第 3 章

顺 序 结 构

C++ 的程序是结构化程序设计语言。结构化程序包括顺序、选择、循环三种控制结构。顺序结构就是按照语句的先后顺序依次执行程序的方式。

3.1 赋值语句

1. 语句格式

<变量名>=<表达式>

2. 功能

赋值语句的功能是：首先计算表达式的值，再赋给赋值号左边的变量。对任意一个变量必须先赋值，然后才能引用，否则未赋值的变量将以一个随机值参与运算。

3. 说明

(1) 赋值号左边的变量名必须在变量说明部分已经定义过。

“＝”与“＝＝”(两个等号连接在一起)的区别：“＝”赋值号是将其右边的表达式的值赋给左边的变量。例如：X＝X＋1 是将 X 的当前值加上 1，再赋给 X。而“＝＝”双等号是两边值比较的结果。

(2) 赋值号左边只能是变量名，不能是表达式、常量。

(3) 表达式是由常量、变量、函数、运算符和圆括号组成的式子。例如：

```
y=0.5;x=y;x=y+ 2.5;
```

中“＝”的右边均是表达式。

根据每个表达式所得的值的类型，表达式可分为数值表达式(值是实型或整型)、字符表达式(值是字符型)和布尔表达式(值是布尔型)，单独一个常量、一个变量或一个函数都可看作是简单的表达式。

(4) 在 C++ 中对于赋值语句提供了缩写的形式，即当一变量既出现在赋值符号的左边又出现在右边时，可以缩写。例如：

```
x=x * y; 缩写为: x * =y;
x=x+y; 缩写为: x+=y;
```

其他的算术运算“－”、“/”、“%”都可依照同样的规则在赋值中缩写。缩写格式通常更为有效，可读性也比较强。

(5) 赋值表达式是有值的，赋值表达式的值为赋值符左边表达式的值。例如：

```
cout<<(i=1)<<endl;
```

将输出 1，与此同时 i 也将被赋值为 1。

4. 应用举例

【例 3-1】 交换两个变量的值。

【问题分析】 程序中整型变量 a，b 分别赋值 3，2，交换两个变量的值。

【算法设计】 a 的值加 b 的值为 5，仍赋给 a。a 的值改为 5，将 a－b 的值赋给 b，使 b 的值改为 3。a 的值又改为 2，最后输出的 a，b 的值分别为 2，3。

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    int a=3,b=2;
    cout<<"a="<<a<<"  "<<"b="<<b<<endl;          //输出未交换的值
    a=a+b;
    b=a-b;
    a=a-b;
    cout<<"a="<<a<<"  "<<"b="<<b<<endl;          //输出交换后的值
    system("pause");
    return 0;
}
```

运行结果：

```
a=3    b=2
a=2    b=3
```

【例 3-2】 交换两个变量的值的常用方法。

【问题分析】 假如有 A、B 两个 U 盘，分别装有两种不同的内容，如图 3-1 所示。现在需要将两个 U 盘里的内容互换。设计程序完成 A、B 两个变量值的交换。

【算法设计】 现采用第三个 U 盘将两 U 盘里的内容互换。交换方法如图 3-2 所示。

上述过程可描述为：

① C←A

② A←B

③ B←C

只要将例 3-1 的处理交换的部分：a＝a＋b；b＝a－b；a＝a－b；换成：c＝a；a＝b；b＝c；并在程序的说明部分加上变量 c 的类型定义，即可。

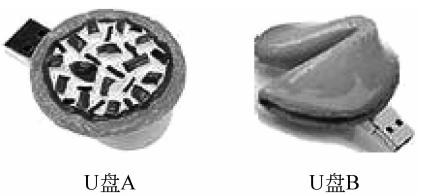

图 3-1　两个装有不同内容的 U 盘

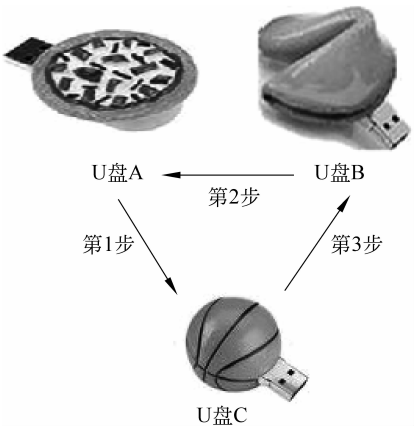

图 3-2　用第三个 U 盘 C，交换 A、B 两个 U 盘内容的过程

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    int a=3,b=2,c;
    cout<<"a="<<a<<"  "<<"b="<<b<<endl;
    c=a;
    a=b;
    b=c;
    cout<<"a="<<a<<"  "<<"b="<<b<<endl;
    system("pause");
    return 0;
}
```

运行结果：

```
a=3    b=2
a=2    b=3
```

【例 3-3】　多个变量的移动。

【问题分析】　如有 A、B、C、D 四个变量，其值如表 3-1 所示。

表 3-1　变量初始值

A	B	C	D
89	76	95	62

若将它们的值按从左向右移动，使最右边的值移动到最左边。试编程实现。

【算法设计】

(1) 开辟工作单元 E。

(2) E←D;D←C;C←B;B←A;A←E。

算法的执行过程如图 3-3 所示。

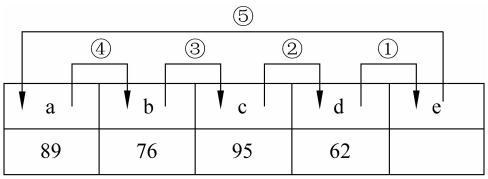

图 3-3 算法的执行过程

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    int a=89,b=76,c=95,d=62,e;
    cout<<a<<"  "<<b<<"  "<<c<<"  "<<d<<endl;
    e=d;
    d=c;
    c=b;
    b=a;
    a=e;
    cout<<a<<"  "<<b<<"  "<<c<<"  "<<d<<endl;
    system("pause");
    return 0;
}
```

运行结果:

```
89  76  95  62
62  89  76  95
```

【例 3-4】 变量赋值与相容赋值。

【问题分析】 类型相容是对参加同一运算的两个对象的类型要求。设有两个变量,如果满足下列条件之一,就说这两个变量的类型相容。

(1) 两个变量的类型相同,分为两种情况。

a. 两个变量被同一类型说明。例如:

```
int a,b[30];
```

b. 两个变量的类型是同一类型标识符。例如:

```
int a[30],b[30];
```

(2) 右边变量的类型是左边变量的子集。

【算法设计】 赋值兼容原则“就左不就右”,允许将较低精度的数据赋值给较高精度类型的变量,如果反过来将较高精度的数据赋值给较低精度的变量,则会有数据的丢失。

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    float c,d;
    cout<<"c="<<c<<endl;
    c=5.0/2.0;
```

```
    cout<<"c="<<c<<endl;
    d=5/2;
    cout<<"d="<<d<<endl;
    system("pause");
    return 0;
}
```

运行结果：

```
c=2.5
d=2
```

可以看出，变量 c 在未赋初值前的值是随机数，赋值语句中的表达式“5/2”被视为整型数据运算，c 的值为 2，而“5.0/2.0”被视为浮点型数据运算，c 此时赋值为 2.5。因此我们必须养成变量赋初值的习惯。

需要注意的是：C++ 允许将较低精度的数据赋值给较高精度类型的变量，如果反过来将较高精度的数据赋值给较低精度的变量，则会有数据的丢失，编译系统会提出警告。

如果将程序稍做改动为：

```
#include<iostream>
using namespace std;
int main()
{
    int c;
    c=5.0/2;
    cout<<"c="<<c<<endl;
    system("pause");
    return 0;
}
```

运行结果：

```
c=2
```

请找出不妥之处，并请改正。

提示：因为 c 为整型变量，而 5.0/2 的值为实型数，故变量与表达类型不同。结果将输出 c=2，只保留了结果的整数部分。若要得到 5.0/2 的准确值，只要将 c 的类型改为实型即可。

3.2　输入/输出流

输入/输出流是输入/输出的一系列字节，平时所用到“cin”、“cout”是标准输入/输出流预定义的两个流类对象，我们主要需要掌握的是“cin”、“cout”的用法。需要注意的是，在使用“cin”、“cout”前，要用编译预处理命令将所使用的头文件包含到源程序中，同时声明命名空间。

```
#include<iostream>
```

```
using namespace std;
```

3.2.1 输出流 cout

当需要在屏幕上显示输出时，可以使用插入操作符“<<”向 cout 输出流中插入字符。

例如：

```
cout<<"hello";
```

这条语句的执行结果是把 hello 输出到屏幕上。

输出语句的作用主要是在程序完成数据处理后，将计算结果以适当形式输出到屏幕上。输出语句有非格式化输出和格式化输出两种类型。

1. 非格式化输出

```
cout<<表达式 1<<表达式 2<<…<<表达式 n;
```

其中“<<”是预定义的插入运算符，作用在“cout”上，实现屏幕输出，输出结果依次为各个表达式的值。若表达式为字符串，则将双引号中间的部分按原样输出屏幕；若表达式为我们之前接触过的 endl，则输出“回车”；若表达式为算术运算，则计算出结果后输出屏幕；若为变量，则直接输出变量的值。当然，如果想输出空格，最简单的办法就是将其视为字符串输出：cout<<“ ”;。

【例 3-5】 非格式化输出。

【问题分析】 如果一个输出语句中有若干输出项，怎样区别这些输出项呢？

【算法设计】 最简单的方法是在输出项之间加入空格。

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    int a=1,b=2,c=3;
    cout<<a<<b<<c<<endl;                    //123
    cout<<a<<"  "<<b<<"  "<<c<<endl;        //1  2  3
    cout<<"Hi"<<"Frank"<<endl;              //HiFrank
    cout<<"Hi"<<"  "<<"Frank"<<endl;        //Hi  Frank
    system("pause");
    return 0;
}
```

运行结果：

```
123
1  2  3
HiFrank
Hi  Frank
```

2. 格式化输出

我们在这里主要介绍利用 C++ 标准库提供的操作符函数控制输出宽度、填充字符和精度的方法。需要注意的是在使用这些操作符函数时需要包含头文件 iomanip，即将文件 iomanip 中的内容写入到程序当中。

(1) 设置输出宽度的函数：setw(宽度值)

我们可根据输出格式的要求使用此函数在输出语句中自动定义每个输出项的宽度。

【例 3-6】 数据按宽度输出。

【问题分析】 通俗地讲，setw(n)的用法就是预设宽度，如：cout＜＜setw(5)＜＜255＜＜endl;。

【算法设计】 控制符是为了保证输出宽度。如：cout＜＜setw(3)＜＜1＜＜setw(3)＜＜10＜＜ setw(3)＜＜100；输出结果为：110100(默认是右对齐)。当输出长度大于 3 时(＜＜1000)，setw(3)不起作用。

【参考程序】

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
    int a=10,b=2,c=100;
    cout<<setw(5)<<123<<endl;
    cout<<a<<b<<c<<endl;
    cout<<setw(2)<<a<<setw(2)<<b<<setw(2)<<c<<endl;
    system("pause");
    return 0;
}
```

运行结果为：

```
  123
102100
10 2100
```

如果域宽比需要的宽度小怎么办？例如 C 的域宽为 2，但'100'的需要宽度为 3，C++ 把它扩展为最小的实际宽度。

若不使用 setw(int)设定，则各项默认宽度为 0，所以是按实际需要位数显示的。

对于浮点型数据，若设定宽度超过其实际宽度，是通过在其小数最后位之后补“0”来达到设定宽度的。

(2) 设置输出填充字符的函数：setfill (填充字符)

此函数常与 setw()函数联合使用，达到向不满设置输出宽度的空间填入指定字符的目的，不设置则填充空格。

【例 3-7】 数据按宽度输出，前面空格以其他字符填充。

【问题分析】 setfill 是设置填充的字符，如：cout＜＜setw(6)＜＜setfill('$')＜＜

27；如输出 $$$$27，$ 需加单引号。即输出总长度为 6 个，若长度不够，则在前面填充 $。

【算法设计】 setfill(char c)用法：就是在预设宽度中如果已存在没用完的宽度大小，则用设置的字符 c 填充。如：cout<<setfill('@')<<setw(5)<<255<<endl;。

【参考程序】

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
    cout<<setw(5)<<setfill('#')<<123<<endl;
    system("pause");
    return 0;
}
```

运行结果：

```
##123
```

(3) 设置输出精度的函数：setprecision(有效位数)

此函数用来指明显示浮点型数据的有效位数，可用的最后一位的值是四舍五入的值。

【例 3-8】 指定实数的输出位数。

【问题分析】 使用 setprecision(n)可控制输出流显示浮点数的数字个数。C++ 默认的流输出数值有效位是 6。

【算法设计】 cout<<setprecision(2) <<a;表示数值有两位有效数字。

【参考程序】

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
    float a=10.0/3,b=2.0/3;
    cout<<a<<"  "<<b<<endl;
    cout<<setprecision(2)<<a<<"  "<<setprecision(2)<<b<<endl;
    system("pause");
    return 0;
}
```

运行结果：

```
3.33333  0.666667
3.3  0.67
```

3.2.2 输入流 cin

当需要将初始数据送到指定变量中去时，可使用“>>”操作符作用于“cin”，实现键

盘输入。

非格式化输入格式如下：

```
cin>>变量 1>>变量 2>>…>>变量 n;
```

cin 的使用过程中要求输入的数据类型与接收变量的类型一致，并且输入的数据间用 Tab 或空格分开，回车键表示输入完毕。

例如：

```
a,b,c 为整型变量, cin>>a>>b>>c
```

执行后，键盘输入：

```
20  30  40 回车
```

结果：

```
a=20, b=30,c=40
```

又例如：

```
x 为整型变量, c 为字符型变量, cin>>x>>c;
```

执行后，键盘输入：

```
$  2.56 回车
```

结果：

```
c='$', x=2.56
```

为了正确输入数据，在输入语句之前，常加入一个输出语句以输出提示信息。

例如：

```
cout<<" a,b,c  ="<<endl;
cin>>a>>b>>c;
语句 2;
⋮
语句 n;
```

【例 3-9】 调用 cin 函数从键盘上输入数据。

【问题分析】 C++ 语言可以通过 cin 函数从键盘上输入数据。每当运行程序时，cin 函数将等待你从键盘上输入数据。

【算法设计】 cin 函数的调用形式如下：cin>>输入项；如果是多项，可以使用逗号分割，最后使用分号结束。例如：以下程序由 cout 语句提示输入，用 cin 语句从键盘输入一个整数给变量 i，最后由 cout 语句输出 i 的值。

【参考程序】

```
#include<iostream>
using namespace std;
int main()
```

```
{
    int i;
    cout<<"input i"<<endl;
    cin>>i;
    cout<<"i=="<<i<<endl;
    system("pause");
    return 0;
}
```

以下是程序运行情况：

```
input i
523(enter)
i=523
```

【例 3-10】 以下程序由键盘输入两个双精度数给变量 x 和 y，输出 x 和 y，在交换 x 和 y 中的值后，再输出 x 和 y，验证两个变量中的数值是否正确地进行了交换。

【问题分析】 实数的变量交换与前面介绍过的整数交换方法相同，用第 3 个变量做交换变量。

【算法设计】 其中 z 就是交换变量。z=x;x=y;y=z;通过三条语句完成变量 x 值和 y 值的交换。

【参考程序】

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
    double x,y,z;
    cout<<"Enter x & y:"<<endl;
    cin>>x>>y;
    cout<<"x="<<x<<setw(6)<<"y="<<y<<endl;
    z=x;x=y;y=z;
    cout<<"x="<<x<<setw(6)<<"y="<<y<<endl;
    system("pause");
    return 0;
}
```

运行结果：

```
Enter x & y:
3.1415926 2.71828
x=3.14159      y=2.71828
x=2.71828      y=3.14159
```

【例 3-11】 编写程序，把输入的分钟换算成小时和分钟表示。

【问题分析】 1 小时等于 60 分，用分钟数除以 60 所得商就是小时数，余数就是分钟数。