C++程序设计

龚雪沅
暨南大学
xygong@jnu.edu.cn

第2章 C++程序设计基础

over

2.1 词法符号

return

2.1.1 标识符

- 标识符是程序员为命名程序中的一些实体而定义的专用单词。
- **C++语言中标识符的命名规则：**

(1)标识符是由英文字母（包括大写和小写）、数字和下划线组成，并且以字母和下划线开始，其后跟零个或多个字母、数字或下划线。

(2)标识符中大写和小写字母是有区别的。

(3)标识符的长度是任意的，有的编译系统仅识别前32个字符。

(4)标识符不能和C++语言的关键字同名。

return

2.1.2 关键字

- 关键字是一种有特殊用途的词法符号，是C++系统预定义的保留字，不能再用作其他用途。
- C++语言中常用的关键字:

auto	break	bool	case	char	catch	class
Const	continue	default	delete	do	double	else
Enum	explicit	export	extern	false	float	for
friend	goto	if	inline	int	long	new

（详细可查看书中P27）

return

2.1.3 运算符

- 运算符是C++语言实现各种运算的符号。
- 根据操作对象个数的不同，可分为单目运算符、双目运算符和三目运算符。
  - ✓单目运算符又称一元运算符，它只对一个操作数进行操作。
  - ✓双目运算符又称二元运算符，它可以对两个操作数进行操作。
  - ✓三目运算符又称三元运算符，它可以对三个操作数进行操作。
  - ✓C++语言中只有一个三目运算符，就是条件运算符?：。

return

2.1.4 分隔符

- 分隔符又称标点符号，是用来分隔单词或程序正文的。
- C++语言中，常用分隔符：

(1)空格符：常用来作为单词与单词之间的分隔符。

(2)逗号：用来作为多个变量之间的分隔符，或用来作为函数多个参数之间的分隔符。

(3)分号：用来作为for循环语句中for关键字后面括号中三个表达式的分隔符，或用作语句结束的标志。

(4)冒号：用来作为语句标号与语句之间的分隔符，或switch语句中关键字case<整型常量>与语句序列之间的分隔符。

return

2.2 基本数据类型

- C++语言的数据类型可分为基本数据类型和非基本数据类型。
- 基本数据类型包括整型、字符型、浮点型和布尔型。
- 非基本数据类型主要包括数组类型、结构体类型、共用体类型、指针类型和空类型等。
- C++语言中各种数据类型（图2-1）

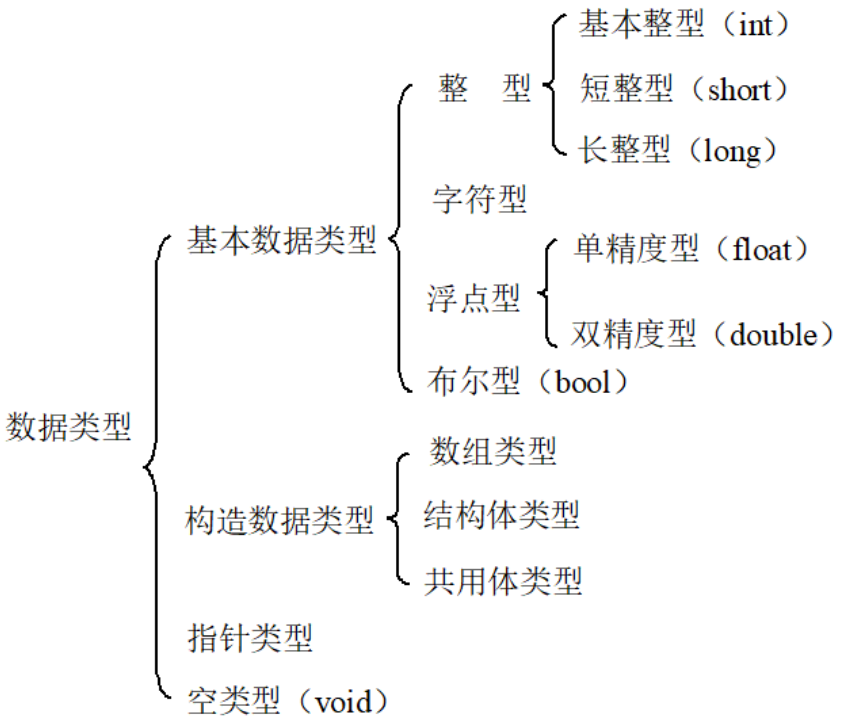


图 2-1　C++语言的数据类型

2.2 基本数据类型

- C++语言的数据类型可分为基本数据类型和非基本数据类型。
- 基本数据类型包括整型、字符型、浮点型和布尔型。
- 非基本数据类型主要包括数组类型、结构体类型、共用体类型、指针类型和空类型等。
- 数据类型决定了数据在内存中所占的空间大小，其表示范围。
- 各种基本数据类型的长度和取值范围见表2-1。

表2-1 C++基本数据类型的长度和取值范围

数据类型	说明	长度（字节）	取值范围
bool	布尔型		true，false
char（signed char）	字符型	1	-128～127
unsigned char	无符号字符型	1	0～255
short（signed short）	短整型	2	-32768～32767
unsigned short	无符号短整型	2	0～65535
int（signed int）	基本整型	4	-2147483648～2147483647
unsigned int	无符号整型	4	0～4294967295
long（signed long）	长整型	4	-2147483648～2147483647
unsigned long	无符号长整型	4	0～4294967295
float	单精度型	4	-3.4×10^{-38}～3.4×10^{38}
double	双精度型	8	-1.7×10^{-308}～1.7×10^{308}

return

2.3 常量与变量

return

2.3.1 常量

■常量定义

在程序运行过程中，值不能被改变的量称为常量。

■常量分类

整型常量

浮点型常量

字符常量

字符串常量

逻辑常量

符号常量

2.3.1 常量

1．整型常量

- (1)十进制整型常量由0~9组成，没有前缀，不能以0开始。
- (2)八进制整型常量以0为前缀，后跟由0~7组成的整型常数。

  例如，0134、-076为合法的八进制整型常量。
- (3)十六进制整型常量以0X或0x为前缀，后跟由0~9和A~F组成的整型常数。

  例如，0x2F、0xA3B4为合法的十六进制整型常量。
- **注意：**整型常量中的长整型用L或l作后缀表示；整型常量中的无符号型用U或u作后缀表示。

2.3.1 常量

2．浮点型常量

- 浮点型常量又称实型常量，是由整数部分和小数部分组成的，只能用十进制表示。
- 浮点型常量有两种表示方法：小数表示法和科学计数法。
- (1)小数表示法：是由符号、数字和小数点组成。例如，9.55、.25等。
- (2)科学计数法：是用指数形式来表示浮点型常量，即在小数表示法后面加上E或e表示指数。例如，3.2E-5，7e10，-34.5e2等。

2.3.1 常量

3．字符常量

■C++中有两种字符常量，即一般字符常量和转义字符常量。

■(1)一般字符常量

一般字符常量是用一对单引号括起来的一个字符，其值为ASCII码值，占据一个字节。

■**注意：**

✓字符常量区分大小写。

✓一个字符常量只能包含一个字符。

✓单引号是字符常量的定界符。

2.3.1 常量

3．字符常量

- C++中有两种字符常量，即一般字符常量和转义字符常量。
- (1)一般字符常量
- (2)转义字符
  - ✓转义字符是一个以“\”开头的特定字符，表示其后的字符具有特殊意义。
  - ✓例如，‘\n’中的n不是代表字符n，而是代表回车换行。
  - ✓常用的转义字符见表2-2。

2.3.1 常量

4．字符串常量

- 字符串常量是用一对双引号括起来的字符序列。
- 在内存中连续存储，并在最后加上字符'\0'作为字符串结束的标志。
- 注意：
  - ✓在C++语言中，字符串常量和字符常量是不同的。例如，"x"和'x'是不同的。
  - ✓不能将一个字符串常量赋给字符常量。

2.3.1 常量

5．逻辑常量

- 在C++程序设计中经常会用到两个逻辑值（也称布尔值）0和1，这两个值称为逻辑常量。
- 逻辑值“0”代表“假”、“不成立”、“false”等，逻辑值“1”代表“真”、“成立”、“true”等。

2.3.1 常量

6．符号常量

■指用一个标识符来表示一个常数。

■定义符号常量的两种方法：

■(1)用const语句定义符号常量

格式： const 数据类型 符号常量=表达式；

例如： const double pi = 3.1415926；

注意： 在定义符号常量时必须进行初始化，否则将出现编译错误。

■(2) 用#define语句定义符号常量

2.3.1 常量

6．符号常量

- 指用一个标识符来表示一个常数。
- 定义符号常量的两种方法：
- (1)用const语句定义符号常量
- (2) 用#define语句定义符号常量

格式：#define 常量名 常量值

例如：#define pi 3.1415926

注意：#define语句的最后不允许加分号“；”。

2.3.2 变量

- 变量是指在程序运行过程中其值可以改变的量。变量是有名字的，在内存中占据一定的存储单元。
- 1．变量的命名规则

✓(1)系统使用的关键字不能再作为变量名。

✓(2)第一个字符必须是字母或下划线，后跟字母、数字或下划线，中间不能有空格。

✓(3)命名变量应尽量做到“见名知意”，这样有助于记忆，增加可读性。

✓(4)在命名变量时，大小写字母是不一样的，习惯上用小写字母命名变量。

2.3.2 变量

- 变量是指在程序运行过程中其值可以改变的量。变量是有名字的，在内存中占据一定的存储单元。
- 2．变量的定义

  格式为：数据类型 变量1，变量2，……；

  其中数据类型可以是前面讲过的各种数据类型。
- 例如：

```
int  x,y,z;        //定义了3个整型变量x、y、z
float  a,b,c;      //定义了3个实型变量a、b、c
```

- 注意：变量必须先定义，后使用。

2.3.2 变量

■**3．变量与初始化**

■定义变量的同时进行赋值，称为变量的初始化。

■格式：数据类型 变量名=初始化值；　　//int x=10;

　　初始化值可以是一个常量，也可以是一个表达式。

■变量也可以先定义变量，后赋值。例如：int x1;x1=10;

■注意：

✓ (1)在一个语句中可以定义同一类型的多个变量；不能在一个语句中同时赋值多个变量。

✓ (2)在同一个程序块中，不能有两个相同的变量名。

✓ (3) 变量赋值时，等号左边的变量类型要和等号右边值的类型匹配。

return

2.4 运算符和表达式

- 表达式是由运算符和各种运算对象组合而成的式子。
- C++语言定义了丰富的运算符。运算符给出计算的类型和参与运算的操作数的个数。
- 运算符分为算术运算符、关系运算符和逻辑运算符等。
- 使用运算符时，要注意以下几点：

✓(1)运算符的功能。如加、减、乘、除等。

✓(2)与操作数的关系，注意操作数的个数和类型。

✓(3)运算符的优先级别。

✓(4)运算符的结合性。

- 常用C++运算符的功能、优先级和结合性（**表2-3**）

2.4 运算符和表达式

2.4.1 算术运算符与算术表达式

- 1．基本算术运算符与算术表达式
- 基本算术运算符有：+（取正或加）、－（取负或减）、*（乘）、/（除）、%（取余）。
- 上述运算符与其在数学中的意义、优先级、结合性基本相同。
- **注意：**要求取余运算符（%）的两个操作数必须是整数或字符型数据。
- 算术表达式是由算术运算符与操作数组成的，其值是一个数值，表达式的类型由运算符和操作数共同确定。

【例2.1】基本算术表达式的计算。

```
#include<iostream>
using namespace std;
int main()
{   int i=4,j=5,k=6;
    int x;
    x=i+j-k;
    cout<<"x="<<x<<endl;
    x=(i+j)*k/2;
    cout<<"x="<<x<<endl;
    x=25*4/2%k;
    cout<<"x="<<x<<endl;
    double y=2.5;
    cout<<"y="<<y-(y+0.5)*2<<endl;
    return 0;
}
```

运行结果：

```
x=3
x=27
x=2
y= -3.5
```

2.4.1 算术运算符与算术表达式

■**2．自增、自减运算符及表达式**

■都是单目运算符，有前置和后置两种形式。

■例如：i++;　　//++后置　--j;　　//--前置

■**注意：**

✓(1)自增、自减运算符只能用于变量。

✓(2)自增、自减运算符的结合方向是自右向左。

✓(3)自增、自减运算符在有些情况下的使用可能会出现歧异，从而产生意想不到的结果。

■例：

【例2.2】增1、减1表达式的计算。

```
#include<iostream>
using namespace std;
int main()
{
    int i,j,k,m,n;
    i=4;     j=i++;
    cout<<"i="<<i<<'\t'<<"j="<<j<<endl;
    i=4;     k=++i;
    cout<<"i="<<i<<'\t'<<"k="<<k<<endl;
    i=4;     m=i--;
    cout<<"i="<<i<<'\t'<<"m="<<m<<endl;
    i=4;     n=--i;
    cout<<"i="<<i<<'\t'<<"n="<<n<<endl;
    return 0;
}
```

运行结果：

```
i=5     j=4
i=5     k=5
i=3     j=4
i=3     k=3
```

return

2.4.2 关系运算符与关系表达式

- **1．关系运算符**
- 用于比较两个操作数的大小，其比较的结果是一个布尔型的值。当两个操作数满足关系运算符指定的关系时，表达式的值为true，否则为false。
- C++中，关系运算符共6个：<、<=、>、>=、==、!=。其中前4种的优先级高于后2种。
- C++语言中true等于1，false等于0。所以，关系运算符的结果可以作为算术运算中的操作数。
- 例如：表达式2 >= 3的结果为0（false）。

2.4.2 关系运算符与关系表达式

- **1．关系运算符**
- 注意：
- (1)不要把关系运算符“= =”误用为赋值运算符“=”。不要将不等于运算符“!=”写成其他语言中的 “<>”。
- (2)'a'>=60的意思是'a'的ASCII码值与60比较大小。
- (3)对数学中关系式5≤x≤20，在C++中不能写成5 <= x <= 20形式，这是错误的。正确的表达式应该是：5 <= x && x <= 20。
- **2．关系表达式：**由关系运算符和操作数组成的值为1（true）或0（false）的式子。

【例2.3】关系表达式的计算。

```
#include<iostream>
using namespace std;
int main()
{   int i=4,j=5;
    cout<<(i>j)<<endl;
    cout<<(i>=j)<<endl;
    cout<<(i<j)<<endl;
    cout<<(i<=j)<<endl;
    cout<<(i==j)<<endl;
    cout<<(i!=j)<<endl;
    return 0;
}
```

运行结果：

```
0
0
1
1
0
1
```

return

2.4.3 逻辑运算符与逻辑表达式

- **1．逻辑运算符**
- 逻辑运算符共有3个：！（逻辑求反）、&&（逻辑与）和||（逻辑或）。
- 逻辑运算的结果是逻辑值。在进行判断时，非零值为真，零为假。
- 注意：逻辑非的优先级最高，逻辑与次之，逻辑或最低。
- **2．逻辑表达式**
- 由逻辑运算符与操作数组成，表达式的值是1（true）或0（false）。

【例2.4】逻辑表达式与关系表达式的计算

```
#include<iostream>
using namespace std;
int main()
{   int x=3,y=5,z;
    z=(x>0)||(y<10);
    cout<<"z="<<z<<endl;
    z=(x==0)&&(y<10);
    cout<<"z="<<z<<endl;
    z=!(x==3);
    cout<<"z="<<z<<endl;
    return 0;
}
```

运行结果：

```
z=1
z=0
z=0
```

return

2.4.4 赋值运算符与赋值表达式

- C++中的赋值运算符分为两种：简单赋值运算符和复合赋值运算符。

1．简单赋值运算符“=”

表达式形式：变量 = 表达式

2．复合赋值运算符

- 由一个数值型运算符和基本赋值运算符组合而成。
- 共10个：+=、－=、*=、/=、%=、<<=、>>=、&=、^=、|=。

复合赋值表达式形式：变量 #= 表达式

其中，‘#’表示数值型运算符。

2.4.4 赋值运算符与赋值表达式

- C++中的赋值运算符分为两种：简单赋值运算符和复合赋值运算符。
- **2．复合赋值运算符**
- 形式：变量 #= 表达式　　**//等价于：变量 = 变量 # 表达式**
- 例如：a += 5　等价于　a = a+5

  m %=7　等价于　m = m % 7
- 复合赋值运算符的优先级、结合性都与赋值运算符相同。
- **3．赋值表达式**
- 由赋值运算符与操作数组成，把赋值运算符右边表达式的值赋给左边的变量。赋值表达式的类型为左边变量的类型。
- 在C++语言中还可以连续赋值。

【例2.5】赋值表达式的应用。

```
#include<iostream>
using namespace std;
int main()
{  int m=3,n=4,k;
   k=m++ - --n;       cout<<"k="<<k<<endl;
   char x='m',y='n';
   int z;    z=y<x;    cout<<"z="<<z<<endl;
   z=(y==x+1);
   cout<<"z="<<z<<endl;
   z=('y'!='Y');        cout<<"z="<<z<<endl;
   int a=1,b=3,c=5;    a+=b*=c-=2;
   cout<<"a="<<a<<','<<"b="<<b<<','<<"c="<<c<<endl;
   return 0;
}
```

运行结果：

```
k=0
z=0
z=1
z=1
a=10,b=9,c=3
```

return

2.4.5 位运算符

- C++有6个位运算符：~（按位求反）、&（按位与）、|（按位或）、^（按位异或）、>>（右移位）、<<（左移位）。
- 注意：
  - ✓位运算操作数只能是整型或字符型的数据，不能为实型数据。
  - ✓移位运算的结果就是位运算表达式的值，参与运算的两个操作数的值并没有发生变化。

【例2.6】位运算符的应用。

```
#include<iostream>
using namespace std;
int main()
{   int a=25,b=18,m,n,i,j,k;
    m=a&b;      cout<<"m="<<m<<endl;
    n=a|b;      cout<<"n="<<n<<endl;
    i=a^b;      cout<<"i="<<i<<endl;
    j=a<<1;     cout<<"j="<<j<<endl;
    k=a>>1;     cout<<"k="<<k<<endl;
    return 0;
}
```

运行结果为：

```
m=16
n=27
i=11
j=50
k=12
```

return

2.4.6 其他运算符

- **1．条件运算符**
- 三目运算符，形式为：表达式1？表达式2:表达式3
- 执行过程：先分析表达式1，其值为真时，则表达式2的值为条件表达式的值；否则表达式3的值为条件表达式的值。
- 优先级低于算术运算符、关系运算符和逻辑运算符，高于赋值运算符。
- 结合性为“从右到左”。
- 例如：求a和b中较大者。

```
max = a > b ? a : b
```

return

2.4.6 其他运算符

- **2．逗号运算符**
- 由逗号运算符构成的表达式。
- 一般形式为：

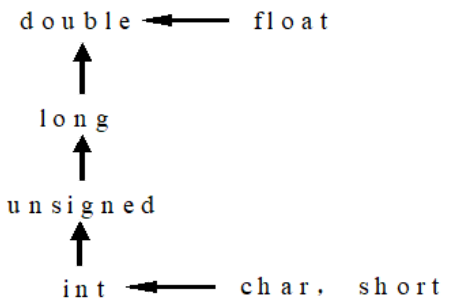
  表达式1，表达式2，…，表达式n
- 执行规则：从左到右，逐个表达式执行，最后一个表达式的值是该逗号表达式的值。
- 注意：逗号运算符的优先级最低。
- 例如：

  a=3,a+l,a*a　　结果为9

return

2.4.7 表达式中数据类型的转换

- **1. 隐含转换**
- 当操作数的类型不一致时，表达式的类型就取决于操作数中类型最高的操作数类型。
- 转换规则：



- **注意：** 隐含转换是由编译系统自动完成的，计算完成后，操作数仍保持原有的数据类型。

2.4.7 表达式中数据类型的转换

- **2. 强制转换**
- 作用是将表达式的类型强制转换成指定的数据类型。
- 一般形式：
- 数据类型（表达式）或 （数据类型）表达式
- 例如：

```
double(a)      //将a强制转换成double型
float(5%3)     //将5%3的结果转换成float型
```

- **注意**：如果将高类型转换成低类型，会造成数据精度的损失，是一种不安全的类型转换。

return

2.5 控制语句

- C++语言规定：语句必须以分号结束。
- **1．表达式语句**：由一个表达式加上分号组成。
- 例如：int i;　　a=3*4+5；//赋值语句
- **2．复合语句**：也称块语句，是由两条或两条以上的语句组成，并用“{ }”括起来的语句。
- **注意：**

✓复合语句的右括号后没有分号。

✓复合语句在语法上相当于一条语句。

- **3．控制语句**：通常包括选择语句、循环语句和转移语句。
- **4．空语句**：只有一个分号，它不作任何操作。

2.5 控制语句

2.5.1 顺序结构

- 顺序结构是程序设计中最简单、最常用的基本结构。
- 在顺序结构中，各个程序段按照先后顺序依次执行，中间没有跳转语句，程序的执行顺序不会改变。

【例2. 8】计算3个整数中的最大值。

```
#include<iostream>
using namespace std;
int main()
{  double a,b,c,max;
   cout << "输入3个数：";      cin >> a >> b>> c;
   max = a>b?a>c?a:c:b>c?b:c;
   cout << "max = " <<max << endl ;
   return 0;
}
```

运行结果为：
输入3个数：54 98 76
max = 98

2.5.2 选择语句

- 在实际应用中，有许多问题要根据是否满足某些条件来选择程序下一步要执行的操作。
- 选择结构特点：对给定的条件进行判定，并根据判定结果决定执行哪些操作，不执行哪些操作，从而控制程序执行的流程。
- C++语言中提供的选择结构：
  - ✓ if语句
  - ✓ switch语句

2.5.2 选择语句

- **1．if语句**
- 用来有条件地执行某一语句系列。
- if语句主要有3种形式:
  - ✓（1）简单if语句
  - ✓（2）两分支if语句
  - ✓（1）多分支if语句

2.5.2 选择语句

- **1．if语句**
- （1）简单if语句

  格式：if（表达式）

  　　{　语句；　}
- 执行过程：首先计算表达式的值，如果表达式的值不为0，表示条件判定为真，花括号{}内的语句将被执行；否则，将执行{}后面的语句。
- 注意：

✓表达式一般是关系表达式，并且必须用（）括起来。

✓语句可以是一条语句，也可以是多条语句。如果只有一条语句，则{ }可以省略。

【例2.9】简单if语句的应用。

```
#include<iostream>
using namespace std;
int main()
{   float score;
    cout<<"Please enter your score:"<<endl;
    cin>>score;
    if (score>=60)
          cout<<"Passed!"<<endl;
    if (score<60)
    {     cout<<"No passed!"<<endl;
          cout<<"You should do your best to study"<<endl;
    }
    return 0;
}
```

运行结果：

```
Please enter your score：
70
Passed!
```

2.5.2 选择语句

■**1．if语句**

■（2）两分支if语句

格式：　if（表达式）

{　语句1；　}

else

{　语句2；　}

■执行过程：首先计算表达式的值，如果表达式条件判定为真，则执行语句1，否则将执行语句2。

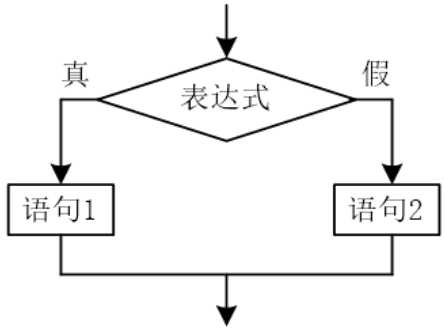


图2-5　两分支if语句

【例2.10】利用两分支if语句改写例2.9。

```
#include<iostream>
using namespace std;
int main()
{   float score;
    cout<<"Please enter your score:"<<endl;
    cin>>score;
    if (score>=60)
            cout<<"Passed!"<<endl;
    else
    {       cout<<"No passed!"<<endl;
            cout<<" You should do your best to study"<<endl;
    }
    return 0;
}
```

2.5.2 选择语句

■**1．if语句**

■（2）两分支if语句

格式：　if（表达式）

```
{　语句1;　}
else
{　语句2;　}
```

■**If语句的嵌套**：在if语句的内部还使用if语句。

■注意：

✓嵌套的层次一般不超过两层。

✓在if语句嵌套使用时，else和离它最近的上面的if配对。

【例2.11】使用if语句嵌套比较两个数的大小。

```
#include<iostream>
using namespace std;
int main()
{     int x,y;
      cout<<"输入两个整数";    cin>>x>>y;
      cout<<"x="<<x<<"  y="<<y<<endl;
      if(x!=y)
            if(x>y)
                  cout<<"x>y"<<endl;
            else
                cout<<"x<y"<<endl;
      else
                  cout<<"x=y"<<endl;
      return 0;
}
```

运行结果：

```
输入两个整数69  78
x=69  y=78
x<y
```

2.5.2 选择语句

- **1．if语句**
- （3）多分支if语句

格式：if（表达式1） <语句1>

else if（表达式2） <语句2>

else if（表达式3） <语句3>

⋮

else if（表达式n） <语句n>

else <语句n+1>

- 执行过程：先计算表达式1的值，如果表达式1为真，则执行语句1，否则判定表达式2，如果为真，则执行语句2，依此类推，直到所有的表达式均不满足，执行语句n+1。

2.5.2 选择语句

- **1．if语句**
- （3）多分支if语句

格式：if（表达式1） <语句1>

else if（表达式2） <语句2>

else if（表达式3） <语句3>

⋮

else if（表达式n） <语句n>

else <语句n+1>

- 【例2.12】输入学生成绩，给出相应等级。

90~100	优秀	80~89	良好	70~79	中等
60~69	及格	60分以下	不及格		

【例2.12】输入学生成绩，给出相应等级。

运行结果：
输入学生成绩：79
中等

```
#include<iostream>
using namespace std;
int main()
{   int score;
    cout<<"输入学生成绩：";     cin>> score;
    if(score>=90)  cout<<"优秀"<<endl;
    else    if(score>=80)  cout<<"良好"<<endl;
            else    if(score>=70)  cout<<"中等"<<endl;
                    else    if(score>=60)  cout<<"及格"<<endl;
                            else  cout<<"不及格"<<endl;
    return 0;
}
```

2.5.2 选择语句

- 2．switch语句（开关语句）
- 语法格式：

```
switch（表达式M）
{  case 常量表达式M1：语句1;
   case 常量表达式M2：语句2;
        ┇
   case 常量表达式Mn：语句n;
   default：语句Mn+1;
}
```

- 常量表达式通常为整型数值和字符常量;
- 语句是由1条或多条语句组成的语句段，可以是空语句。若是多条语句，用花括号{}括起来。

2.5.2 选择语句

- 2．switch语句（开关语句）
- 执行过程：先计算switch语句中的表达式，然后按先后顺序将结果与case中的常量表达式的值进行比较。如果两者相等，程序就转到相应case处开始顺序执行。若没有找到，就从default处开始执行。如果没有default，则转到switch语句后面的语句。
- 注意：

若希望switch语句在执行完某一case后面的语句后，不再执行其后面case和default分支，就需要在每个case的末尾加上一条break语句，表示跳出switch语句。

- 例：

【例2.13】根据考试成绩的等级给出百分制分数段。

```
#include<iostream>
using namespace std;
int main()
{   char grade;
   cout<<"请输入成绩：  "<<endl;    cin>>grade;
   if (grade>='a' && grade<='z')      grade-=32;     //小写转大写
   switch(grade)
   {   case  'A' :cout<<"90~100"<< endl;
       case  'B' :cout<<"80~89"<< endl;
       case  'C' :cout<<"70~79"<< endl;
       case  'D' :cout<<"60~69"<< endl;
       case  'E' :cout<<"60分以下"<< endl;
       default:cout<<"Input error!"<<endl;
   }
   return 0;
}
```

问题：输出结果是不符合题目原意

运行结果：

```
请输入成绩：
B
80~89
70~79
60~69
60分以下
Input error!
```

【例2.13】根据考试成绩的等级给出百分制分数段。

```
#include<iostream>
using namespace std;
int main()
{   char grade;
   cout<<"请输入成绩：  "<<endl;    cin>>grade;
   if (grade>='a' && grade<='z')      grade-=32;     //小写转大写
   switch(grade)
   {   case  'A' :cout<<"90~100"<< endl;break;
       case  'B' :cout<<"80~89"<< endl;break;
       case  'C' :cout<<"70~79"<< endl;break;
       case  'D' :cout<<"60~69"<< endl;break;
       case  'E' :cout<<"60分以下"<< endl;break;
       default:cout<<"Input error!"<<endl;
   }
   return 0;
}
```

运行结果：
请输入成绩：
B
80~89

解决方法：加**break**语句。

return

2.5.3 循环语句

- 在程序设计中遇到需要重复执行的操作，可以使用循环语句来实现。
- C++有3种循环语句：while、do-while和for循环语句。
- **1．while循环语句**
- 语法形式：

```
while（表达式）
    循环体;
```

其中，while是关键字；

表达式可以是C++语言中任何合法的表达式，用来判断执行循环体的条件；

循环体由语句组成，可以是一条语句或多条语句。

2.5.3 循环语句

- 在程序设计中遇到需要重复执行的操作，可以使用循环语句来实现。
- C++有3种循环语句：while、do-while和for循环语句。
- **1．while循环语句**
- 执行过程：

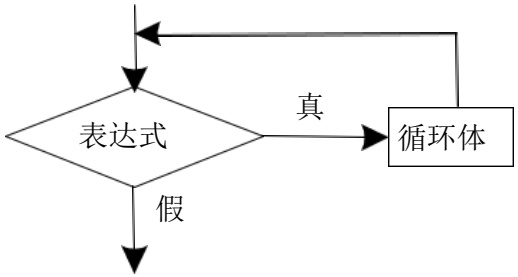


图 2-8 while 循环执行过程

2.5.3 循环语句

- 在程序设计中遇到需要重复执行的操作，可以使用循环语句来实现。
- C++有3种循环语句：while、do-while和for循环语句。
- **1．while循环语句**
- **注意：**
  - ✓如果循环体有多个语句时，要用大括号{} 括起来。
  - ✓循环语句要包含循环变量、循环条件和对循环变量有改变使得有限次循环之后能满足循环终止条件而结束循环的三部分组成。

【例2.14】编程计算1~100之和。

```
#include<iostream>
using namespace std;
int main()
{   int i=1,sum=0;
    while(i<=100)
    {
          sum+=i;
          i++;
    }
    cout<<"sum="<<sum<<endl;
    return 0;
}
```

运行结果：
sum = 5050

2.5.3 循环语句

- **2．do-while循环语句**
- 语法形式：

```
do
{    ……   //循环体部分 }
while(表达式);
```

- 与while循环语句的区别：
- do-while语句先执行循环体，后求表达式的值；while语句先求表达式的值，后执行循环体。
- do-while语句的循环体至少执行一次。While语句有可能一次都不执行循环体。
- 注意：do-while循环语句最后的分号不可缺少。

【例2.15】利用do-while循环语句改写例2.14。

```
#include<iostream>
using namespace std;
int main()
{    int i=1,sum=0;
     do
     {
         sum+=i;    i++;
     }
     while(i<=100);
     cout<<"sum="<<sum<<endl;
     return 0;
}
```

运行结果：
sum = 5050

2.5.3 循环语句

■3．for循环语句

■语法形式：

for(表达式1；表达式2；表达式3)

循环体；

■ 其中，表达式1通常用来给循环变量赋初值；表达式2用来设置循环条件；表达式3用来修改循环变量的值。

■执行过程：

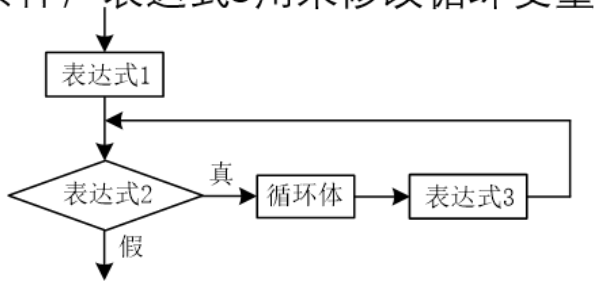


图2-10　For循环语句执行过程

【例2.16】利用for循环语句改写例2.14。

```
#include<iostream>
using namespace std;
int main()
{
   int i,sum=0;
   for(i=1; i<=100; i++)
      sum+=i;
   cout<<"sum="<<sum<<endl;
   return 0;
}
```

运行结果：

```
sum = 5050
```

2.5.3 循环语句

■3．for循环语句

■注意：

- ✓for语句中的3个表达式可以没有。但必须注意每个表达式后的分号不能省略。这时在循环体内必须有其他控制循环执行的语句，否则会形成死循环。
- ✓表达式1如果没有或不是用来给循环变量赋初值，则应在for语句前给循环变量赋初值。
- ✓表达式2如果没有，则在for语句循环体内应有其他控制循环执行的语句，否则会形成死循环。
- ✓表达式3如果没有或不是用来修改循环变量的值，则应在for语句循环体内设置相应的语句。

【例2.17】输出100~200以内的所有素数。

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int main()
{     int i,j,t=0;
      for(i=100;i<200;i++)
      {      int k=(int)sqrt(i);
               for(j=2;j<=k;j++)
                        if(i%j==0)   break;
                        if(j>k)    {
                                    cout<<setw(4)<<i;
                                   if(++t%8==0)cout<<endl;
                                  }
      }
      cout<<endl;     return 0;
}
```

运行结果：

101 103 107 109 113 127 131 137
139 149 151 157 163 167 173 179
181 191 193 197 199

return

2.5.4 转移语句

- 转移语句作用：主要用于改变程序中语句的执行顺序，使程序从某一语句有目的地转移到另一语句继续执行。
- C++提供了goto语句、break语句和continue语句等转移语句。
- **1．goto语句**
- 语法格式为：　　goto 语句标号;
- 作用：使程序转移到语句标号所标示的语句处继续执行。语句标号是一种用来标识语句的符号。
- C++中，goto语句的使用被限制在一个函数体内。在同一函数中，语句标号应该是唯一的。
- 注意：由于goto语句破坏程序结构，提倡不使用。

2.5.4 转移语句

■2．break语句

■语法形式：　break;

■适用场合：

✓用在switch语句中，用来退出switch语句。

✓用在循环语句的循环体中，用来退出循环语句。。

■3．continue语句

■语法格式为：　continue;

■适用场合：

只能用在循环语句的循环体内。在循环执行过程中，遇到continue语句，程序将结束本次循环，接着开始下一次循环。

【例2.18】从键盘上输入10个整数，若是正整数则求和，若是负整数则不进行计算，继续输入数据，若输入0则终止程序。

```
#include<iostream>
using namespace std;
int main()
{   int num,sum=0;
    cout<<"Please input number:"<<endl;
    for(int i=0; i<=9;i++)
    {   cin>>num;
        if(num= =0) break;
        if(num<0) continue;
        sum+=num;
    }
    cout<<"sum="<<sum<<endl;
    return 0;
}
```

2.6 案例实战

2.6.1 实战目标

■目标

（1）理解C++程序的顺序、选择和循环3种结构。

（2）熟练掌握常用选择语句和循环语句的使用。

（3）根据需求编写相应的程序，解决实际问题。

return

2.6.2 功能描述

- 案例名称：团购订单信息管理系统
- 功能说明：
  - ✓要求编写一个的菜单程序。
  - ✓菜单中包括对订单的添加、查询、修改、删除和浏览等功能。
  - ✓系统设有口令，只有正确输入口令才能使用该信息管理系统。

2.6.2 功能描述

■功能具体说明

- ✓**菜单的设计**：共设置6个选项，包括订单的添加、查询、修改、删除、浏览和退出。退出系统前菜单是重复循环的。
- ✓**订单信息的设计**：本案例采用简化形式，只定义了订单编号、商品编号、商品单价、商品数量、收件人姓名等。
- ✓**添加订单**：添加时订单的详细信息从键盘输入相应内容。
- ✓**浏览订单**：显示当前订单的所有信息，要求有格式控制。
- ✓**查询、修改、删除订单**：目前只针对一个订单进行操作，所以这3个选项不做任何操作，留待以后补充完善。
- ✓**口令设置**：被设为一个字符串常量。程序开始运行时，要求通过键盘输入口令。3次输入不正确，直接结束程序。

return