CHAPTER 3

第3章 C++语言初步

3.1 基本数据类型

3.1.1 数据类型概述

对数据的解释，我们称之为类型(type)。在程序设计中，数据类型有重要的意义，究其本质，数据类型是规范了不同数据在计算机内存储空间的大小与存储的具体形式。程序中所有的数据都属于特定的类型，数据的表示方式、取值范围以及对数据可以进行的操作都由数据所属的类型决定。为了让编译程序了解数据所属类型，在C++中遵循“先声明、后定义”的原则，这样在编译程序生成目标代码时可以知道需要分配多大的存储空间以及引用这个数据的规则。

在C++中，由于用户可以根据需要自定义类型，因此类型是非常丰富的。C++中提供的基本数据类型有四种，它们是字符型、整型、实型、空值型。另外，C++还允许用户自定义复合类型，即指针型、引用型、数组型和枚举型。值得一提的是，C++中允许用户自定义构造类型，如类、结构体和联合体等。这些类型中不同于C语言的主要是引用型与类，引用型将在有关变量与函数的讲解中详细说明，而整个面向对象的程序设计的学习中都将对“类”进行学习。

有关基本数据类型的说明如表3-1所示，其中数据长度以32位操作系统的微型机上使用的VC++ 6.0为准，其他版本可能有所不同，要参考相应的参考手册。

表3-1 C++基本数据类型

类型		说明	数据长度	值域
整型	(signed)int	(有符号)整型	4字节	$-2^{31}\sim2^{31}-1$
	unsigned int	无符号整型	4字节	$0\sim2^{32}-1$
	(signed)short	(有符号)短整型	2字节	$-2^{15}\sim2^{15}-1$
	unsigned short	无符号短整型	2字节	$0\sim2^{16}-1$
	(signed) long	(有符号)长整型	4字节	$-2^{31}\sim2^{31}-1$
	unsigned long	无符号长整型	4字节	$0\sim2^{32}-1$
字符型	char	字符型	1字节	0～255
	signed char	有符号字符型	1字节	−128～127
	unsigned char	无符号字符型	1字节	0～255

续表

类　型		说　明	数据长度	值　域
实型	float	单精度浮点型	4 字节	7 位有效数字
	double	双精度型	8 字节	15 位有效数字
	long double	长双精度型	8 字节	15 位有效数字
	void	空值类型	无	无

说明：

(1) 类型修饰符 signed 和 unsigned 用于修饰字符型和整型。

(2) 类型修饰符 short 和 long 用于修饰字符型和整型。

(3) 当用 signed 和 unsigned、short 和 long 修饰 int 整形时，int 可省略。

(4) 除表 3-1 以外，C/C++都可以自定义枚举 enum、联合 union 和 struct 结构体类型。

(5) 多数的 C++版本还提供 bool(布尔型)和 wchar_t(宽字符型)。

3.1.2 常用数据类型应用

1. 整数

C++的整数有 3 种常用的类型，short int 表示短整数，long int 表示长整数，int 表示一般整数，它们在计算机内占据的字节数与计算机软硬件环境有关。一般地，短整数占 2 个字节，长整数占 4 个字节，普通整数占 2 或 4 个字节。

在 C++中，使用的整数可以用八进制、十六进制和十进制 3 种方式表示。人们一般习惯使用十进制数表示，因为它符合人们的日常习惯，但有时也会用到八进制或十六进制数，因为它们与二进制数换算起来比较简单。

八进制数通常是无符号数，必须以 0 开头，即以 0 作为八进制数的前缀。如 012(十进制为 10)、0101(十进制为 65)、0177777(十进制为 65 535)都是合法的八进制数，而 125(无前缀 0)、0392(包含了非八进制数码 9)、－0123(出现了负号)都是不合法的八进制数。

十六进制数也表示无符号数，即无正负之分，前缀为 0X 或 0x，其数码取值为 0～9，A～F 或 a～f。如 0X2A(十进制为 42)、0XA0 (十进制为 160)、0XFFFF (十进制为 65 535)等都是合法的十六进制数，而 5A (无前缀 0X)、0X3H (含有非十六进制数码 H)、－0X12A (出现了负号)等不是合法的十六进制数。

整数在表示时，除了用前缀表示进制以外，还可以使用后缀。当要表示一个长整型数时，可以在该数的后面加上 L 或 l，如 12 表示普通整数，而 12L 表示长整数。而且，前缀，后缀可同时使用以表示各种类型的数。如 0XA5L 表示十六进制长整数 A5。

2. 实数

C++的实数有两种，即 float 和 double，分别称为浮点型和双精度型，它们的区别在于所表示的数据范围与精度不同，在内存中所占的字节数也不同。实型的常数默认为是 double 的，如果想表示 float 的常数，则须在数字后面加上 F 或 f，如 12.3f 是浮点数，而 12.3 是双精度数。

C++中的实数有两种描述形式，一种是定点数形式，如 12.3，另一种是指数形式，如 1.23E20 或 1.23e20，表示 1.23×10^{20}，使用时注意，e 或 E 的前面必须有数字，即使是 1，也不能缺省不写，后面的值则必须为整数。

3. 字符型

在内存中，字符数据以 ASCII 码的形式存储，在一定范围内可以与整数混用，但含义有所不同。ASCII 码是一种给字符编码的的国际标准，它以 0～255 的整数表示字符，比如十进制数 65，表示'A'。尽管整数与字符可以在一定范围内相互赋值，但含义不同，字符可以是字符集中任意字符。但数字被定义为字符型之后就不再有原来数值的意义，如'5'和 5 是不同的。'5'是字符，不是整数 5，是对应'5'的 ASCII 码值，即 53。关于字符与整数的理解，请仔细体会下面的示例 3-1。

【示例 3-1】 字符与整数混用示例。

```
#include <iostream.h>

void main()
{
    int i = 65;
    char c = i;
    cout <<"The value of integer: "<< i << endl;
    cout <<"The value of character: "<< c << endl;
}
```

该程序的执行结果如下。

```
The value of integer: 65
The value of character: A
```

在 C++中，有些特殊字符用转义字符表示，转义字符以反斜线"\"开头，后跟一个或几个字符。转义字符具有特定的含义，不同于字符原有的意义，故称“转义”字符，表 3-2 列出了常用的特殊字符。

表 3-2 常用的转义字符及其含义

转 义 字 符	转义字符的意义	ASCII 代码
\0	空字符	0
\n	回车换行	10
\t	横向跳到下一制表位置	9
\b	退格	8
\r	回车	13
\f	走纸换页	12
\\	反斜线符"\"	92
\'	单引号符	39
\”	双引号符	34
\a	鸣铃	7
\ddd	1～3 位八进制数所代表的字符	
\xhh	1～2 位十六进制数所代表的字符	

表中的\ddd 和\xhh 正是为此而提出的。ddd 和 hh 分别为八进制和十六进制的 ASCII 代码。如\101 表示字母 A,\134 表示反斜线,\XOA 表示换行等。转义字符在输出中有许多应用,如想让计算机的扬声器发出铃声,可以使用下面的语句。

```
cout <<'\a';        //程序运行时会发出"叮"的一声
```

如希望在屏幕上输出以下内容:

```
She said: "How are you! "
```

就不能用以下语句:

```
cout <<"She said: "How are you! "";
```

这是因为,双引号在 C++中是有特殊作用的,上述写法会使 C++编译器产生错误,应该使用下面的语句:

```
cout <<"She said: \"How are you! \"";
```

4. 字符串型

字符串是由一对双引号括起的字符序列。例如,""、"CHINA"、"C++ program"和"$12.5"等都是合法的字符串,其中第一个是空串。

字符串和字符是不同的量。它们之间主要有以下区别。

- 字符由单引号括起来,字符串由双引号括起来;
- 字符只能是单个字符,字符串则可以含零个或多个字符;
- 字符型数据一般占一个字节的内存空间。字符串占的内存字节数等于字符串中字节数加 1。增加的一个字节中存放字符"\0"(ASCII 码为 0),这是字符串结束的标志。如字符串"CHINA"在内存中所占的字节如下。

C	H	I	N	A	\0

字符'a'和字符串"a"虽然都只有一个字符,但在内存中的情况是不同的。

'a'在内存中占一个字节,可表示如下。

a

"a"在内存中占二个字节,可表示如下。

a	\0

3.1.3 保留字与标识符

保留字是指一些具有专门的意义和作用的单词,在 C++中这些保留字都是小写的,表 3-3 列出了 C++中的所有保留字。

现实世界中每个事物都有自己的名字,从而与其他事物区分开来。在程序设计语言中,要对程序中各个元素加以命名,这种用来标识变量、常量、自定义类型、函数和标号等元素的记号称为标识符。C++中标识符的 BNF(Backus-Naur Form,巴科斯范式)定义如下。

表 3-3 C++的保留字

asm	auto	break	case	catch	char
class	const	continue	default	delete	do
double	else	enum	extern	float	for
friend	goto	if	inline	int	long
new	operator	private	protected	public	register
return	short	signed	sizeof	static	struct
switch	template	this	throw	try	typedef
union	unsigned	virtual	void	volatile	while

```
标识符 ::= 字母{字母|数字|下划线}|下划线{字母|数字|下划线}
  字母 ::= A|B|…|Z|a|b|…|z
  数字 ::= 0|1|…|9
下划线 ::= _
```

根据上述定义，程序员在 C++中的命名应遵循如下原则。

(1) 名称是由英文字母、下划线开头，由英文字母、数字、下划线组成的字符序列。

(2) C++中区分大小写。

(3) 名称不能是 C++中的保留字。

(4) 命名时应遵循“见名知义”的原则。

(5) 不同编译环境对标识符能识别的长度互不相同，为增强程序的可移植性，一般不要定义过长名字的标识符，通常应在 16 个字符以内。

3.1.4 变量

在程序执行过程中，其值可以改变的量称为变量。在 C++中，并不要求所有的变量都在过程的最前面定义，但每个变量都要在使用之前先进行定义，这个定义用来说明变量的存储类型、数据类型、变量名以及变量的初值。变量定义的标准格式如下。

```
[存储类型]  数据类型  变量名[=初值表达式];
```

其中存储类型的定义有 4 种，它们是默认的自动型(auto)、寄存器型(register)、静态(static)与外部型(extern)，存储类型描述了变量在内存中的存储特性。数据类型描述了变量在内存中占据空间的大小。变量名是变量的标识，是由英文字母、数字和下划线组成且由字母或下划线开头的字符序列，C++中的标识符是区分大小写的，命名时一般要见名知义，这样可以提高程序的可读性。而初值表达式在 C++中并不一定是常数表达式，它可以出现常量、已定义过的变量和函数等。

在变量中，不同于 C 语言的是引用型变量。那么什么是引用呢？简单地说，引用就是给一个变量起个别名，两个名字对应同一个地址，这使变量与它的引用总是具有相同的值。引用型变量定义的格式如下。

```
数据类型名    &引用变量名=被引用变量名
```

例如：

```
int a = 5;  int &a1 = a;
```

这时就称 $a1$ 是 a 的引用变量，a 是 $a1$ 的被引用变量，$a1$ 与 a 共用同一块内存，因此 $a1$ 的值也是 5。在使用引用型变量的时候应注意以下几个问题。

(1) 被引用的变量必须已定义且有初值。

(2) VC++中也允许做引用的引用，如“int a=5; int &a1=a; int &a2=a1;”。

(3) 在 VC++中，引用变量与被引用变量必为相同类型。

【示例 3-2】 引用型变量的定义与使用示例。

```
#include <iostream.h>
void main( )
   {int a = 1;
   int & a1 = a;
   cout <<"a = "<< a <<"    a1 = "<< a1 << endl;
   a1 = 10;
   cout <<"a = "<< a <<"    a1 = "<< a1 << endl;
   }
```

运行结果如下：

```
a = 1   a1 = 1
a = 10   a1 = 10
```

从这个例子中可以看出，变量与它的引用总是具有相同的值，无论程序中改变了谁的值，另一个都会跟着变。但是，在一个程序中像 a 这样的普通变量有两个名字，这显然不是必须的，在多数情况下也是不必要的，而且这还会降低程序的可读性。那么为什么还要引入引用型变量呢？这是因为引用型的形式参数在函数中可以实现“变参”效果，返回为引用的函数可使函数出现在等号的左边，这大大地弥补了 C 语言只有值形参的缺点，关于这些内容本书将在函数一节中介绍。

3.1.5 常量

在程序执行过程中，其值不能改变的量称为常量。普通常量的类型是根据数据的书写形式来决定的。例如，2 和 600 是整型的，0.1 和 2e－5 等是实型的，'d'和'1'等是字符型的。而"abcd"和"a＝123\n"等用双引号引起来的字符序列称为字符串型的，在 C++中也可以像 C 语言中一样去使用转义字符。

有一种特殊的常量是用标识符表示的，称为符号常量。符号常量主要用于帮助记忆和提高程序的可读性与可维护性。例如，程序中经常用到圆周率，假设定为 3.14，那么如果想提高圆周率的精度到 3.14159 时，它在程序中的每次出现都要进行修改，如果这里用 pi 表示这个圆周率，每次用时都写 pi，则要改变 pi 的精度就只须改变 pi 的初值就可以了。原来在 C 语言中用编译预处理语句＃define 进行符号常量的定义，由于＃define 是一种简单的机械替代，这种机械替代不进行任何类型检查与计算，因此，有时会带来意想不到的副效应。由于 C++是面向 C 语言全兼容的，因而在 C++中也可以用＃define 进行符号常量的定义，但为避免＃define 的副效应，一般使用 const 语句定义。这两种符号常量的主要区别如下。

- const 定义的符号常量是在执行时起作用，而 define 的定义是在编译时起作用；

• const 说明的量有实际内存，可以定义指针；

• const 进行类型检查与表达式计算，减少编译错误，从而避免 define 的副效应。

(1) 用 const 定义符号常量的基本格式如下。

const 数据类型 常量名 = 初值表达式

例 1：定义 pi 为 3.14159。

```
const  float  pi = 3.14159;
```

如果用 # define 定义，则格式如下。

```
#define  pi  3.14159
```

(2) const 还可以与指针连用，主要有两种方式：

例 2：定义一个指向常量的指针。

```
const  int  *p;
```

注意：这里的 *p* 实际上是变量，但这个指针只能用于指向由 const 定义的符号常量。这种情况可以不定义初值，*p* 可以指向不同的符号常量，常用于定义专门指向字符串常量的指针，如：

```
const  char  *P;
```

例 3：定义一个指针型符号常量。

```
char  c[10] = "ABC";
char  *const  p = c;
```

这种定义 *p* 将指向一个固定地址，即为地址常量，可以用来指向变量地址，也可以用于指向数组首地址。

(3) const 还可以与引用连用。

例 4：定义一个 const 型的引用。

```
int  va = 3;
const  int  ca = 2;
const  int  &c1 = va,  &c2 = ca;
```

在 VC++中，const 型常量的引用一定是 const 的，如 ca 的引用只能定义成常量型引用，而不能定义成“int &c2＝ca;”；但普通变量也可以定义为 const 型的引用，如 va 的引用 *c*1 就可以定义成 const 型的。在这里，被 const 说明过的量 *c*1 与 *c*2 是不允许被赋值的，不论被引用的对象是否为常量，*c*1 与 *c*2 都当成常量使用。

【示例 3-3】 const 应用综合实例。

```
#include < iostream.h >
#include < string.h >
void main( )
{const int a1 = 1,a2 = 2;
 const int *pin;
```

```
char c1[10] = "ABC";
char * const pch = c1;
pin = &a1;
cout << * pin << endl;pin = &a2;
cout << * pin << endl;
cout << c1 << endl << pch << endl;
strcpy(c1,"XYZ");
cout << c1 << endl << pch << endl;
 }
```

运行结果如下：

```
1
2
ABC
ABC
XYZ
XYZ
```

3.1.6 构造数据类型

C++的构造类型主要有数组、枚举、结构与类，关于“类”的应用将在第 4 章以后重点介绍，本节简单介绍其他 3 种构造类型。

1. 数组

有时，人们想在程序中表示许多类型相同的数据，例如，想记录一个班级每个人的成绩，假如这个班级有 30 人，那是否要定义 30 个不相关的整型变量呢？当然不能，那样会非常麻烦，C++提供了一种简单的方法，可以用数组来表示，数组变量可以存放一组具有相同类型的数据，定义数组的语法格式如下。

数组类型　数组名[数组元素个数];

其中，数据类型可以是简单数据类型，也可以是构造类型。例如，上面所说的 30 个同学的成绩数据，可定义如下。

```
int grade[30];
```

可以用数组名加下标的方式来访问数组中的每个元素，下标的范围是从 0 到元素个数减 1，如 grade[0]表示数组中第 1 个元素，即第 1 个同学的成绩，grade[29]表示数组中第 30 个元素。下面的语句可用于输出第 5 个同学的成绩。

```
cout << grade[4];
```

字符数组是一种常用的数组，即数组中的每个元素都是字符。事实上，C++中常用字符数组描述字符串变量，例如，可定义如下字符型数组。

```
char str1[6] = "China";
```

这样的定义是说明，str1 数组有 6 个存储空间，除了存储字符串中的 5 个字符以外，还在最

后存储了'\0'这个字符串结束标志。有了上述定义，可以用 str1[2]表示字符'i'，用 str1 表示整个字符串，例如：

```
cout << str1 << endl;
```

这个语句就可直接输出 str1 整个字符串。

当人们去看电影的时候，电影票上会写上“几排几号”，此时观众会首先根据“几排”找到所在的行，然后根据“几号”找到具体位置。这样的数据可用二维数组来描述。一个二维数组的定义方式如下。

```
数组类型  数组名[行数(常量表达式)][列数(常量表达式)];
```

例如，定义一个 8 行 9 列的二维整型数组 a，写成如下形式。

```
int a[8][9];
```

其中行号范围是 0～7，列号范围是 0～8，共描述 72 个数组元素，如同一维数组一样，可以用数组名加下标的方式来使用数组元素，如 a[2][2]表示第 3 行第 3 列的元素。数组元素在内存中是按顺序存放的，如 a 数组的存放方式如图 3-1 所示。

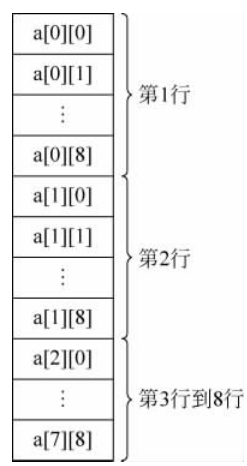

图 3-1 a 数组存放形式

类似地，人们还可以定义多维数组，定义与存储的方法，与二维数组相似，不再赘述。

2. 枚举

表示一星期中的某一天时，有 7 种取值：Sunday、Monday、…、Saturday。当然，人们可以用整数来表示这种数据：

```
int Weekday;      //Weekday 取值范围为 0～6
```

这种表示有两个缺点，一是不直观，二是容易出错。如果给 weekday 赋值为 10，在编译上不会有错误，但这明显有问题，会让程序在执行过程中出现错误。在 C++中，可用“枚举”的数据类型来解决这个问题。枚举的使用很像一个整数类型，定义具有以下形式：

```
enum 枚举类型名 {常量 1, 常量 2, …, 常量 n};
```

它表示定义一种枚举的数据类型，具有这个数据类型的变量所有可以取的值都列在后面的大括号中。比如定义描述星期的枚举类型如下：

```
enum Weekday {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
```

然后，可以用枚举类型 Weekday 定义变量：

```
Weekday week;
```

这时，变量 week 的取值就有 7 种，即 Sunday、Monday、…、Saturday。如果人们希望变量 week 表示星期六，则可以给 week 赋值：

```
week = Saturday;
```

枚举类型中的每个元素的值实际上都是整数，枚举类型本质上说，也就是一个整数的集

合，默认下，第一个枚举元素被赋值 0，接下来的枚举元素取值是前面一个枚举元素的取值＋1。有时候我们希望显式地指定某个枚举元素的值，例如：

```
enum some_fruit {apple = 3, orange, banana = 4, bear};
```

这样，apple 值为 3、banana 值为 4；那么 orange 和 bear 呢？前面说过，默认下“接下来的枚举元素取值是前面一个枚举元素的取值＋1”。既然这两个枚举元素没有显式赋值，那么就按照默认规则，orange 取值为 4，bear 取值为 5。从这个例子也可以看出，同一枚举中枚举子的取值不需要唯一。

由于枚举类型的值实际上是整数，所以可以将枚举类型变量的值直接赋值给整型变量，但是反过来不能，因为枚举变量的取值范围是有限的，如果整型量的取值超出枚举量的范围则是没有意义的。

一般地，枚举类型主要适用于表示少量状态的情况，比如描述星期、月份、是与否、男或女等等，使用枚举型量可以大提高程序的可读性，在实际工作中可以适当地使用。

3. 结构

结构体是 C++中比较常用的数据类型，往往由一个数据的不同方面信息共同组成。例如，图书馆要记住馆中的藏书信息、读者信息、借书信息，而藏书信息包括书的编号、书名、作者和页数等，读者信息包括读者的借书证号、读都名称、职业和年龄等，借书信息包括借阅人的借书证号、借书编号和借书时间等。如果我们要在程序中记录所有这些信息，则需要将隶属于同一类数据的相关信息组合起来，以便与其他信息区分开来，这就需要使用结构(struct)来达到这一目的。如藏书信息就可以表示为如下。

```
struct Book
{
   int book_no;                    //书的编号
   int page_num;                   //书的页数
   char author_name[30];           //作者姓名
   char book_name[60];             //书名
};
```

定义了结构体类型后，在表示具体的一本书时，就可以定义这个结构类型的变量，如：

```
Book a_book;
```

为这本书赋值时，可以为结构的各个组成部分分别赋值，如：

```
a_book.book_no = 112233;
a_book.page_num = 301;
```

为使某个结构变量有初始值，可以在定义这个变量时直接为它初始化，例如：

```
Book a_book = {112233, 301, "马慧彬", "C++程序设计"};
```

组成结构的各个成员的数据类型可以是任意简单数据类型，也可以是构造数据类型。另外，结构在 C++中的使用，有许多地方与类的使用相似，这将在后面介绍类的时候再详细说明。

3.2 表达式与类型转换

3.2.1 表达式

表达式是由操作符与运算符按一定的语法形式组成的符号序列。每个表达式经过运算后都会有一个确定的值，并且这个值一定属于某一个特定类型，这个类型也称为表达式的类型。表达式的求值次序取决于表达式中各种运算符的优先级与结合性。C++语言规定的部分运算符优先级与结合性如表 3-4 所示。

表 3-4 C++语言运算符优先级与结合性

优 先 级	运 算 符	结 合 性
1	()	左结合
2	!、～、++、--、+(取正)-(取负)sizeof	右结合
3	*、/、%	左结合
4	+、-	左结合
5	<<、>>	左结合
6	<、<=、>、>=	左结合
7	==、!=	左结合
8	&	左结合
9	^	左结合
10	|	左结合
11	&&	左结合
12	||	左结合
13	?：	右结合
14	=、+=、-=、*=、/=、%=、<<=、>>=、&=、^=、|=	右结合
15	,	左结合

当表达式中出现不同类型的操作数时，为对这个表达式进行求值，需要将表达式中的部分操作数进行类型转换，以保证二元运算符两边的操作数的类型一致。转换可以是两种方式，即自动类型转换与强制类型转换。

3.2.2 自动类型转换

自动类型转换是由编译器在编译时自动完成的，在 C++中规定了如下几个自动类型转换规则。

(1) 二元算术运算中的自动转换是一步一步进行的，以较高类型为准进行转换。所谓较高类型指存储空间较大者，C++中基本数据类型的类型级别如图 3-2 所示。

(2) 赋值运算的自动转换以赋值号左侧的变量类型为准进行转换。

(3) 在函数中，实在参数类型以形式参数类型

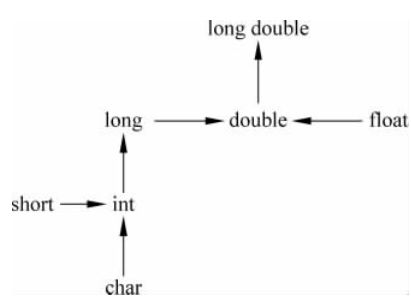

图 3-2 C++中基本数据类型的类型级别

为准进行转换，函数返回值类型以函数类型为准进行转换。

(4) 关于对象的类型转换有两种方式，若在类中重载了某类型符，可以进行将对象转换为某类型的转换，若在类中定义了关于某类型的构造函数，可以进行将某类型转换为对象的类型。具体转换方式在后续章节中介绍。

3.2.3 强制类型转换

在程序中，人们也可以显式地进行类型转换，在任何表达式中都可以将某个量强制转换成所需要的类型。强制类型转换的格式如下。

类型名(表达式)　或　(类型名)表达式

例如：

```
cout << double(5)/8 <<'\t'<< 5/ (double) 8 <<'\t'<< 5/8 <<'\t'<< double(5/8)<< endl;
```

的输出结果为

```
0.625   0.625   0       0
```

下面分析一下这个语句。

double(5)/8 表达式中首先进行了强制类型转换，这样/的前面是“double”类型，后面是“int”类型，这时要进行自动的类型转换，将分母 8 转换成 double 类型，然后进行运算，结果是 double 型的。5/(double)8 表达式与第一个输出项相似；而 5/8 这个式子，由于二者都是 int 型的，因此没有类型转换发生，结果是 int 型的。最后一项的 double(5/8)表达式虽然进行了强制类型转换，但这是对 5/8 的结果 0 进行强制转换，结果虽为 double 类型，但值为 0.0，显示为 0。

关于类型转换还应注意以下问题。

(1) 类型转换是一步一步进行的，而不是整个表达式统一只做一次转换。

(2) 类型转换的结果是临时的，参与运算后这个临时的值就会销毁，参与类型转换的变量并不会因此改变它原有类型。

(3) 类型转换占用系统时间，并且有时的自动转换会带来负效应，因此应谨慎使用类型转换。

3.3 指　　针

3.3.1 指针的定义与使用

在路上行走时，经常会看到路牌，按照路牌所指示的方向，人们就会很容易地找到目的地。在 C++中，指针就起到路牌的作用，它指向了另一个内存地址，这个地址中就存放了人们想要的东西。

1. 指针的定义与初始化

先看下列定义。

```
int *pi;                    //定义一个指向整型空间的指针
double *pd1, *pd2;          //定义两个指向双精度型数据的指针
char *str;                  //定义一个指向字符空间的指针,常用来定义字符串的首地址
```

这里,pi、pd1、pd2、str都是指针类型的变量,*表示定义的是指针,而不是一般的变量,int、double、char是指针所指向的内存空间中数据的类型,int *、double *、char *看成是指针数据类型,用来定义pi、pd1、pd2、str这样的指针变量。指针的值是另外一个变量的内存地址,比如,可以定义一个整型变量,然后,将这个整型变量的地址赋给一个整型指针变量,定义如下:

```
int i;
int* pi = &i;
```

这里在定义指针变量pi时,为它初始化值为&i,& 是C++的一个单目运算符,在这里表示取得跟在它后面的变量的内存地址,因此,这样的定义就是让pi的值是 i 的地址,一般称ip指向 i。

2. 指针的赋值与使用

需要注意的是,在给一个指针赋值时,总是给与其类型对应的内存地址,即整型指针只能被赋给整型量的地址,而不能将一个字符变量的地址赋值给一个整型指针,同样地,也不能将一个整型变量的地址赋值给一个浮点型指针,更不能将一个不是地址的值赋值给指针。例如,下面为指针初始化的语句是不正确的。

```
float f1;
char ch1;
int* pi1 = &f1;             //错误,不能将浮点变量的地址赋值给整型指针
float* pf = &ch1;           //错误,不能将字符型变量的地址赋值给浮点型指针
char* str = 'a';            //错误,常数不是内存地址值
int* pi2 = &200;            //错误,不能对常数进行取地址操作
```

由于指针也是一个变量,因此,它的值是可以改变的,可以将原本指向 i 的指针改成指向j,用下面的语句可以实现。

```
int i,j;
int *pi = &i;
⋮
pi = &j;
```

这里,在使用pi时,不要写成*pi。

可以通过指针去找到一个指针所指示的空间,从而对其所指的变量进行操作。*用在指针变量之前,表示对指针所指的变量空间的引用,可以通过指针来间接地改变所指变量的值。例如:

```
int i = 2;
int *pi = &i;               //指针 pi 指向变量 i
⋮
*pi = 5;                    //等价于 i = 5;
cout << *pi;                //等价于 cout << i;
```

要注意的是，指针变量使用时的 * 与定义时的 * 有本质的区别，使用时的 * 单目运算符，表示对指针所指变量的引用，而定义时的 * ，表示它是指针类型的变量。

【示例 3-4】 理解指针。

```
#include <iostream.h>
void main()
{
    int i = 4,j = 10;
    int * pi = &i;

    cout <<"pi = &i"<< endl;
    cout <<"i = "<< i <<"    &i = "<< &i << endl;
    cout <<"pi = "<< pi <<"    &pi = "<< &pi <<"    * pi = "<< * pi << endl;

    pi = &j;
    cout << endl <<"pi = &j"<< endl;
    cout <<"i = "<< i <<"    &i = "<< &i << endl;
    cout <<"j = "<< j <<"    &j = "<< &j << endl;
    cout <<"pi = "<< pi <<"    &pi = "<< &pi <<"    * pi = "<< * pi << endl;

    * pi = 20;
    cout << endl <<" * pi = 20"<< endl;
    cout <<"i = "<< i <<"    &i = "<< &i << endl;
    cout <<"j = "<< j <<"    &j = "<< &j << endl;
    cout <<"pi = "<< pi <<"    &pi = "<< &pi <<"    * pi = "<< * pi << endl;
}
```

该程序运行结果如下：

```
pi = &i
i = 4    &i = 0x0018FF44
pi = 0x0018FF44    &pi = 0x0018FF3C    * pi = 4

pi = &j
i = 4    &i = 0x0018FF44
j = 10    &j = 0x0018FF40
pi = 0x0018FF40    &pi = 0x0018FF3C    * pi = 10

* pi = 20
i = 4    &i = 0x0018FF44
j = 20    &j = 0x0018FF40
pi = 0x0018FF40    &pi = 0x0018FF3C    * pi = 20
```

结果分析：该程序定义了 3 个变量，其中 i、j 是整型变量，pi 是指向 i 的指针。程序输出分成 3 个部分，每个部分输出 i、j、pi 的信息，可以看出，3 个变量的内存地址是连在一起的。从第一部分输出结果可看出，pi 的值是 i 的地址，而 * pi 的值是 i 的值；从第二部分输出结果可看出，当 pi 指向 j 以后，pi 的值是 j 的地址，* pi 的值是 j 的值；从第三部分输出结果可看出，当为 * pi 赋值 20 以后，j 的值改为 20，可见，对 * pi 操作，与对 pi 所指的变量 j 做操作是一样的。

3. void 指针

有一种特殊类型的指针，这种指针指向的数据类型是 void，比如：

```
void * pv;
```

这表示它只是一个指针，可以存放一个内存地址，但是这个内存地址所指向的空间中存放的数据的类型还不能确定，这需要到使用这个指针时再确定具体的类型，确定的方法是对指针进行强制类型转换，例如：

```
int a = 20;
void * pv = &a;
⋮
int * pi = (int * )pv;          //强制将(void * )转换成(int * )
cout << * pi << endl;
cout << * ((int * )pv);         //强制将(void * )转换成(int * )
```

void 指针经常用在能支持多种数据类型的数据操作的函数中，在 C++的库函数中经常使用，读者可以在以后的深入学习中逐步体会这种指针的用法。

4. NULL 指针值

当一个指针定义时没有为其初始化，且使用前没有为其赋值，那么这个指针的值是随机的，可以指向一个随机的内存，如果这时不小心使用了这个指针，很容易出现非常危险的异常，为了避免这种情况，当一个指针不指向任何内存地址时，常用 NULL 作为指针的值，这样，在使用指针前，也常常判断一下这个指针的值是否为 NULL，如果不是，说明该指针指向了一个合法的空间，可以进行操作，如果是，说明该指针没有具体指向。例如：

```
int * pi = NULL;                //将指针初始化为 NULL
//…中间可能执行很多语句，可能包括对指针的赋值等操作
if(pi!= NULL)                   //判断指针是否有具体指向，如果有，则可以使用指针
//访问 * pi
```

NULL 在动态内存申请和构建链表等操作时都会起很大作用。

3.3.2 指针与数组

数组名本质上说是一个地址常量，这个地址是数组第一个元素的内存地址，因此，可以将数组名赋值给一个指针变量，当指针指向一个数组时，指针也可以当成数组名来使用，可以像数组一样，通过指针去访问数组的各个元素。

例如，定义了一个数组和一个指向数组第一个元素的指针：

```
int a[50];
int * pa = a;                   //数组名本身就是地址，不要用 &a
```

有了这样的定义，a 和 pa 的用法就完全可以互换，如果要访问数组的第一个元素，a[0]、* a、pa[0]、* pa，这 4 种用法都是正确的，pa 与 a 一样，也可以看成是数组的名字，访问数组的第 $i+1$ 个元素，可以用 a[i]，也可以用 pa[i]。a 是地址常量，不能被赋其他地址的

值，而 pa 是变量，可以被重新赋地址值，除了这一点以外，可以认为 pa 与 a 是等价的，示例 3-5 可以验证这一点。

【示例 3-5】 指向数组的指针用法。

```
#include < iostream.h >
void main()
{ int a[5];
  int * pa = a;
  for(int i = 0;i < 5;i++){
      a[i] = i;
      cout << pa[i]<<"     "<< a[i]<< endl;
  }
}
```

运行结果如下：

```
0    0
1    1
2    2
3    3
4    4
```

3.3.3 指针运算

当指针指向一个数组以后，指针可以通过与整数做加或减运算实现在数组中的移动，加 1 就向后移动一个元素，减 1 就向前移动一个元素。指针是一个内存地址，所以当一个指针变量与一个整数相加或相减以后，得到的还是一个内存地址，还可以将其当成指针看待，例如，pa 指向数组 a 的首地址，那么 pa＋i 就是数组中第 i＋1 个元素的地址，因此，*（pa＋i）就是第 i＋1 个元素本身，也等价于 a[i]和 pa[i]。

注意：指针的运算并不是简单的数值的相加，指针的加和减是以指针所指的数据类型的大小为单位进行的。如 pa 指向整型数组 a 的首地址，而整数数组 a 的每一个元素的大小为 4 个字节，则指针 pa 的加减操作就是以 4 个字节为单位进行的，如果第 1 个元素的地址为 0x0018FF34，pa＋1 的结果是地址 0x0018FF38 而不是 0x0018FF35；同样，pa＋4 的结果是 0x0018FF34＋(4＊4)＝ 0x0018FF44，而不是 0x0018FF38，示例 3-6 可以验证这一点。

【示例 3-6】 指针运算是以所指数据类型大小为单位的。

```
#include < iostream.h >
void main()
{
    int a[5];
    int * pa = a;
    for(int i = 0; i < 5;i++)
        cout <<"pa + "<< i <<" = "<< pa + i << endl;
}
```

运行结果如下：

```
pa + 0 = 0x0018FF34
```

```
pa + 1 = 0x0018FF38
pa + 2 = 0x0018FF3C
pa + 3 = 0x0018FF40
pa + 4 = 0x0018FF44
```

当指针指向数组的元素时，通过指针加减运算来移动指针，可以方便地访问数组元素，如示例 3-7。

【示例 3-7】 数组 a 随机产生 10 个 100 以内的整数，pa 指向 a，利用指针运算求数组的最大元素。

```
#include < iostream. h >
#include < stdlib. h >
void main( )
{
    int a[10];
    int * pa = a;
    int max;
    int i;

    for(i = 0;i < 10;i++){
        a[i] = rand( ) % 100;
        cout << a[i]<< endl;
    }

    max = * pa;
    for(i = 1;i < 10;i++)
        if( * (pa + i)> max)
            max = * (pa + i);
    cout <<"\nmax = "<< max << endl;
}
```

另外，要注意的是，同数组下标不能超出范围一样，指针的加减操作也要慎重，不能超过它所指的数组内存范围，如例 3-7 中，pa 所指的 a 数组只有 10 个元素，那么 pa＋20 的操作就是错误的，因为它超过了数组的内存范围。

3.3.4 动态内存申请

在 C 语言中用于动态内存管理的语句主要是使用 malloc( )与 free( )函数，它们一般来自 alloc. h 函数库，当然在 C++中也可以这样做。但在 C++中更常用的不是函数，而是用 new 和 delete 这样的操作符，它们一般配套使用，new 表示从内存中申请一块空间，delete 表示将申请的空间返还给系统。

1. 申请内存 new

new 有下列 3 种格式。

```
new 数据类型;
new 数据类型(初始值);
new 数据类型[常量表达式];
```

其中，第一种格式是申请一个指定数据类型的内存空间，可以将操作结果赋给一个相应数据类型的指针变量，例如：

```
int *pi = new int;              //申请一个整型空间并赋给整型指针变量
int *pf = new float;            //申请一个浮点型空间并赋给浮点型指针变量
⋮
*pi = 10;                       //通过指针可以访问所申请的空间
*pf = 0.01;
```

new 的第二种形式是在申请空间的同时，为空间赋初始值，例如：

```
int *pi = new int(25);
cout<<*pi;                      //输出结果为 25
```

new 的第三种形式是动态申请数组空间，方括号内的常量表达式代表申请空间的大小，这个空间大小是以数据类型大小为单位的，例如，要申请一个能放 25 个整型量的数组，然后赋值给一个指针，格式如下。

```
int *arr = new [25];
```

要注意 int *p=new int[25]与 int *p=new int(25)的区别，前者表示长度，后者表示长度缺省，圆括号中的数据为申请的那个单元的初值。

由于系统资源是有限的，所以，不是任何情况下都能申请到足够的空间，如果因内存不足等异常情况申请空间失败时，将返回 NULL 指针，即返回 0 值。由于内存操作失败是非常危险的，因此，在申请内存时，一般都要判断一下，申请是否成功，如果成功就继续工作；如果失败，马上抛出异常或直接停止工作结束程序运行，比如：

```
int *arr;
arr = new int[100];
if(arr == NULL){                //由于 NULL 的值是 0,因此,这个语句也可写成 if(!arr)
   cout<<"申请内存失败"<<endl;
   exit(0);                     //强制结束程序运行
}
//…继续其他工作
```

2. 释放内存 delete

当不再需要所申请的空间时，需要把这个空间返还给系统。delete 是释放空间的操作符，使用格式有下列两种。

```
delete 指针名;
delete [ ]指针名;
```

这两种格式分别与 new 两种形式相对应，当 new 使用前两种形式时，即申请的是一个空间时，对应的 delete 为第一种格式，也就是只释放指针所指的空间，如：

```
int *pi = new int;
int *pf = new float(0.01);
//使用 pi 与 pf
delete pi;                      //释放 pi
```

```
delete pf;                    //释放 pf
```

如果 new 使用第三种形式申请内存，即申请的是动态数组，则 delete 使用第二种格式，也就是将数组空间全部释放，如：

```
int * arr = new [25];
//使用 arr
delete [ ]arr;
```

使用 new 与 delete 时要注意四点：一是对一个空间操作与对数组空间操作的格式是有区别的；二是如果在程序中用 new 申请空间了，在程序结束前一定要用 delete 释放，避免空间泄漏，使内存得到有效利用；三是 NULL 指针是不能释放的，即用 new 申请如果失败，就不能再释放了；四是一个用 new 申请的指针只能释放一次，不能重复释放。

【示例 3-8】 new 与 delete 的一般用法。

```
#include < iostream. h >
void main( )
{ int * p;
  p = new int;                //申请一个整型单元
   * p = 25;                  //这两个语句可以写成 p = new int(25);
  cout << * p;
  delete  p;                  //释放 p 所指的单元
  }
```

【示例 3-9】 指针应用综合实例。从键盘读入学生的人数，根据学生数申请空间来存放学生信息，从键盘读入所有学生的姓名与成绩，计算并输出学生的平均成绩。

```
#include < iostream. h >
#include < stdlib. h >

struct Student
{
    char name[20];
    float score;
};

void main( )
{
    Student * stus;
    int stunum;
    int i;

    //读入学生数
    cout <<"读入学生数: ";
    cin >> stunum;

    //申请学生信息存储空间
    stus = new Student[stunum];
    if (stus == NULL){
        cout <<"内存申请失败!"<< endl;
```

```
        exit(0);
    }

    //读入学生信息
    cout <<"输入学生信息,名字和成绩"<< endl;
    for(i = 0;i < stunum;i++){
        cout <<"输入第"<< i + 1 <<"号学生信息: ";
        cin >> stus[i].name >> stus[i].score;
    }

    //计算并输出平均成绩
    float sum = 0;
    for (i = 0;i < stunum;i++)
        sum = stus[i].score + sum;
    cout <<"\n 学生平均成绩: "<< sum/stunum << endl;

    //释放空间
    delete []stus;
}
```

输出结果如下:

```
读入学生数: 3
输入学生信息,名字和成绩
输入第 1 号学生信息: mm 50
输入第 2 号学生信息: nn 60
输入第 3 号学生信息: bb 70

学生平均成绩: 60
```

3.4 基本控制结构

控制结构是通过控制语句实现的,除了顺序结构不需要特殊的语句外,控制语句主要有选择结构语句与循环结构语句。

3.4.1 简单的输入/输出语句

在 C 语言中输入/输出是由函数 scanf( )与 printf( )实现的,在 C++中仍然可以采用 C 语言的格式,这两个函数被包含在 stdio.h 函数库中。但是,这样对不同类型的数据采用不同的控制字符,让人很难记住,C++在输入/输出方面做了很大的改进。在 C++中更常用的输入/输出方式是采用“流”来实现。关于“流”,本书将在第 8 章详细介绍。现在只须知道,C++是自带输入/输出的,并且可以根据数据的类型自动使用合适的输出方式。

【示例 3-10】 一个简单的输入/输出程序。

```
#include < iostream.h >
void main( )
{char name[10];
 cout <<"请输入姓名:";
```

```
 cin  >> name;
 cout <<"您好,"<< name <<'!'<< endl;
}
```

程序中 cout 与 cin 都是在 iostream.h 库中定义的操作符,因此在使用前应将这个库文件用#include 包含进去。<<与>>表示数据的流动方向,“cout<<”表示将后面的内容送到标准输出设备,一般指显示器,“cin>>”把从标准输入设备中获取的数据存入后面的变量,标准输入设备一般指键盘。

输入/输出语句格式如下。

输出:

```
cout << … << … ;
```

输入:

```
cin >> … .>> … ;
```

输入/输出语句的特点如下。

- cout 和 cin 可以当作是操作符使用,不是 printf()和 scanf()那样的函数;
- cout 和 cin 等“流”的操作都来自“iostream.h”函数库;
- 也可以使用转义字符,如\n、\t 等,含义与 C 语言中相同;
- endl 用在 cout 中表示换行。

3.4.2 条件语句

1. if 语句

if 语句用于在程序中有条件地执行某一语句序列,语句的格式如下。

```
if (条件表达式)
{  语句序列;
 }
```

当条件表达式结果为真(非零)时,执行语句序列。

2. if…else 语句

if…else 语句根据条件表达式的值决定执行哪一段代码。当条件为真时,执行某段语句序列;当条件为假时,执行另一段语句序列。其语法格式如下。

```
if (条件表达式)
{ 语句序列 1;                    //当条件表达式为真时执行这个语句
  }
else
{ 语句序列 2;                    //当条件表达式为假时执行这个语句
  }
```

3. if…else if…else 语句

if…else if…else 语句用于进行多重判断,其语法格式如下。

```
if (条件表达式 1)
{ 语句序列 1;                    //当条件表达式 1 为真时执行这个语句
  }
else if (条件表达式 2)
{ 语句序列 2;                    //当条件表达式 2 为真时执行这个语句
}
⋮
else
{ 语句序列 n;                    //当以上条件表达式均为假时执行这个语句
  }
```

示例 3-11 是关于分支结构的一个综合示例,请认真体会。

【示例 3-11】 闰年的判断方法是:年份是 4 的倍数闰、100 的倍数不闰、400 的倍数闰、4000 的倍数不闰。请判断由键盘输入的年份是否是闰年。

```
#include < iostream.h >
void main( )
{int year;
     cout <<"请输入年份:";
     cin >> year;
     if(year % 4!= 0)
         cout << year <<"不是闰年!"<< endl;
     else if(year % 100!= 0)
           cout << year <<"是闰年!"<< endl;
         else if(year % 400!= 0)
             cout << year <<"不是闰年!"<< endl;
             else if(year % 4000 == 0)
                 cout << year <<"不是闰年!"<< endl;
                 else cout << year <<"是闰年!"<< endl;
     return;
}
```

4. switch 语句

switch 语句用于定义某个变量在几种不同值时做的不同动作,语法格式如下。

```
switch (含变量的表达式)
{
 case 常量 1: 语句序列 1; break;    //当表达式的值与常量 1 相符时执行语句序列 1
 case 常量 2: 语句序列 2; break;    //当表达式的值与常量 2 相符时执行语句序列 2
⋮
default: 语句序列 n + 1;            //当表达式的值与以上常量值都不相符时执行语句序列 n + 1
}
```

在这个结构中,default 子句也可以缺省。

3.4.3 循环语句

循环语句允许程序重复执行某一组语句,在 C++中提供了 3 种形式的循环结构,即 while、do 和 for 语句,这 3 种结构的最大不同在于怎样控制循环。

1. while 语句

while 语句控制了典型的当型循环，只要条件表达式为真，就重复执行循环体语句，当条件为假时，就结束循环。其格式如下。

```
while(条件表达式)
{循环体语句序列;                    //当条件为真时执行
}
```

【示例 3-12】 用 while 语句计算并输出从 1 加到 100 的和。

```
#include <iostream.h>
void main( )
{
    int sum = 0;
    int i;
    i = 1;
    while(i <= 100){
        sum += i;
        i++;
    }
    cout <<"1 + … + 100 = "<< sum << endl;
}
```

2. do…while 语句

do…while 语句“先执行，后判断”的结构，不同于 while 语句，它是非零次循环结构，即至少执行一次循环体。执行过程是先执行循环体语句，然后判断条件表达式，若条件表达式为真，则继续执行循环体；若条件表达式为假，就终止循环，执行其他语句。其语法格式如下。

```
do
{ 循环体语句序列;
}while(条件表达式);              //当条件为真就再一次执行循环体
```

【示例 3-13】 用 do…while 语句计算并输出从 1 加到 100 的和。

```
#include <iostream.h>
void main( )
{
    int sum = 0;
    int i;
    i = 1;
    do{
        sum += i;
        i++;
    } while(i <= 100);
    cout <<"1 + … + 100 = "<< sum << endl;
}
```

3. for 语句

for 语句通常用于处理已知循环次数的情况，即适用于计数型循环，当然在 C++中 for 语句也可以完成各种非计数型循环。for 语句的语法格式如下。

```
for (表达式 1; 表达式 2; 表达式 3)
{
  循环体语句序列;
}
```

其中，表达式 1 常用于初始化循环变量等，应在循环前完成的简单工作可以用这个表达式表示；表达式 2 是循环条件表达式，当条件为真(非零)时，将执行循环体语句，当条件为假(等于零)时，结束循环；表达式 3 在每次执行循环体后执行，它一般用为循环变量的增量表达式。

【示例 3-14】 用 for 语句计算并输出从 1 加到 100 的和。

```
#include <iostream.h>
void main( )
{
    int sum = 0;
    int i;
    for(i = 1;i <= 100;i++)
        sum += i;
    cout <<"1 + … + 100 = "<< sum << endl;
}
```

4. 多重循环

与所有语言一样，C++中的循环语句也可以嵌套使用，即多重循环，3 种格式的循环语句可以根据需要相互嵌套。

【示例 3-15】 随机生成 10 个 100 以内正整数，存放在数组中，用冒泡法将 10 个数按从小到大排序，输出排序后结果。

```
#include <stdlib.h>
#include <iostream.h>

void main(){
    int i,j,t,a[10],flag;
    cout <<"10 integers:\n";
    for(i = 0;i < 10;i++){
        a[i] = rand() % 100;          //生成随机数
        cout << a[i]<<"    ";
    }
    cout << endl;
    for(i = 0;i < 9;i++) {            //外层循环
        flag = 0;                     //设交换标志为 0
        for(j = 0;j < 10 - i - 1;j++) //内层循环
            if(a[j]> a[j + 1]) {
                t = a[j];             //交换
```

```
                a[j] = a[j + 1];
                a[j + 1] = t;
                flag = 1;              //有交换发生,设标志为 1
               }
        if(flag == 0)break;            //若无交换发生,则表示排序结束
    }
    cout <<"The sequence after sort is:\n";
    for(i = 0;i < 10;i++)
        cout << a[i]<<"    ";
    cout << endl;
    return;
}
```

运行结果如下。

```
10 integers:
41    67    34    0    69    24    78    58    62    64
The sequence after sort is:
0    24    34    41    58    62    64    67    69    78
```

3.4.4 转移语句

转移语句使函数内的程序无条件地改变控制权,即在程序段间进行控制转移。C++中的转移语句包括 break、continue 和 goto 语句,这些语句由于是无条件转移,所以常常与 if 等条件语句配合使用。

1. break 语句

break 语句可以用在 switch 结构、while 结构、do…while 结构和 for 结构四种结构中,用于强行退出结构,而去执行该结构后面的语句。

2. continue 语句

continue 语句只能用在三大循环结构当中,用于终止当前这一次循环,重新判断循环条件,至于是否进行下一次循环,要看循环条件而定。

3. goto 语句

goto 语句是无条件转向语句,可以用在程序中的任何位置,但在结构化程序设计当中,goto 语句只能从结构里向结构外跳转,反之则不行。由于大量使用 goto 语句会大大降低程序的可读性,因此在程序设计中提倡少用 goto 语句。goto 语句的格式如下。

```
goto  语句标号;
```

其中语句标号是标识符,是写在某语句前面,用来标识这条语句的标记。

【示例 3-16】 计算并输出用户从键盘输入的正整数的平均值,输入结束标志是 0,使用永真循环语句完成。

```
#include < iostream.h >
```

```
void main( )
{
    int sum = 0;
    int val;
    int num = 0;
    cout <<"输入正整数,输入为 0 时结束"<< endl;
    do{
        cin >> val;
        if(val == 0)break;
        if(val < 0) continue;
        sum += val;
        num++;
    }while(1);                          //永真循环
    cout <<"平均值 = "<<(float)sum/num << endl;
}
```

下面是两个控制结构的比较综合的例子,请读者参考阅读。

【示例 3-17】 利用牛顿法求一个正数的平方根。

```
#include < iostream.h >
#include < math.h >
void main( ){
    const double e = 1e - 5;            //给定的精度要求
    double num;                         //将由键盘输入的求根的正数
    double root, pre;                   //root 是所求的根,pre 是这次迭代前的 root 的值
    cout <<"Enter a number: ";
    cin >> num;
    if(num < 0)
        cout <<"Input error!"<< endl;
    else
        { root = 1;
          do
          { pre = root;
            root = (num/root + root)/2;
            }while(fabs(pre - root)> e);
          cout <<"The root of "<< num <<" is "<< root << endl;
        }
}
```

【示例 3-18】 模拟电影院售票。电影院的座位有 20 行,每行 24 个座位。程序开始时所有票都没有卖,用户输入要买的票的数目,如果票数还够,就卖给用户,如果不够,就显示提示信息。如果输入 0,则表示购票过程结束。每卖出一次票,显示一下票的出售情况。

```
#include < iostream.h >

const int rows = 20;
const int cols = 24;

void main( ){
    bool seats[rows][cols];
    int i,j;
```

```
    int rem = rows * cols;                  //剩余票数

    //初始化
    for(i = 0;i < rows;i++)
        for(j = 0;j < cols;j++)
            seats[i][j] = false;

    //开始卖票
    int num;                                //用户要买的票数
    while(1){
        cout <<"输入想买的票数,输入 0 表示结束: "<< endl;
        cin >> num;
        if(num == 0)break;
        if(num < 0)continue;                //输入错误,重新输入
        if(num > rem){
            cout <<"剩票不足"<< endl;
            continue;
        }                                   //剩票不足

        rem -= num;
        for(i = 0;i < rows;i++){
            for(j = 0;j < cols;j++)
                if(!seats[i][j]){
                    seats[i][j] = true;
                    num -- ;
                    if(num == 0)break;
                }
                if(num == 0)break;
        }

        for(i = 0;i < rows;i++){
          for(j = 0;j < cols;j++)
            if(seats[i][j])
                cout <<'X';
            else
                cout <<'O';
            cout << endl;
        }
    }
    cout << endl;
}
```

3.5 函　　数

一个大型的程序一般可以分成一系列“单一功能模块”。在 C++中,这种单一的功能模块就是函数。一个完整的 C++程序包含一个主函数和若干个子函数,主函数可以调用其他的子函数,子函数之间也可以互相调用。通过使用函数(function)可以把程序以更模块化的形式组织起来,从而利用 C++所能提供的所有结构化编程的潜力。编译系统本身带有很

多预定义函数，并把它们以库函数的形式提供给程序员，这也大大地降低了程序设计的工作量。

3.5.1 普通函数定义

一个函数是一个可以从程序其他地方调用执行的语句块。在C++中普通函数的定义格式如下。

```
数据类型  函数名(形式参数表)
{
    函数体;
}
```

组成C++函数的要素有函数名、参数、返回类型和函数体。其中函数名、参数表和返回类型一起组成函数头，它是函数的接口，即调用这个函数时需要知道这些信息，而函数体是函数真正的实现。

值得注意的是，在C语言中，如果函数的定义在调用之前，则可以直接使用，但如果是定义出现在调用之后，则要先进行函数说明。在C++中也是这样，而且好的程序会更重视函数的说明，这样会提高程序的可读性，因此，一般地，函数说明要放在程序的最前面。在进行函数说明的时候，C++中必须进行完整的说明，而不能像C语言中那样，只说明返回类型与函数名就够了，也就是说，在C++中，函数说明包括函数返回类型、函数名和完整的形式参数表。这是因为，C语言中函数说明只是为了告诉编译器函数的名字，而在C++中由于函数允许同名，因此仅有名字是不够的，还需要知道形式参数的个数、顺序和类型等信息。

函数名是一个标识符，它代表了这个函数，也就是代表了这个函数所封装的一组代码。函数名的命名规则与变量相同，即由字母、数字和下划线组成且由字母和下划线开头的一串字符。在给函数命名时，应该尽量遵守见名知义的原则，使名字能够代表函数所完成的功能，这样能够使程序的可读性更好一些。

函数参数是函数完成功能所需要的输入信息，例如，定义一个求两个数中较大数的函数，那这两个数就要作为参数，又例如，求 $n!$ 的函数，这个 n 就要作为参数传递给函数。一个函数可以有零个或多个任意数据类型的参数，参数之间用“,”隔开，参数名也是标识符。例如，下面是求两个数中较大数的函数声明，其中 a 与 b 就是两个参数。

```
int max(int a, int b)
⋮
```

在函数定义中的参数称为形式参数，因为这些参数的值是从调用函数的地方传递过来的，在定义时还没有具体的值，如上述例子的参数 a 和 b 就是形式参数。与形式参数对应的是实在参数，就是指在调用这个函数时要传递给形式参数的具体量，例如，下面的程序调用了max函数，其中的 $v1$ 与 $v2$ 就是实在参数。

```
void main( )
{
    int v1 = 5,v2 = 6;
    cout << max(v1,v2)<< endl;
}
```

在调用函数时，实在参数将自己的值传递给形式参数。如在上述例子中，在调用函数的语句“cout<<max(v1,v2)<<endl;”中，v1 将自己的值 5 传递给对应的形式参数 a，于是，a 的值就是 5。同样地，v2 将自己的值 6 传递给 b。

函数的返回类型是函数在调用结束后返回结果值的数据类型，可以是除数组以外的任意类型。如果定义的函数返回类型是 void，就表示函数将没有返回值。函数体中的 return 语句是用来返回函数结果的，这个语句指示系统结束当前函数的执行，返回到调用这个函数的地方继续执行。当函数无返回值时，语句格式如下。

```
return;
```

也可以不写这个语句，函数在执行到函数语句末尾的“}”自动结束调用返回。

当函数有返回值时，可以用下面的两种格式之一。

```
return  表达式;
return(表达式);
```

其中，表达式的值就是函数所要返回的值。

函数体是由一些语句组成，这些语句共同完成了函数的功能。函数体中的语句可以是任意形式的语句，包括变量和常量的定义语句、表达式语句和流程控制语句等。变量的定义如果是在函数体内，则这个变量是局部变量，只能在这个函数体中使用；如果变量的定义是在函数体外，则该变量是全局变量，可以在所有函数中使用。例如，下面的程序，请读者通过注释语句理解全局变量与局部变量。

```
int global;                          //全局变量

void main()
{
 int local1;                         //局部变量
 local1 = 5;                         //局部变量只能在本函数中使用
 global = 9;                         //可以在函数中访问全局变量
}

void func()
{
 int local2;                         //局部变量
 local2 = 6;                         //局部变量只能在本函数中使用
 global = 4;                         //可以在函数中访问全局变量
}
```

示例 3-19 是一个函数声明、实现和调用的完整示例。

【示例 3-19】 定义一个函数 max，用于求两个整数的最大值，并在主函数中调用这个函数。

```
//分析：函数功能是求两个数中的较大者，因此，这两个数应该作为参数，求出的较大的数作为返回值。
#include <iostream.h>

int max(int x, int y);                          //函数声明
```

```
void main()                                          //主函数定义
{
    int v1,v2;
    cout<<"输入两个整数:";
    cin>>v1>>v2;
    cout<<"较大数为:"<<max(v1,v2)<<endl;            //调用函数
}

int max(int x, int y)                                //函数实现
{
    if(x>=y)
        return x;
    else
        return y;
}
```

执行结果如下。

```
输入两个整数:4 5
较大数为:5
```

主函数中定义了两个整型变量,并从键盘读入了这两个变量的值,然后调用函数 max 求这两个变量中较大的值并输出。可以看到,函数调用本身也是一个表达式,它可以用在任何表达式能用的地方。

3.5.2 普通函数的调用机制

1. 普通传值型形式参数

调用一个函数就是暂时中断现有程序的运行,转去执行被调用函数,当被调用函数执行结束后,再返回到中断处继续执行的过程。例如,示例 3-19 中,系统先调用 main( )函数,main( )函数按顺序执行各语句,当需要计算 max($v1$,$v2$)时,就产生了一个函数调用,程序的执行转到 max( )中,并将实参 $v1$ 与 $v2$ 的值传递给形参 x 和 y。当执行结束时,用 return 语句返回最大值,回到调用函数的地方,继续执行后序语句。当然如果一个函数没有返回值,函数调用一般会是一个单独的语句。

函数的调用过程如图 3-3 所示,具体说明如下。

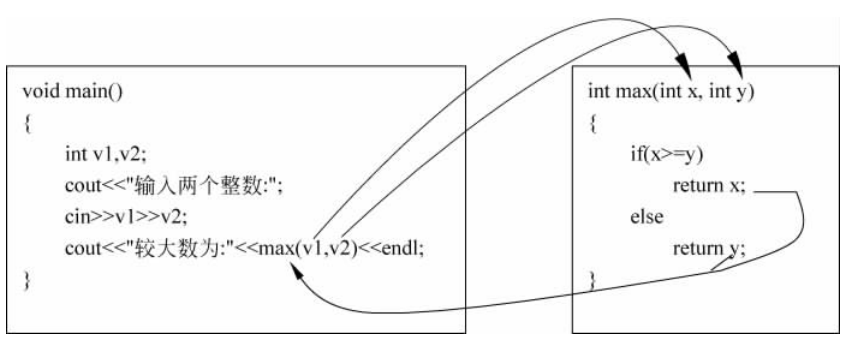

图 3-3 函数调用过程示意图

(1) 当函数调用开始时,建立调用函数的栈空间,保存调用函数的运行状态和返回地址,先将函数进栈,并为函数的形式参数按其数据类型分配动态内存。

(2) 将实参的值对应传递给形式参数。

(3) 执行函数体。

(4) 当执行到 return 语句或函数结束的"}"时,为返回值按函数返回类型分配临时单元,并将返回值放入该单元,函数出栈,清函数所占内存,返回值的临时单元参与主调函数中的所在表达式运算后销毁,继续主调函数的执行。

2. 指针型形式参数

指针本身是一个变量,它的特殊之处在于它的值是一个地址,因而可以通过指针来间接访问另外一个内存地址,这个地址一般是另一个变量存放变量值的地方。当函数的参数是指针时,它的实在参数的值必须是一个地址。这种指针型函数参数提供了一种可以间接修改调用该函数的参数值的手段,但这种做法在 C++中因其容易出错已经很少使用,读者可以通过示例 3-20 对指针型参数的作用有所了解。

【示例 3-20】 定义两个函数,一个是普通的值型形参,一个是指针型形参,在主函数中调用这两个函数。

```
#include "iostream.h"

void fun1(int a)                                    //传值调用
    { cout <<"     形参 a 为: "<< a << endl;
      a -- ;
      cout <<"     形参 a 变为: "<< a << endl;
    }

void fun2(int * b)                                  //指针调用
    {   cout <<"     形参 b 为: "<< b << endl;
        cout <<"     指针形参 b 所指内容为: "<< * b << endl;
        ( * b) -- ;
        cout <<"     指针形参 b 所指内容变为: "<< * b << endl;
    }

void main()
    { int n = 11;
      cout <<"实参 n 为: "<< n << endl;
      fun1(n);                                      //传值调用
      cout <<"数值 n 为: "<< n << endl;

      cout << endl;
      cout <<"数值 n 为: "<< n << endl;
      cout <<"实参 n 的地址为: "<< &n << endl;
      fun2(&n);                                     //指针调用
      cout <<"数值 n 变为: "<< n << endl;
     }
```

函数的运行结果如下。

```
实参 n 为: 11
    形参 a 为: 11
    形参 a 变为: 10
数值 n 为: 11

数值 n 为: 11
实参 n 的地址为: 0x0012FF7C
    形参 b 为: 0x0012FF7C
    指针形参 b 所指内容为: 11
    指针形参 b 所指内容变为: 10
数值 n 变为: 10
```

3. 数组型形式参数

数组型参数是指形式参数的一种特殊情况，C++使用按地址调用的参数传递机制来传递数组，并且C++把所有的数组参数自动转为指针参数。当形式参数被定义为数组时，调用函数时实际上是将实在参数（也是一个数组）的首地址传递给了形式参数（一个指针变量）。

【示例 3-21】 写一个求数组中最小元素的函数，由主函数来调用。

```
#include < iostream.h >

int FindMin(int array[ ],int size);
void main()
{
    int array[10] = {5,6,11,45,34,6,7,12,0,8};
    cout <<"最小值是:"<< FindMin(array,10)<< endl;
}

int FindMin(int array[ ],int size)
{
    int min = array[0];
    for(int i = 1;i < size;i++)
        if(array[i]< min)
            min = array[i];
    return min;
}
```

在这个示例中，数组参数实际上是一个指针，这个指针实际指向实在参数的数组元素。C++规定数组按地址传递是为了节省空间。因为数组一般有比较多的元素，会占用较大的内存空间，如果采用按值传递数组的话，将会需要为存储形参数组元素消耗大量空间。因为数组参数实际上是指针，所以可以通过指针来间接修改实在参数数组元素的值。也就是说，在函数中对数组参数所做的改变会影响到实在参数。

3.5.3 函数与引用

前面已提到了“引用”，在函数中使用引用型形式参数，可以实现原来C语言中不能直

接实现的“变参”效果。试比较下面两个函数，其中例 1 可以实现交换两个变量的值，而例 2 不会实现，这是因为例 1 用了引用型形参，达到了变参的效果，而例 2 是普通的值参。

例 1：

```
void swap(int& m, int& n)
{ int temp = m;
 m = n;
 n = temp;
}
```

例 2：

```
void swap(int  m, int  n)
{ int  temp = m;
 m = n;
 n = temp;
}
```

假设程序中有语句“a＝2，b＝3；swap（a，b）；”，若调用例 1 函数，执行过程是实参 a 将自己的地址传给形参 m，因为 m 是 a 的引用，所以与 a 共用同一个内存；同理，实参 b 也将自己的地址传给形参 n。在例 1 函数中交换 m、n 的值就相当于交换了 a、b 的值。所以这个语句结束后，a 的值是 3，而 b 的值是 2。

同样的语句“a＝2，b＝3；swap（a，b）；”，若调用例 2 函数，执行过程是首先为形参 m、n 申请内存，然后用实参 a 和 b 的值为形参 m 和 n 初始化。因为这个过程是传值的，形参 m、n 各自单独建立内存，进入函数后 m、n 的值就与 a、b 没关系了，因此当函数中 m、n 的值交换后，a、b 的值仍然没有变化。

例 3：

```
void swap(int * pm, int * pn)
{ int temp = * pm;
   * pm = * pn;
   * pn = temp;
}
```

例 3 与例 1 的函数的执行效果是等价的，这是原来 C 语言中解决变参问题的做法，是通过指针型形参去实现变参的效果，这种方法不直观、易出错，还常常用到多级指针，很麻烦，而使用引用型形参去实现变参效果则更方便、更适用。

一个函数也可以返回为引用类型。返回引用的函数可使函数出现在等号的左边，但是要求函数必须返回全局变量。下面示例 3-22 是函数使用“引用”的一个例子，程序中的 index（）函数就是一个返回为引用的函数。这时对函数的使用相当于对返回的变量的使用，当然对函数的赋值也相当于对返回变量的赋值。

【示例 3-22】 函数参数使用“引用”示例。

```
#include < iostream.h >
int a[ ] = {2,4,6,8,10,12};                    //全局数组
int& index(int i)
  {return a[i];}                               //返回一个全局变量
```

```
void swap(int& m, int& n)                          //使用引用型形参
  { int temp = m;
    m = n;
    n = temp;
     }                                             //用于交换两个变量的值
void main( )
  {int x = 5,y = 7;
   swap(x,y);
   cout <<"x = "<< x <<"   y = "<< y << endl;       //输出结果是: x = 7   y = 5
   cout << index(3)<< endl;                        //index(3)相当于 a[3],输出结果是: 8
   index(3) = 100;                                 //相当于为 a[3]赋值
   cout << index(3)<< endl;                        //输出结果是: 100
  }
```

3.5.4 函数与 const

const 是不变的意思,在 C++程序中,经常用来限制对一个对象的操作,如前面 3.1.5 节讲到的 const 常量。const 这个关键字也经常出现在函数的定义中,而且会出现在不同的位置,比如:

```
nt strcmp(const char * str1, const char * str2);   //const 修饰函数参数
const int & min(int&, int&);                       //const 修饰函数返回值
void printMessage(char * msg)const;                //const 修饰类中成员函数,说明其为"只读函数"
```

第三种情况将在下一章学习,下面重点介绍前两种情况。

1. const 参数

const 修饰函数参数是它最广泛的一种用途,它表示函数体中不能修改参数的值(包括参数本身的值或者参数其中包含的值)。

- 情况 1:

```
void function(const int Var);                      //传递过来的参数在函数内不可以改变
```

又如,函数 strcmp 中的参数声明成 const 的,为保证 str1、str2 字符串不被修改。

- 情况 2:

```
void function(const char * Var);                   //参数指针所指内容为常量不可变
```

又如

```
int max(const int array[ ], int size);            //保证 array 数组内容不被改变
```

- 情况 3:

```
void function(char * const Var);                   //参数指针本身为常量不可变
```

- 情况 4:

```
void function(const Class& Var);                   //引用参数在函数内不可以改变
```

这里,参数为引用,是为了增加效率,避免参数传值,用 const 限制,是防止修改。

2. const 返回值

函数返回值为 const 的情形只用在函数返回为引用的时候。例如前面的 min()函数的例子，当把返回为引用的函数再用 const 限定后，是表示这个函数不能作为左值使用，即不能被赋值。如果函数定义形式为：

```
int & min(int&, int&);
```

这时，可以对函数调用进行赋值，如：

```
min(a,b) = 10;
```

但是，如果函数定义形式为：

```
const int & min(int&, int&);
```

这时，就不能对 min(a,b)调用赋值了。

3.5.5 内联函数

有一些代码段非常常用，而且代码又非常短，这时如果把这段代码编写成函数，则会大大增加程序的不必要开销，但不编写成函数，总是重复写类似的代码，又给程序员带来麻烦。这种情况在 C 语言中一般用有参的宏来解决，例如，求某数的绝对值，其编译预处理语句为“#define Myabs(x) ((x)<0? (-(x)):(x))”，这样做虽然解决了上述问题，但同时也带来了宏的负效应，而且宏语句只是简单的机械替代，不会进行类型检查等工作，这也可能带来一些隐性的错误。在 C++中用内联函数来解决这个问题，内联(inline)是内联扩展(inline expansion)的简称。内联函数在实现效果上与函数相同，可避免有参宏语句的副效应；在实现过程上与有参宏语句相似，在编译时用函数体替代函数调用，节省执行时间。将一个函数定义为内联函数，只需在函数定义的最前面加上关键字 inline 即可。格式如下。

```
inline   数据类型   函数名(形参表)
{
    //函数体
}
```

例如，求浮点数的绝对值的内联函数可定义如下。

```
inline  float   Myabs (float x)
    {return   x>0? -x: x;}
```

C++编译器在处理内联函数时比较特殊，在遇到调用内联函数的地方会用函数体中的代码来替换函数的调用，比如：

```
int a = Myabs(x);
```

等价于：

```
int a =  x>0? -x: x;
```

也就是说，程序执行时并没有真正调用函数 Myabs，而是将内联函数的函数体中的语

句直接在函数调用的地方展开了。

内联函数的使用是有限制的，并不是所有函数都能定义成内联的。C++对内联函数的限制如下。

(1) 在内联函数中不能定义任何静态变量。

(2) 内联函数中不能有复杂的流程控制语句，如循环语句、switch 语句和 goto 语句等。

(3) 内联函数不能递归。

(4) 内联函数中不能说明数组。

如果定义的函数比较复杂，违反了上述要求，即使用 inline 限定，系统也将自动忽略 inline 关键字，而当作普通函数来处理。

3.5.6 带有缺省参数的函数

C++对 C 语言的改进，一方面是在设法减少编码的复杂程度，另一方面是使得编译器能检查出来更多的错误。基于这种目标，在 C++的函数中引进了一种新类型的参数，这就是缺省参数。缺省参数就是在调用这个函数时，不要求程序员指定这个实参，而由编译器在需要时给该参数赋予预先指定的值，当然，当程序员需要传递一个与预先指定值不同的值时，就必须显式地指明这个实参。缺省参数是在函数声明中进行说明的。例如：

```
int  sum(int  a, int b=0);
```

这时可以调用只有一个参数的 sum(1)，相当于 1=>a,0=>b，即相当于调用 sum(1,0)。也可以调用两个参数的 sum(1,2)，这时形参 b 就被实参 2 初始化。

在使用缺省参数时，需要注意以下几点。

(1) 缺省值的指定既可以在函数原型中进行，也可以在函数定义中进行，但是，如果一个文件中既有函数原型又有函数定义，则只能在函数原型中指定参数的缺省值。

(2) 缺省参数无论有几个都必须放在参数表的最后，另外在调用时要指定某缺省参数的值，则必须将它前面的所有参数都指定值，这样编译器才能为所有的形式参数赋予合适的初值。例如，下面的函数，定义缺省参数的方法就是错误的。

```
int  sum(int  a=0, int b);
```

又如有定义：

```
int  sum(int  a=0, int b=0);
```

则如下的函数调用参数匹配方式是按照从左到右的顺序进行的。

```
sum( );                                    //等价于 sum(0,0);
sum(2);                                    //等价于 sum(2,0);
sum(2,3);                                  //正常匹配
```

(3) 虽然 C++中的函数原型可以说明多次，但是，函数的同一参数则只能在一次声明中指定缺省值，而不能重复指定。如下面的定义是不合法的。

```
int  sum(int  a, int b=0);
int  sum(int  a, int b=1);
```

3.6 习 题 3

一、选择题

1. 在32位机中,int型字宽为(　　)字节。

A. 2　　B. 4　　C. 6　　D. 8

2. 下列十六进制的整型常数表示中,错误的是(　　)。

A. 0xad　　B. 0X11　　C. 0xA　　D. 4ex

3. 设 $n=10$,$i=4$,则执行赋值运算 n%=i-1 后,n 的值是(　　)。

A. 0　　B. 1　　C. 2　　D. 3

4. 若有 $a=2$,$b=4$,$c=5$,则条件表达式 a>b?a:b>c?a:c 的值为(　　)。

A. 0　　B. 2　　C. 4　　D. 5

5. 若有 $a=1$,$b=2$,则表达式++a==b 的值为(　　)。

A. 0　　B. 1　　C. 2　　D. 3

6. 下列对变量的引用中错误的是(　　)。

A. Int a; int &p=a;　　B. char a; char &p=a;

C. Int a; int &p;p=a;　　D. float a;float &p=a;

7. 在C++语言中,对函数参数默认值的描述正确的是(　　)。

A. 函数参数的默认值只能设定一个

B. 一个函数的参数若有多个,则参数默认值的设定可以不连续

C. 函数参数必须设定默认值

D. 在设定了参数的默认值后,该参数后面定义的所有参数都必须设定默认值

二、简答题

1. 指出下列数据中哪些可作为C++的合法常数。

(a) 588918　　(b) 0129　　(c) 0x88a　　(d) e-5

(e) "A"　　(f) .55　　(g) '\\'　　(h) 0XCD

(i) 8E7　　(j) π

2. 指出下列符号中哪些可作为C++的合法变量名。

(a) ab_c　　(b) β　　(c) 3xyz　　(d) if

(e) cin　　(f) a+b　　(g) NAME　　(h) x!

(i) _temp　　(j) main

3. 根据要求求解下列各表达式的值或问题。

(a) (int)3.5+2.5

(b) (float)(5/2)>2.4

(c) 21/2+(0xa&15)

(d) 设 int i=10,则表达式 i&&(i=0)&&(++i)的值是多少?该表达式运算结束后变量 i 的值为多少?

(e) 设“int a=1,b=2,c=3;”,求表达式 a<b!=c 的值。

(f) 设“int i=1;”,则 i++‖i++‖i++的值是多少? 表达式运算结束后变量 *i* 的值是多少?

(g) 设“int a,b;”,求表达式(a=1,a++,b=1,a&&b++)? a+1:a+4 的值。

(h) 设“int x=5;”,求表达式 x+=x-=x*x 的值。

(i) 设“int x=0,y=2;”,则语句“if(x=0)y++;”执行完毕后变量 *y* 的值是多少?

(j) 设“int a=2,b=3;”,执行表达式 c=b*=a-1 后变量 *c* 的值是多少?

(k) 写出判断字符型变量 *s*1 的值为'0'～'9'的正确的 C++表达式。

(l) 若给定条件表达式(N)? (C++):(c--),写出与 N 功能等价的表达式。

三、读程序写结果

1. 设有以下程序段,请根据要求回答相应问题。

```
int a,b,c,x;
a = 2;b = 3;c = 7;d = 19;
x = d/b % a;
x = d % c + b/a * 5 + 5;
```

2. 设有以下程序段,请根据要求回答相应问题。

```
int a,b,c;
a = b = c = 0;
a = (++b) + (++c);
a = b = c = 0;
a = (b--) + (--c);
```

3. 写出与下列数字表达式相对应的 C++表达式。

(a) a2+2ab+b2　　(b) 4/3πR3　　(c) 5/9(F-32)　　(d) a≤b≤c

4. 设有以下程序段,请写出运行结果。

```
#include<iostream.h>
void func(int x,int y = 10,int z = 20);
void main()
{
    func(11,12,13);
    func(11);
    func(11,12);
}
void func(int x,int y,int z)
{
    cout<<x + y + z<<endl;
}
```

5. 设有以下程序段,请写出运行结果。

```
#include<iostream.h>
viod main()
{   int x,y = 5;
```

```
    If(x = y!= 0)
      cout <<"x = "<< x << endl;
    else
      cout <<"x = "<< x++<< endl;
    cout << endl;
}
```

四、编程题

1. 编写程序，提示用户输入三角形的三条边长，判断该三角形是否为直角三角形，若是，则输出结果以及三角形面积。

2. 编写程序，求解各种数据类型的存储长度并显示出来，在其中找出存储长度最大和最小的两种数据类型并输出。

3. 编写程序，输入一个华氏温度，将其转换为摄氏温度并输出。已知华氏稳定转换为摄氏度的计算公式如下。

$$C=(F-32)*5/9$$

其中，F 为华氏温度，C 为摄氏温度。

4. 编写程序，输入一个十进制表示的正整数，将其转化为二进制表示并输出结果。