1，x0 代表前一次的近似根，x1 代表后一次的近似根。在求出一个 x1 后，把它的值赋给 x0，然后用它求下一个 x1。由于第 1 次执行循环体时，需要对 x0 赋值，故在开始时应先对 x1 赋一个初值（今为 1.5，也可以是接近真实根的其他值）。

15. 用二分法求下面方程在(−10,10)之间的根：

$$2x^3 - 4x^2 + 3x - 6 = 0$$

解：二分法的思路为：先指定一个区间$[x_1, x_2]$，如果函数 $f(x)$在此区间是单调变化，可以根据 $f(x_1)$和 $f(x_2)$是否同符号来确定方程 $f(x)=0$ 在$[x_1, x_2]$区间是否有一个实根。若 $f(x_1)$和 $f(x_2)$不同符号，则 $f(x)=0$ 在$[x_1, x_2]$区间必有一个（且只有一个）实根；如果 $f(x_1)$和 $f(x_2)$同符号，说明在$[x_1, x_2]$区间无实根，要重新改变 x_1 和 x_2 的值。当确定$[x_1, x_2]$有一个实根后，采取二分法将$[x_1, x_2]$区间一分为二，再判断在哪一个小区间中有实根。

如此不断进行下去,直到小区间足够小为止,见图 5.5。

算法如下:

(1) 输入 x_1 和 x_2 的值。

(2) 求出 $f(x_1)$和 $f(x_2)$。

(3) 如果 $f(x_1)$和 $f(x_2)$同符号,说明在 $[x_1,x_2]$区间无实根,返回(1),重新输入 x_1 和 x_2 的值;若 $f(x_1)$和 $f(x_2)$不同符号,则在$[x_1,x_2]$区间必有一个实根,执行(4)。

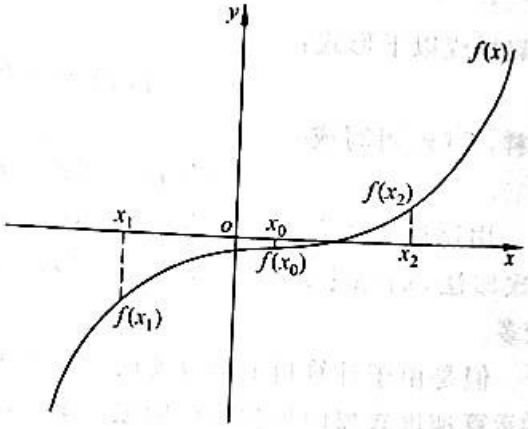

图 5.5

(4) 求 x_1 和 x_2 间的中点:$x_0=\frac{x_1+x_2}{2}$。

(5) 求出 $f(x_0)$。

(6) 判断 $f(x_0)$与 $f(x_1)$是否同符号。

① 如同符号,则应在$[x_0,x_2]$中去找根,此时 x_1 已不起作用,用 x_0 代替 x_1,用 $f(x_0)$代替 $f(x_1)$。

② 如 $f(x_0)$与 $f(x_1)$不同符号,说明应在$[x_1,x_0]$中去找根,此时 x_2 已不起作用,用 x_0 代替 x_2,用 $f(x_0)$代替 $f(x_2)$。

(7) 判断 $f(x_0)$的绝对值是否小于某一个指定的值(例如 10^{-5})。若不小于 10^{-5},就返回(4),重复执行(4)、(5)、(6);若小于 10^{-5},则执行(8)。

(8) 输出 x_0 的值,它就是所求出的近似根。

N-S 图见图 5.6。

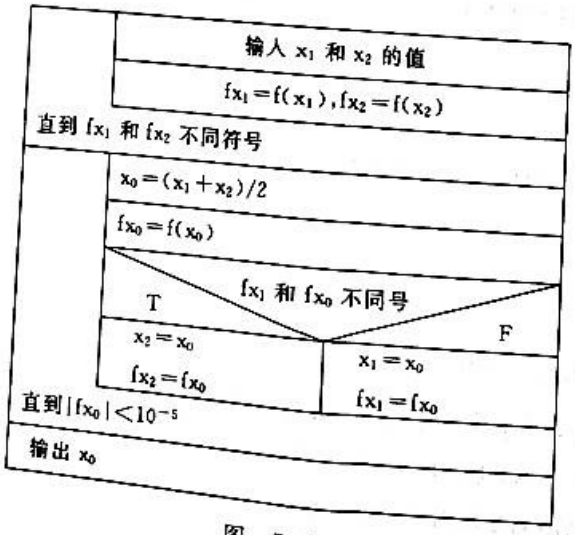

图 5.6

程序如下:

```
#include <stdio.h>
#include <math.h>
int main( )
 {float x0,x1,x2,fx0,fx1,fx2;
   do
    {printf("enter x1 &. x2:");
```

```
    scanf("%f,%f",&x1,&x2);
    fx1=x1*((2*x1-4)*x1+3)-6;
    fx2=x2*((2*x2-4)*x2+3)-6;
   }while(fx1*fx2>0);
  do
   {x0=(x1+x2)/2;
    fx0=x0*((2*x0-4)*x0+3)-6;
    if ((fx0*fx1)<0)
      {x2=x0;
       fx2=fx0;
      }
    else
     {x1=x0;
      fx1=fx0;
     }
   }while(fabs (fx0)>=1e-5);
  printf("x=%6.2f\n",x0);
  return 0;
}
```

运行结果：

```
enter x1 & x2:-10,10
x=  2.00
```

16. 输出以下图案：

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

解：

```
#include <stdio.h>
 int main( )
 {int i,j,k;
  for (i=0;i<=3;i++)
    {for (j=0;j<=2-i;j++)
       printf(" ");
     for (k=0;k<=2*i;k++)
       printf("*");
     printf("\n");
    }
  for (i=0;i<=2;i++)
```

```
    {for (j=0;j<=i;j++)
        printf(" ");
     for (k=0;k<=4-2*i;k++)
        printf("*");
     printf("\n");
    }
    return 0;
}
```

运行结果：

17. 两个乒乓球队进行比赛，各出3人。甲队为A,B,C 3人，乙队为X,Y,Z 3人。已抽签决定比赛名单。有人向队员打听比赛的名单，A说他不和X比，C说他不和X,Z比，请编程序找出3对赛手的名单。

解：先分析题目。按题意，画出图5.7的示意图。

图5.7中带×符号的虚线表示不允许的组合。从图中可以看到：①X既不与A比赛，又不与C比赛，必然与B比赛。②C既不与X比赛，又不与Z比赛，必然与Y比赛。③剩下的只能是A与Z比赛，见图5.8。

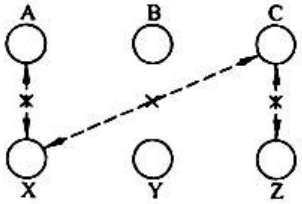

图 5.7

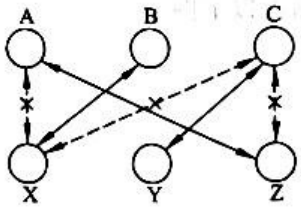

图 5.8

以上是经过逻辑推理得到的结论。用计算机程序处理此问题时，不可能立即就得出结论，而必须对每一种成对的组合一一检验，看它们是否符合条件。

开始时，并不知道A,B,C与X,Y,Z中哪一个比赛，可以假设：A与i比赛，B与j比赛，C与k比赛，即：

A—i

B—j

C—k

i,j,k分别是X,Y,Z之一，且i,j,k互不相等(一个队员不能与对方的两人比赛)，见图5.9。

外循环使i由'X'变到'Z'，中循环使j由'X'变到'Z'(但i不应与j相等)。然后对每一组i,j的值，找符合条件的k值。k同样也可能是'X'、'Y'、'Z'之一，但k也不应与i或j相等。在i≠j≠k的条件下，再把i≠'X'和k≠'X'以及k≠'Z'的i,j,k的值输出即可。

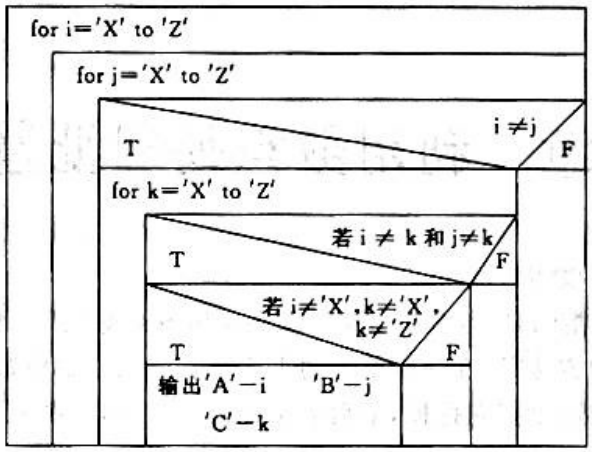

图　5.9

程序如下：

```
#include <stdio.h>
int main( )
  {
    char i,j,k;                          //i 是 a 的对手;j 是 b 的对手;k 是 c 的对手
    for (i='x';i<='z';i++)
      for (j='x';j<='z';j++)
        if (i!=j)
          for (k='x';k<='z';k++)
            if (i!=k && j!=k)
              if (i!='x' && k!='x' && k!='z')
                printf("A--%c\nB--%c\nC--%c\n",i,j,k);
    return 0;
  }
```

运行结果：

说明：

(1) 整个执行部分只有一个语句，所以只在语句的最后有一个分号。请读者弄清楚循环和选择结构的嵌套关系。

(2) 分析最下面一个 if 语句中的条件：i≠'X',k≠'X',k≠'Z'，因为已事先假定 A—i，B—j,C—k，由于题目规定 A 不与 X 对抗，因此 i 不能等于'X'，同理，C 不与 X,Z 对抗，因此 k 不应等于'X'和'Z'。

(3) 题目给的是 A,B,C,X,Y,Z，而程序中用了加撇号的字符常量'X','Y','Z'，这是为什么？这是为了在运行时能直接输出字符 A,B,C,X,Y,Z，以表示 3 组对抗的情况。