第 3 章 流程控制语句

3.1 程序的基本控制结构

3.1.1 语句的分类

一个 C++ 源程序可以由若干个源程序文件组成，一个源程序文件可以由若干个函数和编译预处理命令组成，一个函数由函数说明部分和函数执行部分组成，函数执行部分由数据定义和若干执行语句组成。语句是组成程序的基本单元。C++ 中的语句分为以下几种。

1. 说明语句

说明语句是对变量、符号常量和数据类型的定义性说明。

例如：

```
int a,b,c;                                    //定义整型变量 a、b、c
```

说明语句在程序执行期间并不执行任何操作。如，定义变量语句 int a,b,c;是告诉编译系统为变量 a、b、c 分配 12B 的存储空间用于存放变量的值。程序执行时，该语句就不起任何作用了。

说明语句可出现在函数内、外，允许出现在程序的任何地方。

2. 表达式语句

表达式语句是最简单的语句形式，一个表达式后面加上一个分号就构成了表达式语句，一般格式为

表达式；

例如，赋值表达式可以构成赋值语句：

```
a=5;
```

3. 空语句

只由一个分号构成的语句称为空语句。空语句不执行任何操作，但具有语法作用，例如 for 循环在有些情况下循环体是空语句，也有些情况下循环条件判别是空语句，这些将在 3.3 节的循环语句中介绍。大多数情况下，从程序结构的紧凑性与合理性角度考虑，尽量不要随便使用空语句。

4. 复合语句

由一对花括号“{}”括起来的一组语句构成一个复合语句。复合语句描述一个块，在

语法上起一个语句的作用。

对单个语句，必须以“;”结束；对复合语句，其中的每个语句仍以“;”结束，而整个复合语句的结束符为“}”。

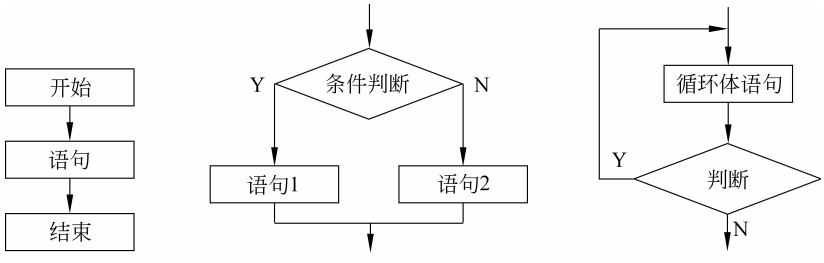
5. 流程控制语句

流程控制语句用来控制或改变程序的执行方向。

3.1.2 结构化程序的控制结构

用基本结构按一定规律组成并对算法进行描述，保证和提高算法的质量，是编写好的程序的基础，用结构化的方法设计的程序就称为结构化程序。

结构化程序由 3 种基本控制语句组成，即顺序控制、条件分支控制和循环控制；每一种控制都有赖于一种特定的程序结构来实现，因此也就有 3 种基本的程序结构：顺序结构、条件分支结构和循环结构。3 种基本结构的流程图如图 3.1 所示。



(a) 顺序结构　(b) 条件分支结构　(c) 循环结构

图 3.1　3 种基本结构框图

(1) 顺序结构：按语句的先后顺序执行。

(2) 条件分支结构：由特定的条件决定执行哪个语句的程序结构。可进一步分为单分支结构和多分支结构，在 C++ 中用 if 语句和 switch 语句实现。

(3) 循环结构：由特定的条件决定某个语句重复执行次数的控制方式。可进一步分为先判断后执行和先执行后判断。在 C++ 中用 while 语句、for 语句和 do…while 语句实现。

3.1.3 顺序结构程序应用举例

【例 3.1】 已知有 3 个整数，分别可以构成三角形的 3 条边，求面积。

分析：根据海伦公式，已知三角形的 3 条边 a、b、c，则 $p=(a+b+c)/2$，$S=\sqrt{p(p-a)(p-b)(p-c)}$。

```
#include <iostream>
#include <cmath>
using namespace std;
void main()
{   int a,b,c;
    double s,p;
```

```
    cin>>a>>b>>c;                          //a,b,c必须能构成三角形
    p=(a+b+c)/2;
    s=sqrt(p*(p-a)*(p-b)*(p-c));
    cout<<"area="<<s<<endl;
}
```

程序运行后输入

```
5  6  7
```

显示结果为

```
area=14.6969
```

【例 3.2】 从键盘输入一个正的 3 位整数，输出它的逆转数。例如，输入 358，逆转数为 853。

分析：分别取出 3 位数中的每一位数，然后乘以相应位置上的“权”即可。

```
#include <iostream>
using namespace std;
void main()
{   int a;
    cin>>a;
    a=(a%10) * 100 +a%100/10*10 +a/100;
    cout<<a;
}
```

上述两程序都是按书写的顺序从上往下执行的，主要由简单的赋值语句、输入输出语句和注释语句组成。

3.2　流程控制语句

3.2.1　if 语句

if 语句称为分支语句或条件语句，其功能是根据给定的条件选择程序的执行方向。if 语句的基本格式为

```
if (表达式)  语句 1;
else  语句 2;
```

其中的表达式称为条件表达式，可以是 C++ 中的任意合法表达式，如算术表达式、关系表达式、逻辑表达式或逗号表达式等。语句 1 和语句 2 也称为内嵌语句，在语法上各自表现为一个语句，可以是单一语句，也可以是复合语句，还可以是空语句。分支语句的执行流程是，先计算表达式的值，若表达式的值为真(或非 0)，则执行语句 1，否则(表达式的值为假，或为 0)执行语句 2。

分支语句在一次执行中只能执行语句 1 或语句 2 中的一个。如果语句 2 是空语句，

else 也可以省略。这种情况下当条件表达式的值为假时，将不产生任何操作，直接执行分支语句之后的语句。例如，对于下列分支函数：

$$y=\begin{cases}0, & x<0\\ x^3+3x, & x\geqslant 0\end{cases}$$

用 if 语句可以描述为

```
if (x<0)  y=0;
else  y=x*x*x+3*x;
```

也可以这样描述：

```
y=0;
if (x>=0)  y=x*x*x+3*x;
```

这种描述的思想是，令 y 的值为 0，如果 $x>=0$，重新计算 y 的值，否则(即 $x<0$)，y 的值不变。

【例 3.3】 输入一个年份，判断是否为闰年。

分析：假定年份为 year，闰年的条件是：year%4==0&&year%100!=0 || year%400==0。

```
#include <iostream>
using namespace std;
void main(){
    int year;
    cout<<"输入年份:"<<endl;
    cin>>year;
    if (year%4==0&&year%100!=0||year%400==0) cout<<year<<"是闰年"<<endl;
    else cout<<year<<"不是闰年"<<endl;
}
```

运行结果如下：

```
输入年份:
1900
1900 不是闰年
```

【例 3.4】 从键盘上输入 3 个整数，输出其中的最大数。

分析：读入 3 个数，先求出两个数中较大者，再将该大数与第三个数比较，求出最大数。

```
#include <iostream>
using namespace std;
void main(){
    int a, b, c, max;
    cout<<"输入 3 个整数:";
    cin>>a>>b>>c;
```

```
    cout<<"a="<<a<<'\t'<<"b="<<b<<'\t'<<"c="<<c<<endl;
    if(a>b) max=a;
    else max=b;
    cout<<"最大数为:";
    if(c>max) cout<<c<<endl;
    else cout<<max<<endl;
}
```

运行结果如下：

```
输入三个整数:
2  9  6
a=2  b=9  c=6
最大数为: 9
```

在if语句中，如果内嵌语句又是if语句，就构成了嵌套if语句。if语句可实现二选一，而嵌套if语句则可以实现多选一的情况。嵌套有两种形式，一种是嵌套在else分支中，格式为

```
if (表达式 1)  语句 1;
else if (表达式 2) 语句 2;
    else if …
    else 语句 n;
```

另一种是嵌套在if分支中，格式为

```
if (表达式 1) if (表达式 2)  语句 1;
else 语句 2;
```

【例 3.5】 用嵌套if语句完成例3.4的任务。

方法1：采用第二种嵌套形式。

```
#include <iostream>
using namespace std;
void main(){
    int a, b, c, max;
    cout<<"输入三个整数:";
    cin>>a>>b>>c;
    cout<<"a="<<a<<'\t'<<"b="<<b<<'\t'<<"c="<<c<<endl;
    if(a>b) if(a>c)  max=a;                          //a>b且a>c
            else max=c;                              //a>b且a<c
    else if(b>c)  max=b;                             //b>=a且b>c
            else max=c;                              //b>=a且b<c
    cout<<"最大数为:max="<<max<<endl;
}
```

运行结果如下：

```
输入三个整数:
3  7  12
a=3  b=7  c=12
最大数为: max=12
```

方法 2：采用第一种嵌套形式。

```
#include <iostream>
using namespace std;
void main(){
    int a,b,c,max;
    cout<<"输入三个整数:";
    cin>>a>>b>>c;
    cout<<"a="<<a<<'\t'<<"b="<<b<<'\t'<<"c="<<c<<endl;
    if(a>b&&a>c)  max=a;
    else if(b>a&&b>c)  max=b;
         else  max=c;
    cout<<"最大数为: max="<<max<<endl;
}
```

运行结果如下：

```
输入三个整数:
8  1  5
a=8  b=1  c=5
最大数为: max=8
```

嵌套 if 语句同样可以省略任何一个 else，这时要特别注意 else 和 if 的配对关系。C++ 规定了if 和 else 的“就近配对”原则，即相距最近且还没有配对的一对 if 和 else 首先配对。按上述规定，第二种嵌套形式中的 else 应与第二个 if 配对。如果根据程序的逻辑需要改变配对关系，则要使用块的概念，即将属于同一层的语句放在一对“{}”中。如第二种嵌套形式中，要让 else 和第一个 if 配对，语句必须写成

```
if (表达式 1) {
    if (表达式 2)  语句 1;
}
else 语句 2;
```

请看以下两个语句：

```
//语句 1
if (n%3==0)
if (n%5==0)  cout<<n<<"是 15 的倍数"<<endl;
else  cout<<n<<"是 3 的倍数但不是 5 的倍数"<<endl;
//语句 2
if (n%3==0) {
if (n%5==0)  cout<<n<<"是 15 的倍数"<<endl;
```

```
}
else  cout<<n <<"不是 3 的倍数"
```

两个语句的差别只在于一个“{}”,但表达的逻辑关系却完全不同。可以看出第二种嵌套形式较容易产生逻辑错误,而第一种形式的配对关系则非常明确,因此从程序可读性角度出发,建议尽量使用第一种嵌套形式。

【例 3.6】 某商场优惠活动规定,某商品一次购买 5 件以上(包含 5 件)10 件以下(不包含 10 件)打 9 折,一次购买 10 件以上(包含 10 件)打 8 折。设计程序,根据单价和客户的购买量计算总价。

```
#include <iostream>
using namespace std;
void main(){
    float price,discount,amount;                    //单价、折扣和总价
    int count;                                      //购买件数
    cout<<"输入单价:"<<endl;
    cin>>price;
    cout<<"输入购买件数:"<<endl;
    cin>>count;
    if(count<5)  discount=1;
    else if(count<10)  discount=0.9;
        else  discount=0.8;
    amount=price * count * discount;
    cout<<"单价:"<<price<<endl;
    cout<<"购买件数:"<<count<<"\t\t"<<"折扣:"<<discount<<endl;
    cout<<"总价:"<<amount<<endl;
}
```

运行结果如下:

```
输入单价:
80
输入购买件数:
8
单价: 80
购买件数: 8    折扣: 0.9
总价: 576
```

【例 3.7】 求一元二次方程 $ax^2+bx+c=0$ 的根。其中系数 $a(a\neq 0)$、b、c 的值由键盘输入。

分析:输入系数 $a(a\neq 0)$、b、c 后,令 delta$=b^2-4ac$,若 delta$=0$,方程有两个相同实根;若 delta>0,方程有两个不同实根;若 delta<0,方程无实根。

```
#include <iostream>
#include <cmath>
```

```
using namespace std;
void main(){
    float a,b,c;
    float delta,x1,x2;
    const float zero=0.0001;                          //定义一个很小的常数
    cout<<"输入三个系数 a(a!=0), b, c:"<<endl;
    cin>>a>>b>>c;
    cout<<"a="<<a<<'\t'<<"b="<<b<<'\t'<<"c="<<c<<endl;
    delta=b*b-4*a*c;
    if(fabs(delta)<zero){                             //绝对值很小的数即被认为是 0
      cout<<"方程有两个相同实根:";
      cout<<"x1=x2="<<-b/(2*a)<<endl;
    }
    else if(delta>0){
      delta=sqrt(delta);
      x1=(-b+delta)/(2*a);
      x2=(-b-delta)/(2*a);
      cout<<"方程有两个不同实根:";
      cout<<"x1="<<x1<<'\t'<<"x2="<<x2<<endl;
    }
    else                                              //delta<0
      cout<<"方程无实根!"<<endl;
}
```

运行结果如下：

```
输入三个系数 a(a!=0),b,c:
3  6  2
a=3  b=6  c=2
方程有两个不同实根: x1=-0.42265  x2=-1.57735
```

程序中有一个需要说明的问题，在判断 delta 是否为 0 时不是直接用表达式 delta==0，而是定义一个很小的常数 zero，用表达式 fabs(delta)<zero 进行判断。这是程序中经常遇到的关于实数判断的问题。由于实数在计算机中用浮点数表示，只能是近似值，即两个实数是不会精确相等的。因此在判断两个实数是否相等时，比较规范的方法是用两数误差小于一个很小的数的方法。例如，两实数 x 和 y 如果满足表达式 fabs(x-y)<1e-4，则认为 x 等于 y。但在 C++ 中，由于运算中使用的是双精度数，精度足够高，因此大多数情况下实数也可以直接判等，即对于本例也可以用 delta==0 进行判断。

3.2.2 switch 语句

用嵌套 if 语句可以实现多选一的情况。另外 C++ 中还提供了一个 switch 语句，称为开关语句，也可以用来实现多选一。它根据给定条件从多个分支语句序列中选择一个语句序列作为入口开始执行。格式为

```
switch (表达式) {
    case 常量表达式 1: <语句序列 1;><break;>
    case 常量表达式 2: <语句序列 2;><break;>
                       ⋮
    case 常量表达式 n: <语句序列 n;><break;>
    <default: 语句序列>
}
```

其中表达式作为判断条件，称为条件表达式，取值为整型、字符型、布尔型或枚举型，关于枚举类型稍后介绍。各常量表达式是由常量构成的表达式，类型与条件表达式相同。各语句序列可以是一个语句，也可以是一组语句。

开关语句的执行过程是：先求条件表达式的值，在常量表达式中找与之相等的分支作为执行入口，并从该分支的语句序列开始执行下去，直到遇到 break 语句或开关语句的花括号“}”为止。当条件表达式的值与所有常量表达式的值均不相等时，若有 default 分支，则执行其语句序列，否则跳出 switch 语句，执行后续语句。

关于 switch 语句，有几点需要注意：

(1) 各个 case(包括 default)分支出现的次序可以任意，通常将 default 放在最后。

(2) break 语句可选，如果没有 break 语句，每一个 case 分支都只作为开关语句的执行入口，执行完该分支后，还将接着执行其后的所有分支。因此，为保证逻辑的正确实现，通常每个 case 分支都与 break 语句联用。

(3) 每个常量表达式的取值必须各不相同，否则将引起歧义。

(4) 允许多个常量表达式对应同一个语句序列。例如：

```
char score;
cin>>score;
switch (score) {
case 'A': case 'a': cout<<"excellent"; break;
case 'B': case 'b': cout<<"good"; break;
default: cout<<"fair";
}
```

(5) 从形式上看，switch 语句的可读性比嵌套 if 语句好，但不是所有多选一的问题都可由开关语句完成，这是因为开关语句中限定了条件表达式的取值类型。而在有些情况下，尽管条件表达式本身不符合数据类型的要求，但经过处理后便可用开关语句实现。

【例 3.8】 快递公司对所运货物实行分段计费，如表 3.1 所示。

表 3.1　快递公司收费表

里程/km	1 公斤基本运费/元	每超 1 公斤加收费用/元	里程/km	1 公斤基本运费/元	每超 1 公斤加收费用/元
$s<200$	13	4	$500\leqslant s<1000$	18	8
$200\leqslant s<500$	15	6	$s\geqslant 1000$	20	10

设里程为 s，1 公斤的基本运费为 p，货物重量为 w，每超过 1 公斤加收费用为 q，则总

运费 f 为

$$f = p + (w-1)q$$

设计程序，当输入 p、w 和 s 后，计算运费 f。

分析：如果用 switch 语句，必须使表达式符合语法要求。分析发现，里程 s 的分段点均是 100 的倍数，因此，将里程 s 除以 100，取整数商，便得到若干整数值。程序如下：

```
#include <iostream>
#include <iomanip >
#include <cmath>
using namespace std;
void main(){
  int c,s,q,p;
  float w,d,f;
  cout<<"输入重量 w 和里程 s: "<<endl;
  cin>>w>>s;
  c=s/100;
  switch(c){
      case 0: case 1: {p=13; q=4;   break;}
      case 2: case 3: case 4: {p=15; q=6;   break;}
      case 5: case 6:case 7: case 9: {p=18; q=8;   break;}
      case 10: {p=20; q=10;   break;}
      default: {p=20; q=10;   }
    }
    f=p+ceil(w-1) * q;                                    //ceil 表示向上取整
    cout<<"运输单价:"<<p<<'\t'<<"重量:"<<w<<'\t'<<"里程:"<<s<<endl;
    cout<<"运费:"<<f<<endl;
}
```

运行结果如下：

```
输入重量 w 和里程 s:
2.3  230
运输单价: 15    重量: 2.3    里程: 230
运费: 27
```

【例 3.9】 设计一个计算器程序，实现加、减、乘、除运算。

分析：读入两个操作数和运算符，根据运算符完成相应运算。

```
#include <iostream>
using namespace std;
void main(){
    float num1,num2;
    char op;
    cout<<"输入操作数 1,运算符,操作数 2:"<<endl;
    cin>>num1>>op>>num2;
```

```
    switch(op){
        case '+':  cout<<num1<<op<<num2<<"="<<num1+num2<<endl; break;
        case '-':  cout<<num1<<op<<num2<<"="<<num1-num2<<endl; break;
        case '*':  cout<<num1<<op<<num2<<"="<<num1*num2<<endl; break;
        case '/':  cout<<num1<<op<<num2<<"="<<num1/num2<<endl; break;
        default :  cout<<op<<"是无效运算符!";
      }
}
```

运行结果如下：

```
输入操作数1,运算符,操作数2:
3 * 7
3*7=21
```

3.3　循环控制语句

3.3.1　for 循环

for 语句也称 for 循环，语句格式为

for (表达式1;表达式2;表达式3)　循环体语句

该语句的执行过程是：先求表达式1的值，再求表达式2的值，如果表达式2的值为真(或非0)，则执行循环体语句，并求表达式3的值，然后再计算表达式2的值，并重复以上过程，直到表达式2的值为假(或为0)，结束该循环语句。图3.2是for语句的执行流程。

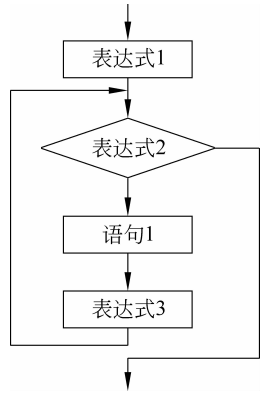


图3.2　for语句的执行流程

关于for语句，有以下几点说明：

(1) 从执行流程看，for语句属于先判断型，因此与while语句是完全等同的。

(2) for语句中的3个表达式都是包含逗号表达式在内的任意表达式。从逻辑关系看，循环初始条件可在表达式1中给出，循环条件的判断可包含在表达式2中，而循环条件变量的修改可包含在表达式3中，也可以放在循环体中。如1＋2＋…＋100的和，用for语句可描述为

```
for (i=1, s=0; i<=100; i++)  sum+=i;
```

(3) for语句中的3个表达式可部分或全部省略，但两个分号不能省略。

从执行流程看，如果表达式1放在for语句之前，或表达式3放在了循环体中，那么相应地在for语句中就可省略表达式1或表达式3。如上述语句还可写为

```
i=1; sum=0;
for (;i<=100; ) {
```

```
    sum+=i;
    i++;
}
```

实际上,表达式 2 也可省略。如果表达式 2 省略,系统约定其值为 1,这种情况表达式 2 的值恒为真。即

```
for (;;) {…}
```

等同于

```
for (; 1;) {…}
```

这种形式的循环如不加控制将导致死循环。为避免死循环,循环体内必须用 break 语句来终止循环,具体方法稍后介绍。

【例 3.10】 意大利数学家斐波那契提出了一个有趣的问题:假定每对兔子每月生出一对小兔子,新生的一对小兔子三个月后又可以生小兔子,假定所有兔子都不会死,一年后会有多少对兔子?具体说,第一个月只有一对兔子,第二个月由于新生小兔子不能生育,仍然只有一对兔子,第三个月小兔子开始生育,因此当月有两对小兔子,此后每个月的兔子数都是上个月和当月新生兔子数之和。由此可抽象出一个数列:0,1,1,2,3,5,8,…。这个数列称为 Fibonacci 数列,可用函数描述如下:

$$\text{fib}(n)=\begin{cases}0, & n=0\\ 1, & n=1\\ \text{fib}(n-2)+\text{fib}(n-1), & n>1\end{cases}$$

设计程序输出 Fibonacci 数列的前 20 项,要求每行输出 5 个数据。

```
#include<iostream>
#include<iomanip>
using namespace std;
const int m=20;
void main(){
    int fib0=0,fib1=1,fib2;
    cout<<setw(15)<<fib0<<setw(15)<<fib1;
    for(int n=3;n<=m;n++){
      fib2=fib0+fib1;
      cout<<setw(15)<<fib2;
      if(n%5==0)  cout<<endl;                    //控制每行 5 个数据
      fib0=fib1;  fib1=fib2;
    }
}
```

运行结果如下:

```
  0          1          1          2          3
  5          8         13         21         34
 55         89        144        233        377
610        987       1597       2584       4181
```

【例 3.11】　输出“1＊1＝1　2＊2＝4　3＊3＝9　…　9＊9＝81”的式子。

```
#include <iostream>
using namespace std;
void main()
{   int a;
    for(a=1;a<=9;a++)
    cout<<a<<"＊"<<a<<"="<<a＊a<<"  ";
}
```

3.3.2　do-while 循环

do-while 语句称为直到型循环，格式为

do 循环体语句　while(表达式)

该语句的执行过程是：执行一次循环体语句，然后计算表达式的值，若表达式的值为真(或非 0)，则重复上述过程，直到表达式的值为假(或为 0)时结束循环。图 3.3 给出该语句的执行流程图。

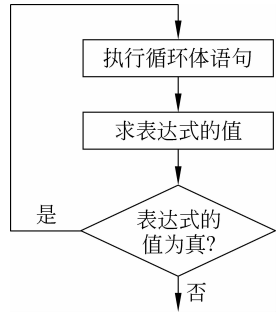


图 3.3　do-while 语句的执行流程图

do-while 语句在绝大多数情况下都能代替 while 语句，两个语句之间的区别是，do-while 语句无论条件表达式的值是真是假，循环体都将至少执行一次；而 while 语句如果条件表达式的初值为假，则循环体一次也不会执行。

【例 3.12】　用迭代法求 $x=\sqrt{a}$ 的近似值。求平方根的迭代公式为

$$x_{n+1}=(x_n+a/x_n)/2$$

要求前后两个迭代根之差小于 10^{-5}。

分析：这是递推算法的一个应用。从键盘读入一个正数赋给 a，人为估计一个值作为迭代初值 $x0$，假定取 $a/2$，根据迭代公式求出 $x1$，若 $|x1-x0|<10^{-5}$，则 $x1$ 就是所求的平方根近似值；否则，将 $x1$ 赋给 $x0$，再用公式迭代出新的 $x1$。重复以上过程，直到 $|x1-x0|<10^{-5}$ 为止。代码如下：

```
#include<iostream>
#include<math>
using namespace std;
void main(){
    float x0,x1,a;
    cout<<"输入一个正数:"<<endl;
    cin>>a;
    if(a<0)  cout<<a<<"不能开平方!"<<endl;
    else {                                          //有实数解的情况
      x1=a/2;                                       //x1 用作保存结果
      do {
```

```
        x0=x1;
        x1=(x0+a/x0)/2;
      } while (fabs(x1-x0) >=1e-5);
      cout<<a<<"的平方根为:"<<x1<<endl;
    }
}
```

运行结果如下：

```
输入一个正数:
5
5 的平方根为: 2.23607
```

【例 3.13】 输入一段文本，统计文本的行数、单词数及字符数。假定单词之间以空格、制表符或换行符间隔，假定文本没有空行。

分析：通过本例介绍一个输入结束符 EOF。执行 cin.get()将返回字符的 ASCII 码，而当读入的字符为键盘上的 Ctrl+Z 时，将返回一个整数－1，该值被定义为 EOF，因此可用这个符号作为文本输入的结束标志。该程序设计思想为：逐个读入文本中的字符，直到读到 Ctrl+Z 键为止。其中行结束标志为字符'\n'。先令行数 nline、单词数 nword 和字符数 nch 的初值均为 0。在读入过程中，每读到一个非间隔符，nch＋＋；每读到一个'\n'，nline＋＋。另设一个变量 isword，作为是否读到单词的标志。读到字符时 isword＝1，读到间隔符时 isword＝0。如果读到一个间隔符而此时 isword 值为 1，表明一个单词结束，则 nword＋＋。

```
#include<iostream>
using namespace std;
void main(){
    char ch;
    int nline=0,nword=0,nch=0;
    int isword=0;
    cout<<"输入一段文本(无空行):"<<endl;
    do{
      ch=cin.get();
      if(ch=='\n')  nline++;                          //遇换行符行数加 1
      if(ch!=' '&& ch!='\t'&&ch!='\n'&&ch!=EOF){      //读到非间隔符
        if(!isword)  nword++;                         //在单词的起始处给单词数加 1
        nch++;                                        //字符数加 1
        isword=1;
      }
      else  isword=0;                                 //读到间隔符
    } while(ch!=EOF);                                 //读到文本结束符为止
    cout<<"行数:"<<nline<<endl;
    cout<<"单词数:"<<nword<<endl;
    cout<<"字符数:"<<nch<<endl;
}
```

运行结果如下：

```
输入一段文本(无空行)：
hello
start exercise
good work
<Ctrl+Z>
行数：3
单词数：5
字符数：22
```

3.3.3　while 循环

while 语句也称为当循环。语句格式为

while (表达式)　循环体语句;

其中表达式是 C++ 中任一合法表达式，包括逗号表达式；循环体语句可以是单一语句，也可以是复合语句。while 语句的执行过程是：先计算表达式的值，如果值为真(或非0)，则执行循环体，然后再计算表达式的值，并重复以上过程，直到表达式的值为假(或为 0)，便不再执行循环体，循环语句结束。图 3.4 给出该语句的执行流程图。

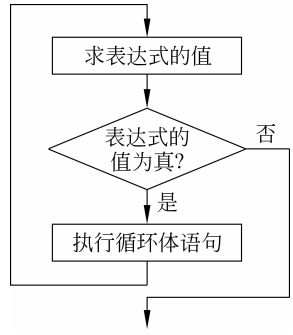


图 3.4　while 语句的执行流程图

【例 3.14】 计算 1＋2＋…＋100 的值。

分析：计算累加和实际上是重复一个循环，在循环中将下一个数与累加和相加。

```
#include <iostream>
using namespace std;
const int n=100;              //采用常变量有利于修改程序
void main(){
    int i=1,sum=0;            //循环初始条件
    while(i<=n){
      sum+=i;
      i++;                    //修改循环条件
    }
    cout<<"sum="<<sum<<endl;
}
```

运行结果如下：

```
sum=5050
```

在有循环语句的程序中，通常在循环开始前对循环条件进行初始化，如上例中的 i＝1；而在循环语句中要包含修改循环条件的语句，如上例中的 i＋＋，否则循环将不能终止而陷入死循环。

C++ 表达方式灵活，上例中的循环语句还可以写成

```
while (i<=n)  sum+=i++;
```

或者

```
while (sum+=i++, i<n);                          //循环体为空语句
```

需要说明的是，虽然 C++ 可以让代码最大限度地优化，但往往造成可读性降低，因此程序设计者只需理解这种表达方法的意义，而设计时主要追求的目标应是可读性。

3.4 循环的嵌套

一个循环体内又包含另一个完整的循环结构，称为循环的嵌套。循环嵌套包括双重循环和多重循环，可以是多个 for 的嵌套，也可以是 for 与 while 或 do 的任意组合形成的嵌套，一般把两层以上的循环程序称为多层循环的程序。

例如：

```
//以下是 while 和 for 的循环嵌套
while()                                         //外循环
{   …
     for(…){…}                                  //内循环
    …
}
//以下是 for 的多层循环嵌套
for( i=1;i<10;i++)                              //外循环
{   for (k=1;k< 6;k++)                          //内循环
    …
}
```

【例 3.15】 用双循环实现输出以下图形。

```
     *                  *                  A
    ***                *   *               BC
   *****              *   *   *            DEF
  *******            *   *   *   *         GHIJ
 *********          *   *   *   *   *      KLMNO
   (a)                (b)                (c)
```

分析：在打印图形时，首先要分析图形的特点，且每行打印多少个“*”号，从中找出规律。对图(a)来说，空格逐行减少，每次少 1 个，而“*”的个数逐行增加，每行多 2 个，如果设 a 为行数，随着行数 a 的增加，空格个数为 $5-a$，“*”号的个数为 $2a-1$。利用双循环可以编写程序如下：

```
#include <iostream>
using namespace std;
void main()
{   int a,b;
```

```
    for(a=1;a<=5;a++)
    {   for(b=1;b<=5-a;b++)cout<<' ';                    //输出每行前的空格
        for(b=1;b<=2*a-1;b++)cout<<"*";
        cout<<endl;
    }
}
```

思考：为什么有两个 b 循环，根据该程序的编写思路，写出图(b)和图(c)的程序。

3.5　跳转语句

通常程序语句都是顺序执行的，包括循环语句和分支语句，也是按照语句的语法要求顺序执行相应的操作。C++ 还提供了若干转向语句，可以改变程序原来的执行顺序。

3.5.1　break 语句

break 语句只能用在 switch 语句和循环语句中，从 break 语句处跳出 switch 语句或循环语句，转去执行 switch 语句或循环语句之后的语句。break 语句在 switch 语句中的用法前面已经介绍过，在循环语句中用来提前终止循环，请看下例：

```
for (i=10; i<20; i++) {
    if (i%3==0) break;
    cout<<i<<'\t';
}
```

该程序段执行后将输出

```
10  11
```

3.3.1 节中介绍过，for 语句的 3 个表达式均可以是空语句，在这种情况下，for 循环必然是如下形式：

```
for(;;) {
    …
    if(表达式)  break;
    …
}
```

否则将导致死循环。可见 break 语句的参与使循环语句的使用更加灵活。

需要注意的是，在嵌套循环中，break 语句终止的是其所在的循环语句，而并非终止所有的循环。例如：

```
for (;;) {
    for (;;) {
      …
      … break;
```

```
    …
}
    语句 1;
    …
}
```

当程序执行到 break 语句时，终止的是内层循环，接着执行语句 1。

【例 3.16】 给定正整数 m，判定其是否为素数。

分析：如果 $m>2$，m 是素数的条件是不能被 $2,3,\cdots,\sqrt{m}$（取整）整除。因此可以用 $2,3,\cdots,\sqrt{m}$（取整）逐个去除 m，如果 m 被其中某个数整除了，则不是素数，否则是素数。

```
#include <iostream>
#include <cmath>
using namespace std;
void main(){
    int m,i,k;
    cout<<"输入整数 m:"<<endl;
    cin>>m;
    if(m==2)   cout<<m<<"是素数"<<endl;
    else{
      k=sqrt(float(m));
      for(i=2;i<=k;i++) if (m%i==0)  break;          //只要有一个整除，就可停止
      if(i>k)  cout<<m<<"是素数"<<endl;              //循环提前终止表示不是素数
      else  cout<<m<<"不是素数"<<endl;
    }
}
```

执行结果如下：

```
输入整数：
35
35 不是素数
```

3.5.2 continue 语句

continue 语句只能用在循环语句中，用来终止本次循环。当程序执行到 continue 语句时，将跳过其后尚未执行的循环体语句，开始下一次循环。下一次循环是否执行仍然取决于循环条件的判断。例如：

```
for (i=10; i<20; i++) {
    if (i%3==0)  continue;
    cout<<i<<'\t';
}
```

该程序段执行后将输出

```
10  11  13  14  16  17  19
```

continue语句与break语句的区别在于，continue语句结束的只是本次循环，而break结束的是整个循环。

【例3.17】 利用随机函数产生10组2位数的整数，给小学生做加法，输出最后的得分。

分析：该题需要用循环多次产生2位数的随机整数，并判断学生是否做对了，如果正确，加10分，否则不加分。然后继续做下一道题。

```
#include <iostream>
#include <ctime>                                    //调用系统时间
#include <cstdlib>                                  //调用随机数的函数
using namespace std;
void main()
{   int i,a,b,c,s;
    srand((unsigned)time(0));                       //使每次产生的随机数不同
    for(i=1;i<=10;i++)                              //产生10组数
    {    a=rand()%90+10;                            //产生2位整数
        b=rand()%90+10;
        cout<<a<<"+"<<b<<"=";
        cin>>c;
        if(a+b==c){cout<<"   恭喜你,答对了!"<<endl;s++; continue;}
        cout<<"答错了!"<<endl;
    }
    cout<<"总分是："<<s*10<<endl;
}
```

3.5.3　goto语句

goto语句又称转向语句，其功能是令程序跳转到程序指定的某标号语句处。由于标号语句的设定较为灵活，因此，goto语句的跳转控制比上面介绍的跳转语句有更大的随意性。其格式为

goto 标号；

关键字goto指明该语句为goto语句，其后必须有一个标号。

标号是一个标识符，其定义出现在标号语句：

标号：语句

例如：

```
…
L1: 语句 S1
…
goto L2;
```

```
…
L2: 语句 S2
…
goto L1;
…
```

语句 goto L2;把控制流程跳转到 L2: 语句 S2;,而语句 goto L1;又跳转到 L1: 语句 S1;。标号语句可以出现在转向它的 goto 语句之后,也可以出现在其之前。goto 语句和 break、continue、return 语句都是无条件转移语句,无须任何判断,程序执行到此时,必须跳转到一个确定的位置。不过,goto 语句与其他跳转语句不同的是: goto 语句转向的目标位置由程序员任意确定,而 break,continue 和 return 语句的转向目标位置是唯一地由其本身的位置所决定的。goto 语句的转向目标——标号语句的位置虽然可由程序员确定,但不是完全任意的,它应满足下面的限制:

(1) 一个函数定义(函数体)内的 goto 语句不可转向到函数之外。换句话说,函数的出口点只能是 return 语句或函数体结束。

(2) 一个块语句(包括函数体和循环体)外的 goto 语句不可转向到该程序块之内,因为这往往会产生计算机难于处理的局面。

即使有这样的限制,goto 语句仍然是十分自由的,goto 语句的使用容易造成以下问题:

(1) 使程序的静态结构与程序的动态结构差别增大,使程序段之间形成“交叉”的关系,不利于程序的维护和调试。

(2) 一个好的程序,它的各个程序段最好是单入口单出口的,没有死循环和死区(不可到达的程序段)。但 goto 语句的使用容易破坏这种状态。

历史上关于 goto 语句的讨论是程序设计和软件开发史上的重大事件之一,最近一次的讨论是由“goto 语句是有害的”这一观点引起的,最终以“限制使用 goto 语句”作为结论,从而对软件开发的发展产生了重要影响。

因此,本书建议不用或少用 goto 语句。但在某些特定场合下 goto 语句可能会显出其价值,比如在多层循环嵌套中,要从深层地方跳出所有循环,如果用 break 语句,不仅要使用多次,而且可读性较差,这时 goto 语句可以发挥作用。

3.5.4 return 语句

return 语句又称返回语句。只用于函数定义,其功能如下:

(1) 把程序运行的流程跳转到该函数的调用点,或函数调用的出口点。

(2) 计算返回表达式 E,并把其值作为该函数的返回值。

return 语句的格式为

```
return;
```

或

```
return 表达式 E;
```