

计算机专业

C++ 程序设计

目录

第1章 认识 C++ 的对象

1.1 初识 C++ 的函数和对象

1. 混合型语言
[单选、填空] C++ 程序以 .cpp 作为文件扩展名, 有且只有一个主函数 main()。
2. 注释方式
[单选、填空] /* 多行注释 */ 和 // 单行注释
3. 输入和输出
iostream.h 是 I/O 流的标准头文件。

C++ 预定义了 4 个流: cin、cout、cerr 和 clog。这 4 个流与 4 个具体的输入输出设备相联接, 具体的联接设备如下:

- (1) cin 与标准输入设备相联接。
- (2) cout 与标准输出设备相联接。
- (3) cerr 与标准错误输出设备相联接(非缓冲)。
- (4) clog 与标准错误输出设备相联接(缓冲)。

在默认情况下, 操作系统指定标准输出设备是显示终端, 标准输入设备是键盘。而标准错误输出设备则总是显示终端。当然, 可以将标准输入输出设备重定向为其他设备, 比如文件等。

4. 使用命名空间

using namespace std; // 使用命名空间

5. 对象的定义和初始化
int z(0); 等价于 int z = 0;
6. 函数原型及其返回值
函数库函数和自定义函数。使用变量规则: 必须先声明, 后使用, 函数也是。
7. const 修饰符
使用 const 修饰的标识符定义是一类特殊的常量, 称为符号常量, 或 const 变量。

1.2 认识 C++ 语言面向过程编程的特点

1. 函数重载
同一函数定义几个版本, 从而使一个函数名具有多种功能。
2. 数据类型
(1) void, C++ 增加的数据类型, 无类型标识符, 只能声明函数的返回值类型, 不能声明变量。
(2) 整数常量有 4 种类型: 十进制常量、长整型常量八进制常量和十六进制常量。
3. 动态分配内存
new 类型名 [size]
指针名 = new 结构名; // 分配
delete 指针名; // 释放
4. 引用
引用实际就是变量的别名。
数据类型 & 别名 = 对象名;

第2章 从结构到类的演变/2

第3章 函数和函数模板/4

第4章 类和对象/6

第5章 特殊函数和成员/11

第6章 继承和派生/13

第7章 类模板与向量/15

第8章 多态性和虚函数/16

第9章 运算符重载及流类库/18

C++ 程序设计基础知识/21

如：int x = 9; int &a = x;

5. 对指针使用 const 限制符
指向常量的指针：const int * p;

常量指针：int * const p = &x;

指向常量的常量指针：const int * const p = &x;

6. 数据的简单输入与输出
[单选,填空] 两种格式控制方式：iso_base 类提供的接口；操控符的特殊函数。

表 1-1 操控符及其含义

名 称	含 义	作 用
dec	设置转换基数为十进制	输入/输出
oct	设置转换基数为八进制	输入/输出
hex	设置转换基数为十六进制	输入/输出
endl	输出一个换行符并刷新流	输出
resetsflags(long flag)	清除 flag 指定的标志位	输出
setsflags(long flag)	设置 flag 指定的标志位	输出
setfill(char ch)	设置 ch 为填充字符	输出
precision(int n)	设置浮点数输出精度 n	输出
setw(int width)	设置输出数据字段宽度 width	输出

表 1-2 常量及其含义

常量名	含 义
ios_base::left	输出数据按输出域左端对齐输出
ios_base::right	输出数据按输出域右端对齐输出
ios_base::showpoint	浮点输出时必须带有一个小数点
ios_base::showpos	在正数前添加一个“+”号
ios_base::scientific	使用科学计数法表示浮点数
ios_base::fixed	使用定点形式表示浮点数

1.3 程序的编辑、编译和运行的基本概念

源文件中含有包含头文件的预编译语句，经过预编译后，产生翻译单元，以临时文件的形式存放于计算机之中。之后进行编译，进行语法检查，产生目标文件（*.obj）。此工程中的所有目标文件经过连接，产生可执行文件。连接包含 C++ 的库函数和标准类库的连接。

第 2 章 从结构到类的演变

2.1 结构的演化

1. 结构发生质的演变
 - (1) 函数与数据共存

```
struct 结构名
  | 数据成员
  | 成员函数
  | ;
```

访问成员方法：结构对象.成员函数
默认访问权限是 public 类型。

(2) 封装性

将数据成员使用 private 关键字定义，则产生封装性。

2. 使用构造函数初始化的对象

构造函数名对对象名(初始化参数)；

2.2 从结构演变一个简单的类

关键字 class 代替 struct，就是一个标准的类。

2.3 面向过程与面向对象

面向过程不必了解计算机内部逻辑，精力集中在算法逻辑和过程。面向对象设计，进行功能和数据抽象，对象是功能和数据抽象的统一。

2.4 C++ 面向对象程序设计的特点

1. 对象

[填空] C++ 使用对象名、属性和操作三要素来描述对象。

2. 抽象和类

[填空] 类是具有相同的属性和操作的一组对象集合，其内部包括属性和操作两个主要部分。类的作用是定义对象。
类给出了属于该类的全部对象的抽象定义，而对象则是符合这种定义的实体。所以将对象称作类的一个实例。

3. 封装

将类封装起来，也是为了保护类的安全。所谓安全，就是限制使用类的属性和操作。在类中，封装是通过存取实现的，对象的外部只能访问对象的公有部分，不能直接访问对象的私有部分。

4. 继承

继承是一个类可以获得另一个类的特性的机制，继承支持层次概念。

5. 多态性

[填空] 不同的对象可以调用相同名称的函数，但可能导致完全不同的行为的现象称为多态性。

2.5 使用类和对象

1. [填空] 使用 string 对象，必须包含这个类的头文件 string.h，即 #include <string.h> 或 #include <string> using namespace std;

对象使用自己成员函数的方法是：
对象名.成员函数

常用的函数如下：

size():计算并输出字符串长度。
substr():用来返回字符串的子串。

find():用来在主串中检索所需字符串，找到返回所在位置，找不到返回 -1。
2. [填空] 使用 complex 类定义复数对象，需包含头文件：

#include <complex>
complex <数据类型> 对象名(实部值, 虚部值);
complex 的 real 和 image 成员函数用来输出对象的实部和虚部的值。

2.6 string 对象数组与泛型算法
string 类有一对用来指示其元素位置的基本成员函数：指示第一个元素的 begin 和指示最后一个元素之后的字符串结束标记 end。

第3章 函数和函数模板

3.1 函数的参数及其传递方式

【改错、程序】函数参数有两种传递方式：传值和传引用。传引用其实就是传对象的地址，所以也称传地址方式。

1. 对象作为函数参数

将对象作为函数参数，是将实参对象的值传递给形参对象，这种传递是单向的。形参是对象的地址值。虽然参数传递方式仍然是传值方式，但因为形参传递的就是实参拥有的备份，当在函数中改变形参时，改变的是这个备份中的值，不会影响原来实参的值。

```
#include <iostream.h>
#include <conio.h>
void swap( int * a,int * b); //函数原型声明
void main()
{
    int x=3;
    int y=4;
    cout << "Before swap" << " x = "
    << x << " y = " << y << endl;
    swap( &x,&y );
    cout << "After swap" << " x = "
    << x << " y = " << y << endl;
    getch();
}
void swap( int * a,int * b)
{
    int temp;
    temp = * a;
    * a = * b;
    * b = temp;
}
```

运行结果：

```
Before swap x=3 y=4
After swap x=4 y=3
```

3. 引用作为函数参数

形参是对象的引用，实参是对象，实参和形参对对象代表一个对象，所以改变形参对象的值就是改变实参的值。

4. 默认参数

默认参数是在函数原型中说明的，默认参数可以多于1个，但必须放在参数序列的后部。如：Display(string s1, string s2 = "", string s3 = " ")；

5. 使用 const 保护数据

用 const 修饰传递参数，它只能使用参数而无权修改。主要是为了提高系统的自身安全。

3.2 深入讨论函数返回值

【程序】返回值类型可以是除数组和函数以外的任何类型。当函数返回值是指针或引用对对象时，需要注意的是，函数返回所指的对象必须继续存在，因此不能将函数内部的局部对对象作函数的返回值。

1. 返回引用的函数：数据类型 & 函数名(参数列表)；

例如声明：int & index(int i)；调用：index(3)；

2. 返回指针的函数：类型标记符 * 函数名(参数列表)；

例如声明：float * input(int &)；

调用：float * a = input(num);

3. 返回对象的函数

例如声明：string input(const int)；

调用：string str = input(n);

4. 函数返回值作为函数的参数

例如声明：int max(int,int);

调用：cout << max(55 ,max(25 ,39)) << endl;

3.3 内联函数

【单选、填空】定义内联函数使用关键字inline。在C++中，除具有循环语句、switch语句的函数不能说明为内联函数外，其他函数都可以说明为内联函数。

使用内联函数能加快程序执行速度，但如果函数体语句多，则会增加程序代码的大小。由于编译器必须知道内联函数的函数体，才能进行内联替换，因此，内联函数必须在程序中第一次调用此函数的语句出现之前定义。

3.4 函数重载和默认参数

【程序】不同对象调用函数名具有多种功能，称这种特征为多态性。分为静态多态性和动态多态性。

函数重载是静态多态性。

double max(double m1,double m2) { ... }

int max(int m1,int m2) { ... }

char max(char m1,char m2) { ... }

带默认值参数：

```
int add( int m1 = 0,int m2 = 0,int m3 = 0,int m4 = 0 ) { ... }
```

3.5 函数模板

【程序】将函数模板与某个具体数据类型连用，就产生了模板函数。

函数模板定义：

函数模板名 < 模板参数 > (参数列表)

默认形式：函数模板名(参数列表)

C++ 还专门定义一个仅仅用在模板中的关键字 typename，它的用途之一是代替 template 参数列表中的关键字 class。

例如：

```
#include <iostream>
using namespace std;
template <typename T>
T max(T m1,T m2)
{
    if(m1 > m2) return m1:m2;
    else return m2;
}
```

```

template <typename T>
T min(T m1, T m2)
{
    return (m1 < m2) ? m1 : m2;
}

void main()
{
    cout << max("abc", "abC") << endl;
    cout << min(2, 3, 5, 8) << endl;
    cout << min<int>(8, 3) << endl;
}

```

运行结果：

```

abC
2.3
3

```

第 4 章 类和对象

4.1 类及其实例化

[单选、填空、程序] 对象就是一类物体的实例, 将一组对象的共同特征抽象出来, 从而形成“类”的概念。

1. 定义类

类中声明的任何成员不能使用 `extern`、`auto` 和 `register` 关键字进行修饰, 类中有数据成员和成员函数, 而且不能在类声明中对数据成员使用表达式进行初始化。

(1) 声明类

```

class 类名
{
private:
    私有数据和函数
public:
    公有数据和函数
protected:
    保护数据和函数
};

```

如果没有使用关键字, 则所有成员默认声明为 `private` 权限。 `public`、`private` 和 `protected` 的使用顺序和次数也都是任意的。

(2) 定义成员函数

类中声明的成员函数用来对数据成员进行操作, 还必须在程序中实现这些成员函数。返回类型类名::成员函数名(参数列表)

```

|成员函数的函数体|
“::”是作用域运算符, “类名::成员函数名”的意思就是对属于“类名”的成员函数进
行定义, 而“返回类型”则是这个成员函数返回值的类型。

```

可以使用 `inline` 将成员函数定义为内联函数。

如果在声明类的同时, 在类体内给出成员函数的定义, 则默认认为内联函数。

(3) 数据成员的赋值
不能在类体内部数据成员赋值, 数据成员的具体值是用来描述对象的属性的。只有

产生了一个具体的对象, 这些数据值才有意义。如果在产生对象时就使对象的数据成员具有指定值, 则称为对象的初始化。

2. 类的对象

对象和引用都使用运算符“.”访问对象的成员, 指针则使用“->”运算符。
成员使用规律：

(1) 类的成员函数可以直接使用自己类的私有、保护和公有成员(数据成员和成员函数)。

(2) 在类外不能直接使用私有、保护成员, 但可以使用公有成员来间接访问私有、保护成员。

(3) 可以定义类的多个对象, 不同的对象可以有不同的属性值。

定义类对象指针的方法：

类名 * 对象指针名；

对象指针名 = 对象的地址；

`Point a; Point * p2; p2 = &a;`

也可以直接进行初始化：

类名 * 对象指针名 = 对象的地址；

如：`Point a; Point * p2 = &a;`

类对象的指针可以使用“->”运算符访问对象的成员：

对象指针名 -> 对象成员名

如：`p2 ->x; p2 -> fun();`

3. 数据封装

C++ 对其对象的数据成员和成员函数的访问是通过访问控制权限来限制的。 C++ 通过类实现数据封装, 即通过指定类成员的访问权限来实现。

4.2 构造函数

[程序] 构造函数对象用来设置初值的函数。

构造函数的特征：

1. 构造函数名称与类名称相同且可以由用户自行定义。

2. 构造函数不能有返回值, 也不能声明返回类型。

错误的构造函数声明：

```

void Grade(char * stu_name , int stu_grade);
char * Grade(char * stu_name , int stu_grade);

```

3. 构造函数可以有多个, 即构造函数可以重载。

4. 构造函数在建立对象时自动被调用。

5. 没有参数的构造函数称为主构造函数(`default constructor`)。没有定义构造函数时, 自动产生默认构造函数。

6. 构造函数也受数据封装限制而有数据隐藏的特性。

构造函数设置成员初值方法有两种：一种是在函数体内赋值, 另一种是采用初始化列表的形式。

类名::类名(形参1, 形参2, …形参n)

{数据成员1 = 形参1;

数据成员2 = 形参2;

…

数据成员n = 形参n;

```

}类名(形参1, 形参2, …形参n):数据成员1(形参1), 数据成员2(形参2),

```

...，数据成员 n(形参 n)

...

这两种形式等价，但有些不能在函数体内赋值，如对于常数类型（const）与引用类型（reference），必须在定义时给定数值。但类中不可以做初值设置，这时就必须使用到成员初始化列表形式。

7. 构造函数和运算符 new：new 用于建立生存期可控的对象，new 返回这个对象的指针。
8. 带默认值的构造函数：即在声明中指定形参的默认值。
9. 复制构造函数：复制构造函数用已有的对象来建立新对象，必须使用对象的引用作为形参。

复制构造函数将真正产生两个各自占有独立空间且有相同内容的对象。复制构造函数的定义：

```
X::X(X&)
{
    ...
}
```

为了防止修改原有对象，形参使用常引用即 X::X(const X &)。

4.3 析构函数

【填空、程序】对对象删除时被调用的函数。

1. 析构函数与类的名称一样，但在名称的前面加上波浪号（~）。

析构函数特点：

- (1) 析构函数不能有返回值，也不能声明返回类型。
- (2) 析构函数不能有任何参数。
- (3) 允许有一个唯一的析构函数，即不能重载。

析构函数作用：释放新定义对象时用户通过构造函数向系统申请的存储空间。对象离开其有效范围时，便会自动调用析构函数。

全局对象和静态对象的析构函数在程序运行结束之前调用。类的数据成员的每个元素调用一次析构函数。全局对象数组的析构函数在程序运行结束之前被调用。

2. 析构函数和运算符 delete
运算符 delete 与析构函数一起工作。当使用 delete 删一个动态对象时，它首先为这个对象调用析构函数，然后再释放这个动态对象占用的内存，这和使用 new 建立动态对象的过程正好相反。

3. 默认析构函数：在定义类时没有定义析构函数，系统自动产生一个函数体为空的默认析构函数。

4.5 成员函数重载及默认参数

函数可以重载，成员也可以重载，也可以带默认值。

4.6 this 指针

1. [单选、填空] this 指针隐含于每一个类的成员函数中。每一个类的成员函数都有

一个 this 指针变量。当某个对象调用成员函数时，this 指针指向这个对象。

2. [单选、填空] this 指针由编译器自动产生。通过 this 指针可以引用到对象的任何成员（静态成员没有 this 指针）。成员函数也可以显式使用 this 指针。经常省略“->”。

例如：

```
void Point::Sety(int a,int b)
{
    x = a;//等价于 this->x=a;
    y = b;//等价于 this->y=b;
```

4.7 一个类的对象作为另一个类的成员

【填空】类的数据成员，可以是普通的数据类型，也可以是类对象类型。执行构造函数时，先执行成员对象的构造函数，再执行本身的构造函数。

如：

```
#include <iostream.h>
class A
{
public:A() { cout<<"A"; }
};

class B
{
public:B() { cout<<"B"; }
};

A a;
B b;
```

4.8 类和对象的性质

1. 对象的性质

- (1) 同类对象之间可以相互赋值。
- (2) 可以使用对象数组。
- (3) 可以指向对象的指针。
- (4) 对象可以用作函数参数。
- (5) 对象作为函数参数时，可使用对象、对象引用和对象指针。
- (6) 一个对象可以作为另一个类的成员。

2. 类的性质

- (1) 使用类权限，类本身成员可以使用类的所有成员；类对象只能访问公有成员；其他函数不能使用类的私有成员，也不能使用公有成员函数。
- (2) 不完全的类声明，用于在类没有完全定义之前就引用该类的情况。不完全声明的类不能实例化。

```
class A;
class B
{
    A a;
    ...
};
```

在 C++ 中,一个对象的可能消息集是在对象的类描述中说明的,每个消息在类描述中用一个相应的方法给出,即使用成员函数定义操作。

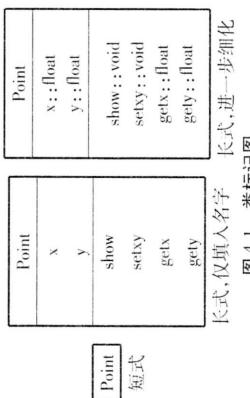
(3) 室类不包括任何声明,如 class A; ;
 (4) 类作用域:声明类时使用的一对花括号形成类作用域。
 使用 struct、class 都可以设计类,但 struct 默认权限是 public 类型,而 class 默认认为 private 类型。

4.9 面向对象的标记图

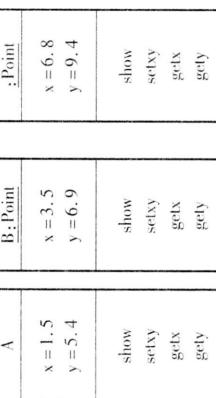
面向对象解决代码编写 OOP 向面向对象分析 OOA 和面向对象设计 OOD 发展。人们采用图像形式将面向对象分析、设计和实现阶段对问题的描述直观地表示出来。1992 年 OMG(面向对象管理)制定的面向对象分析和设计的国际标准 UML 问世。

UML 是一种可视化建模语言,主要用于面向对象分析和建模。

1. 类和对象的 UML 标记图



长式,仅填入名字



使用对象名 使用类和对象名 对象名空缺
图 4-1 类标记图

(3) 条件编译指令:#if #else #endif。

(4) 关键字 defined 不是指令,而是一个预处理操作符。用于判定一个标识符是否已经被#define 定义。已被定义为真,否则为假。

第 5 章 特殊函数和成员

5.1 对象成员的初始化

```

[程序] class A
|_类名1 成员1;
|_类名2 成员2;
...
类名n 成员n;

```

〔填充〕规模较大的类,一般包括一个头文件和一个实现文件。所有编译指令都以#开始,每条指令单独占用一行。

(1) 嵌入指令#include 指标编译器将一个源文件嵌入到带有#include 指令的源文件中该指令所在的位置处。尖括号或双引号中的文件名可含有路径信息。

(2) 宏定义:#define 宏名 替换正文。#define 指令定义一个标识符及串,在源程序中每次遇到该标识符时,编译器均用定义的串代替之。该标识符称为宏名,而将替换过程称之为宏替换。

(3) 条件编译指令:#if #else #endif。

(4) 关键字 defined 不是指令,而是一个预处理操作符。用于判定一个标识符是否已经被#define 定义。已被定义为真,否则为假。

5.2 静态成员

〔单选 填空 改错〕简单成员函数是指声明中不含 const、volatile、static 关键字的函数。如果类的数据成员或成员函数使用关键字 static 进行修饰,这样的成员称为静态数据成员或静态成员函数,统称为静态成员。静态成员只能在类外进行初始化。在类外定义静态成员时,不得用 static。

```

如: class Test
|
| static int x;
|
int Test::x = 25。

```

2. 对象、类和消息
〔填充〕对象的属性是描述对象的数据成员。数据成员可以是系统或程序员定义的数据类型。对象的属性又称对象的状态。

对象的行为是定义在对象属性上的一组操作的组合。操作是响应消息而完成的算法,表示对象内部实现的细节。对象的操作集合体现了对象的行为能力。

对象的属性和行为是对象的组成要素,分别代表了对象的静态和动态特征。

消息是向对象发出的服务请求,它是面向对象系统中实现对象间的通信和请求任务的操作。消息传递是系统构成的基本元素是程序运行的基本处理活动。

一个对象可以同时向多个对象发送消息,也可以接收多个对象发来的消息。消息值反映发送者的请求,由于消息的识别和解释取决于接受者,因而同样的消息在不同对象中可解释成不同的行为。

静态数据成员，可以使用类名形式访问，即类名::静态成员，或者使用对象访问，即对象。

静态成员函数与一般成员的区别是：

- (1) 建立对象前，静态成员已经存在。
- (2) 静态成员是所有对象共享，被存储在公用内存中。
- (3) 静态成员函数不能直接访问非静态成员。
- (4) 静态成员函数不能被说明为虚函数。
- (5) 没有 this 指针。

如#include <iostream.h>

```
class Count
{
public:
    Count() { count++; } // /调用构造函数时, count 加 1
    static int get() { return count; }
};

静态成员类在全局范围内赋初值：
int Count::count = 0;
调用静态成员函数：Count::get();
```

5.3 友元函数

1. [单选·填空·程序] 友元函数可以访问私有、公有和保护成员。友元可以是一个类或函数，尚未定义的类也可以作为友元引用。友元函数虽然在类的定义内声明，但仍不属于该类的成员。友元可以放在公用或私有部分。

使用友元的问题是：它允作访问对象的私有成员，破坏了封装和数据隐藏，导致程序可维护性变差，因此使用时要权衡得失。

友元函数可以在类中声明时定义。如果在类外定义，不能再使用 friend 关键字。

2. 成员函数用做友元一定要指明友元函数所在的类。

如：

```
friend void One::finc(Two &x);
```

友元关系不具有传递性，如类 A 是类 B 的友元，类 B 是类 C 的友元时，类 A 却不是类 C 的友元。

友元关系不具有交换性，如类 A 是类 B 的友元时，类 B 不一定是类 A 的友元。

5.4 const 对象

[填空、改错、程序] 常对象只能访问 const 成员函数，否则出错。

1. 常量成员包括常数据成员、静态常数据成员和常引用。静态常数据成员需要在类外初始化。常数据成员和常引用只能通过初始化列表来获得初值。常引用作为函数参数，传送的是地址。
2. 常对象声明的同时必须进行初始化。
3. 常成员函数类内声明：类型标识符 类名::函数名(参数列表) const;

类外：类型标识符 类名::函数名(参数列表) const;

用内联函数定义：

类型标识符 函数名(参数列表) const | 函数体 |
在常成员函数里，不能更新对象的数据成员，也不能调用该类中没有用 const 修饰的成员函数。

用 const 声明 static 成员函数没有什么作用，在C++ 中声明构造函数和析构函数时使用 const 关键词均是非法的。

5.6 指向类成员函数的指针

定义形式为：类型(类名::* 指针变量)(参数表)；
指向函数：指针变量 = 类名::函数名；
例如：

```
class A
{
public: int value(int a) {}
;

void main()
{
    int (A::* pfun)(int);
    pfun = A::value;
    A obj(10);
    (obj.* pfun)(15);
}
```

第 6 章 继承和派生

6.1 继承和派生的基本概念

[单选·填空] 用户利用继承从旧的数据类型定义出新的数据类型，在新的数据类型中不但拥有新的数据成员与成员函数，也可以同时拥有旧数据类型中的数据成员与成员函数。

被继承的类为基类，而通过继承关系而定义出的类称为派生类。

派生类可以增加或重新定义成员，同时派生类又可以产生新类，这就是类的层次性。基类定义了对对象的集合，派生类通过增添新的成员限制该定义，以便定义这个集合子集。派生类使用两种基本的面向对象技术：第一种为性质约束，即对基类的性质加以限制；第二种称为性质扩展，即增加派生类的性质。

继承分为单继承和多重继承。由类的层次性，单继承便可形成倒挂的树。

6.2 单一继承

```
1. [程序] 一般形式
class 派生类名:访问控制 基类名
private:
    成员声明列表
protected:
    成员声明列表
public:
    成员声明列表
};
```

访问控制访问权限有三种：公用 public、私有 private 和保护 protected。

2. 派生类的构造函数和析构函数

派生类继承了基类的全部数据成员，但除了构造、析构函数之外的全部成员函数。

派生类构造函数定义形式：

派生类名::派生类名(参数表 0)：基类名(参数表 1)；

在设计派生类的构造函数时，若在基类中定义了构造函数，派生类的构造函数必须调用基类的构造函数。即派生类的构造函数除了必须初始化自身的数据成员外，也必须初始化基类中的数据成员。

构造函数的调用顺序：

派生类与基类都定义有构造函数时，则编译器先调用基类的构造函数，如有对对象成员则执行对象成员的构造函数，最后是派生类的构造函数。析构函数与构造函数完全相反的顺序。

3. 赋值兼容规则：指在公有派生情况下，一个派生类对象可以作为基类对象来使用的情况。

有三种情况：派生类对象可以赋给基类的对象，派生类的对象可以初始化基类引用，派生类对象的指针可以赋值给基类指针。

4. 派生类对基类成员的访问

表 6.3 派生类对基类成员的访问能力

基类	公有继承方式	保护继承方式	私有继承方式
public	public	protected	private
protected	protected	protected	private
private	no access	no access	no access

6.3 多重继承

单一继承只从一个基类继承，而多重继承可以有多个基类。其表示方法为：

class C:public A, public B

...

//基类 A 与基类 B 同时为 public 基类

多重继承的派生类负责所有基类中的成员初始化。基类构造函数的调用顺序取决于定义派生类时指明的基类顺序。

6.4 二义性

【程序】多重继承中存在名字冲突解决不确定性的两种选择：

(1) 使用范围运算符指明引用的成员所属的类范围。类名::标识符。

(2) 重新定义基类中冲突的成员，隐藏基类中的同名成员。

派生类支配基类的同名函数。

派生类对象、成员函数

首先查找派生类的成员函数，如果有则执行派生类的成员函数，没有时执行基类的成员函数。如果要调用基类的成员函数，

派生类对象.基类::成员函数；

第 7 章 类模板与向量

7.1 类模板

1. [程序] 类模板的成立成分及语法

(1) 函数模板，即建立一个通用函数，其函数类型和形参类型不具体指定，用一个虚拟的类型来代表。这个通用函数就称为函数模板。将函数模板与具体数据类型连用，就产生了函数，又称这个过程为函数模板实例化，这种形式就是类型参数化。类模板与函数模板相似，将类中一些成员的类型，设计一个通用类型 T，T 是对类描述，称为类模板。类模板与某种特定数据类型联系起来，就产生一个特定的类型为模板类。类模板大简化程序设计。

类模板声明的方法：

```
template <类模板参数> class 类名//类体{
```

(2) 类模板的对象：类模板也称为参数化类。类模板定义对象格式如下：

类名<模板实例化参数类型>对象名(构造函数实参列表)；

类名<模板实例化参数类型>对象名//默认或无参数构造函数

在类体外面定义成员函数时，必须用 template 重写类模板声明。格式：

```
template <模板参数>
```

返回类型 类名<模板类型参数>::成员函数名(函数参数列表) //函数体|

例如：

```
#include <iostream.h>
template <class T>
class A
{
public:
    T max(T a, T b);
};
```

template <class T>

```
T A <T> ::max(T a, T b)
```

return (a > b) ? a : b;

void main()

{ A < int > at;

cout << at. max(34, -12) << endl;

}

运行结果:34

2. 类模板的派生与继承

类模板可以继承，继承的方法与普通的类一样。声明模板继承之前，必须重新声明类模板。模板类的基类和派生类都可以是模板类。

7.2 向量与泛型算法

1. [填空] 定义向量列表

向量(vector)类模板定义在头文件 vector 中，它提供 4 种构造函数，用来定义由各元

素组成的列表。length 表示长度, type 表示数据类型, name 表示对象名。

vector<type> name; //向量空表

vector<type> name (length); //具有 length 个 type 的向量, 元素初始化为 0

vector<type> name (length, a); //具有 length 个 type 的向量, 元素初始化为 a

vector<type> name1 (name); //使用已定义的向量 name 构造向量 name1

不能使用列表初始化向量, 但可以先初始化一个数组, 然后把数组的内容复制给向量。
如: int ia[10] = {1, 3, 5, 7, 9, 12, 14, 16, 18, 20}; vector<int> Vb (ia, ia + 10);
ia 是数组名, 表示数组起始地址, ia + 10 是 Vb 的结束标志, Vb 的长度为 10。因为向量自动产生一个结束标志, 所以 Vb 不需要与 ia 等长, 可以比 ia 长(值不确定)。

2. [填充] 泛型指针

“与操作对象的数据类型相互独立”的算法称为泛型算法。

iterator 在 STL 里面是一种通用指针, 它在向量中的作用相当于 T *。用 iterator 声明向量的正向泛型指针的一般形式:

vector<type>::iterator 泛型指针名;

用 iterator 声明向量的逆向泛型指针的一般形式:

vector<type>::reverse_iterator 泛型指针名;

例如:vector<char>::iterator pr;/p 指针, 注意没有 *

3. [填充] 向量最基本的操作方法

(1) size(): 返回当前向量中已经存放的对象的个数。

max_size(): 返回向量可容纳最多对象的个数。

capacity(): 返回无需再次分配内存就能容纳的对象个数。

empty(): 当前向量为空时, 返回 true 值。

(2) 访问向量中对象的方法

front(): 返回向量中的第 1 个对象。

back(): 返回向量中的最后一个对象。

operator[] (size_type, n): 返回向量中的第 n + 1 个对象(下标为 n 的向量元素)
(3) 在向量中插入对象的方法
push_back (const T&): 向向量尾部插入一个对象。
insert(iterator it, const T&): 向 it 所指的向量位置前插入一个对象。
insert(iterator it, size_type n, const T& X): 向 it 所指向量位置前插入 n 个值为 X 的对象。

(4) 在向量中删除对象的方法

pop_back (const T&): 删除向量中最后一个对象。

erase(iterator it): 删除 it 所指向的容器对象。

clear(): 删除向量中的所有对象, empty() 返回 true 值。

第8章 多态性和虚函数

8.1 多态性

[填充] 静态联编所支持的多态性称为编译时的多态性, 通过重载实现。动态联编所支持的多态性称为运行时的多态性, 由虚函数来支持。

1. 静态联编中的赋值兼容性及名字支配规律
类的对象和调用的函数一一对应, 编译时即可确定调用关系, 从而产生编译时的多态性。

2. 动态联编的多态性

当编译系统编译含有虚函数的类时, 将为它建立一个虚函数表, 表中的每一个元素都指向一个虚函数的地址。虚函数的地址取决于对象的内存地址, 也就是不同对象具有不同的地址。

8.2 虚函数

1. [填充, 程序] 虚函数的定义
虚函数只能是类中的一个成员函数, 但不能是静态成员, 关键字 virtual 用于类中该函数的声明中。

virtual 返回类型 函数名(形参列表);

在派生类中定义了一个同名的成员函数时, 只要该成员函数的参数个数和相应类型以及它的返回类型与基类中同名的虚函数完全一样, 则无论是否为该成员函数使用 virtual, 它都将成为一个虚函数。

2. [填充, 程序] 虚函数实现多态性的条件

产生运行时的多态性有 3 个前提:
(1) 类之间的继承关系满足赋值兼容性规则。
(2) 改写了同名虚函数。
(3) 根据赋值兼容性规则使用指针或引用。

3. [填充, 程序] 构造函数和析构函数调用虚函数

在构造函数和析构函数中调用虚函数采用静态联编, 即它们所调用的虚函数是自己所在的类或基类中定义的函数, 但不是任何在派生类中重定义的虚函数。
C++ 不支持虚构造函数, 但支持虚析构函数。

4. [填充, 程序] 纯虚函数

在许多情况下, 不能在基类中为虚函数给出一个有意义的定义, 这时可以将它说明为纯虚函数, 将其定义留给派生类去做。说明纯虚函数的一般形式如下:
class 类名
{ public:
 virtual 函数类型 函数名(参数列表) = 0;
};

一个类可以说明多个纯虚函数, 包含有纯虚函数的类称为抽象类。一个抽象类只能作为基类来派生新类, 不能说明抽象类的对象。
使用纯虚函数必须注意以下几个问题:
(1) 纯虚函数的声明格式。
(2) 含有纯虚函数的类不可以用来定义对象。
(3) 当派生类中没有重新定义基类中的纯虚函数时, 必须继续声明这些虚函数为纯虚函数, 派生类仍然是抽象类, 如果派生类实现了基类的非纯虚函数, 派生类就可以定义对象。
(4) 含有纯虚函数的类也可以定义其他的非纯虚函数, 虽然程序中不可以定义该类的对象, 但还可以调用这些一般的虚函数。

(3) 当派生类中没有重新定义基类中的纯虚函数时, 必须继续声明这些虚函数为纯虚函数, 派生类仍然是抽象类, 如果派生类实现了基类的非纯虚函数, 派生类就可以定义对象。

(4) 含有纯虚函数的类也可以定义其他的非纯虚函数, 虽然程序中不可以定义该类的对象, 但还可以调用这些一般的虚函数。

多重继承是多个单一继承的组合, 因此多重继承情况下的虚函数调用与分析单继承有相似之处。

8.3 多重继承与虚函数

8.4 类成员函数的指针与多态性

在派生类中，当有一个指向基类成员函数的指针指向一个虚函数，并且通过指向对象的基本类指针或引用访问这个虚函数时，仍发生多态性。

第 9 章 运算符重载及流类库

9.1 运算符重载

1. [填充、操作] 重载对象的赋值运算符

编译器在默认情况下为每个类生成一个默认的赋值操作，用于同类的两个对象之间相互赋值。但对有些类可能是不正确的，如字符串类。

重新定义运算符时，不可以违反运算符原先是 C++ 中的使用规则，例如：纯双目运算符“/”不可以被用户定义为单目运算符。
要将运算符重载必须调用运算符函数来完成，运算符函数的形式如下：

```
operator op(argument_list);
其中 op 是要重载的运算符（运算符必须是 C++ 中所定义的运算符）。
例如：str & operator = (str & a) {
    operator op(argument_list);
    其中 op 是要重载的运算符（运算符必须是 C++ 中所定义的运算符）。
    例如：str & operator = (str & a) {
        s2.operator = (s1);
        s2 = s1; C++ 编译解释为：
        s3.operator = (s2.operator = (s1));
    2. [填充、操作] 运算符重载的实质
    运算符重载有两种，将作为类的成员函数的重载运算符称为类运算符，而将作为类的友元的运算符称为友元运算符。
    不能重载的运算符有：“_”、“_”、“*”、“_”和“?”。“#”不是运算符，不能重载。
    只能用类运算符重载的有：“_”、“_”、“_”、“_”、“_”、“_”。
    3. <<、>> 和 + + 运算符重载
    定义形式：ostream & operator << ( ostream & output, 类名 & 对象名 )
    //.../函数代码
    return output;
    调用形式：operator << ( cout, 对象名 );
    定义形式：istream & operator >> ( istream & input, 类名 & 对象名 )
    //.../函数代码; return input;
    调用形式：operator >> ( cin, 对象名 );
    class Test
    {
    public:
        int i;
        float f;
        char ch;
    }
```

```
    stream << obj.i << " ";
    stream << obj.f << " ";
    stream << obj.ch << endl;
    return stream;
}
istream & operator >> ( istream & instr, Test &obj )
{
    instr >> obj.i;
    instr >> obj.f;
    instr >> obj.ch;
    return instr;
}
```

```
void main()
{
    TestA(45,8,3,'w');
    operator << ( cout, A );
    Test B;
    operator >> ( cin,B );
}

int operator + + ()//前缀 + + n
{
    ...;return 返回值;
}
int operator + + (int )//后缀 n + + , 不用给出形参名
{
    ...;return 返回值;
}
友元运算符：
friend int operator + + ( number & )//前缀 + + n
friend int operator + + ( number & , int )//后缀 n + + , 不用给出 int 类型的形参名
4. 类运算符和友元运算符的区别
若运算符所含的操作数希望进行隐式类型转换，则运算符应通过友元来重载。
如果一个运算符的操作需要修改类对象的状态，则应当使用类运算符，这样更符合
据封装的要求。
```

9.2 流类库

1. 流类库的基础类

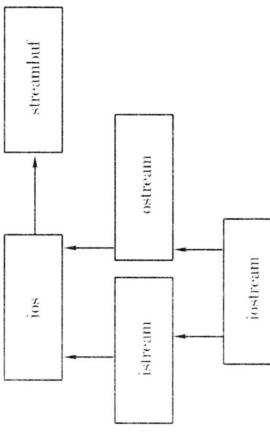


图 9-1 iostream 流类库的基础类关系图

- 默输出格式控制
- 默输入输出格式能够自动识别浮点数并用最短的格式输出。
关于数值数据，默方式能够自动识别浮点数并用最短的格式输出。
字符的读入，对单字符来讲，它将舍去空格，直到读到字符为止；对字符串来讲，它从
读到第一个字符开始，到空格符结束。对于字符串组，使用数组名来整体读入。

Bool 布尔类型、VC 把输入 0 褒别为 false，其他的值均识别为 1，输出只有 1 和 0 两个值。

3. 使用 ios_base 类

(1) ios_base 类派生 ios 类，ios 又是 istream 和 ostream 的虚基类。

常量：skipws, left, dec, oct, hex, showbase, showpos, fixed 等。
处理标志的成员函数：

```
long flags( long ), long flags( ), long set( long, long ), int width( ), char fill( char ), int precision( int )等。  
(2) 成员函数：在使用成员函数进行格式控制的时候，set 用来设置，unset 用来恢复默认设置。它们被流对象调用。  
如：cout << showpoint << 123.3; cout << cout.precision() << " " << 3.14;  
cout.set( ios_base::showpoint ); cout << 123.45 << " " ; cout.unset( ios_base::showpoint );
```

9.3 文件流

C++ 总共有输入文件流、输出文件流和输入输出文件流 3 中，并已经将它们标准化。要打开一个输入文件流，需要定义一个 ifstream 类型的对象；要打开一个输出文件流，需要定义一个 ofstream 类型的对象；如果要打开输入输出文件流，则要定义一个 fstream 类型的对象。这 3 种类型都定义在头文件 <fstream> 里。

1. 使用文件流

对文件进行操作的方法：

(1) 打开一个相应的文件流。ofstream mystream;
(2) 把这个流和相应的文件关联起来。

mystream.open("myText.txt");

可以用一条语句

ofstream myStream("myText.txt");

来完成上述两步。

(3) 操作文件流。使用 << 和 >>，进行文件读写。

(4) 关闭文件流：文件流名.close();

2. 流成员函数

(1) 输出流的 open 函数

void open(char const * , int filemode, int = filebuf::openprot);

形参 1，表示要打开的文件名；形参 2，表示文件的打开方式；形参 3，文件保护方式，一般采用默认值。

[程序] 打开方式：

ios_base::in 打开文件读操作

ios_base::out 打开文件进行写操作，默认模式

ios_base::app 打开文件在文件末尾添加数据

ios_base::trunc 如果文件存在，将其长度截断为 0，并清除原有内容

(2) 输入流类的 open 函数

[程序] 使用默认构造函数建立对象，然后调用 open 成员函数打开文件。

ifstream infile; infile.open("filename", iosmode);

也可以使用指针。

ifstream * pfile;pfile->open("filename", iosmode);

使用有参构造函数

```
ifstream infile ("filename", iosmode);
ios_base::in 打开文件用于输入(默认)。
```

ios_base::binary 指定文件以二进制方式打开，默认为文本文件。

(3) close 函数：用来关闭与一个文件流相关联的磁盘文件。

(4) 错误处理函数

bad()：如果进行非法操作，返回 true，否则返回 false。

clear()：设置内部错误状态，可清除所有错误位。

eof()：若提取操作已到文件末尾，返回 true，否则返回 false。

good()：如果没有错误条件和没有设置文件结束标志，返回 true，否则返回 false。

fail()：与 good 相反，操作失败返回 false，否则返回 true。

C++ 程序设计基础知识

一、数据类型

1. 注释

```
(1) //注释单行
(2) /* */ 可注释多行
```

2. 分号 块语句必须以“;”结束。

由分号标识一个块，以构造一个复合语句。

3. 关键字

表 10-1 ANSI C++ 标准规定的关键字

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

表 10-2 ANSI C++ 标准补充的关键字

bool	catch	class	const_cast	delete	dynamic_cast
explicit	false	friend	inline	mutable	namespace
new	operator	private	protected	public	reinterpret_cast
static_cast	template	this	throw	true	try
typeid	typename	using	virtual	wchar_t	

[注]

(1) 支持 sizeof 运算符，能计算出数据类型的字节长度，所有的数据类型对于任何硬件平台可能不是定长的（与 JAVA 不同）。

(2) goto 为关键字，但在 C++ 中不提倡使用。

(3) 基本数据类型。

表 10-3 C++ 的基本数据类型

类型名	长度(字节)	取值范围
bool	1	false, true
char(signed char)	1	-128 ~ 127
unsigned char	1	0 ~ 255
short(signed short)	2	-32768 ~ 32767
unsigned short	2	0 ~ 65535
int(signed int)	4	-2147483648 ~ 2147483647
unsigned int	4	0 ~ 4294967295
long(signed long)	4	2147483648 ~ 2147483647
unsigned long	4	0 ~ 4294967295
float	4	$3.4 \times 10^{-38} \sim 3.4 \times 10^{38}$ (绝对值精度)
double	8	$1.7 \times 10^{-308} \sim 1.7 \times 10^{308}$ (绝对值精度)
long double	8	$1.7 \times 10^{-308} \sim 1.7 \times 10^{308}$ (绝对值精度)

C++ 是强类型语言，要求程序设计者在使用数据之前对数据的类型进行声明。

4. 变量定义

(1) 变量命名：C++ 是大小写敏感的。

遵守如下的规则：

① 大小写字母、下划线和数字组成。

② 不能是 C++ 关键字，不能与 C++ 的库函数名、类名和对象名同。

③ 中间不能有空格。

(2) 命名规则：常采用匈牙利命名法。

(3) 变量定义方式：在一个语句里建立多个同一类型的变量。

例如：int iStudentNo, iPenCount, iTemp;

(4) 赋值与初始化：例如：int Temp, iPenCount = 100；

(5) typedef 提供已有数据类型或自己定义类型的同义词。

例如：typedef int INT, integer; INT i;

5. 常量

(1) 整型常量。

(2) 实型常量。

(3) 字符常量。

表 10-4 C++ 中的字符转义序列

转义序列	对应值	功能
\a	7	响铃
\b	8	退格
\f	12	换页
\n	10	换行

二、运算符和结合性

表 10-5 运算符和结合性			
优先级	运算符	功能	结合性
1	::	类域	左结合
	::	全局域	
	.	成员选择	
	- >	成员选择	
	[]	下标	左结合
2	()	函数调用	
	()	类型构造	
	++	后缀增量	
	--	后缀减量	
	sizeof	变量长度	右结合
	sizeof	类型长度	
	++	前缀增量	
	--	前缀减量	
	~	按位取反	
	!	逻辑非	右结合
	-	单目减	
	+	单目加	
	&	取地址	
	*	值引用	
	new	动态分配	

优先级	运算符	功能	结合性
3	delete	变量释放	
	delete []	数组释放	右结合
	()	强制类型转换	
	*	指向成员选择	
4	- > *	指向成员选择	左结合
	*	乘	
5	/	除	左结合
	%	取余(取模)	
6	+	加	左结合
	-	减	
7	<<	按位左移	左结合
	>>	按位右移	
	<	小于	
8	<=	小于等于	左结合
	>	大于	
	>=	大于等于	
9	==	等于	左结合
	!=	不等于	
10	&	按位与	左结合
11	,	按位异或	左结合
12		按位或	左结合
13	&&	逻辑与	左结合
14		逻辑或	左结合
15	:	条件表达式	右结合
	=	赋值	
	*=	乘赋值	
	/=	除赋值	
16	%=	取余赋值	右结合
	+=	加赋值	
	-=	减赋值	
	<<=	按位左移赋值	

三、分支结构

1. if语句

格式 1: if(条件表达式)
语句;
格式 2: if(条件表达式)

语句;

格式 3: if(条件表达式)
语句;
语句;else
语句;

成对规则: else 连接到上面第一个没有配对的且为可见的 if 上。

2. 条件运算符
(条件表达式)? (条件为真的表达式) : (条件为假时的表达式)
如: x = a < b? a:b

3. switch语句: 是多分支选择语句

switch(表达式)

case 常量表达式 1: 语句组 1

case 常量表达式 2: 语句组 2

...

case 常量表达式 n: 语句组 n

default: 语句组 n + 1

执行过程如下:

(1) switch后面括号中的表达式只能是整型、字符型或枚举表达式。 case 后面的常量表达式类型必须与其匹配。
(2) 当表达式的值与某一个 case 后面的常量表达式值相等时, 就执行 case 后面的语句, 都没有匹配时, 就执行 default 后面的语句。

(3) case 后面的表达式起标号作用, 标号不能重名。

(4) case 和 default 不改变控制流程, 必须加 break。

(5) 各个 case 的出现次序可以任意。每个 case 分支都带 break 时, 次序不影响执行结果。

(6) 多个 case 可以共用一组执行语句。

(7) default 是可选的。

(8) switch语句可以嵌套。

(9) 用 if 语句与 switch 可以相互补充。

四、循环

```
1. while 语句
while(条件表达式)
```

循环体//循环体有可能一次也不执行

```
2. do...while 语句
do {
```

循环体//循环体至少执行一次
 | while(条件表达式);

```
3. for 语句
for (表达式1;表达式2;表达式3)
```

循环体

 |

(1) 先求解表达式 1。

(2) 求解表达式 2, 若为 0, 则结束循环, 转到(5)。

(3) 若表达式 2 为真, 执行循环体, 然后求解表达式 3。

(4) 转向(2)。

(5) 执行语句下面的一个语句。

4. 跳转语句

(1) break 语句

break 语句用在 while, do...while, for 和 switch 语句中。
在 switch 语句中, break 用来使流程跳出 switch 语句, 继续执行 switch 语句后的语句。

在循环语句中, break 用来从最近的封闭循环体内跳出。

(2) continue 语句

continue 语句用在循环语句中, 作用为结束本次循环, 即跳过循环体中尚未执行的语句结束循环, 接着进行下一次是否执行循环的条件判定。

(3) goto 无条件转移, 不建议使用。

五、函数

1. 函数原型

在 C++ 中不允许使用未经声明的变量与函数。函数声明格式:

 | 类型声明符 函数名称(参数列表)

 | 如:int reinteger(int arg);

2. 函数定义

将一个已声明的函数实体化。包含两部分: 函数声明和函数体。

 | 例如:

```
int noreturn()
```

 | ...

(3) inline 函数(内联函数): 编译器执行 inline 函数的方式是直接将该函数代码放入

调用该 inline 函数的程序中, 将一函数设置成 inline 函数, 必须完成两件工作: 在函数定义前加上关键字 inline 和该函数的定义必须在第一次被调用前出现。

inline 函数与宏的区别: 宏属于文字替换, inline 函数属于程序代码的替代。

(4) 函数参数缺省值
 | 类型声明符 函数名称(参数1, 参数2, 参数3 = 参数值1, 参数4 = 参数值4);

在一个具有多个参数的函数中, 用户不能只为特定参数做缺省值设置,除非该参数右边参数都设置了缺省值。

注意: 参数缺省值设置不可以同时出现在函数原型与函数定义中。

(5) 函数名重载

在 C++ 中允许函数名称重复使用。在一个有效范围内可以有两个或两个以上的函数名称与返回值完全一样。重载函数至少要在参数个数或参数类型上有所不同。

(6) 函数指针

① 函数指针声明: 函数指针必须使得该指针与所指向的函数具有相同的参数表与返回值。

如: double max(int a, int b);

函数指针格式: double (* pf)(int, int);

错误的声明: double * pf(int, int);

② 函数指针的赋值

直接赋值: pf = max;

定义时赋值: double (* pf)(int, int) = max;

函数声明: double (* pf)(int, int) = &max;

③ 函数指针的调用

double result = (* pf)(3, 1);

double result = pf(3, 1); // 同 double result = max(3, 1);

错误的调用: double result = * pf(3, 1);

④ 函数指针数组:

考虑以下函数: void DiskCopy();

void DiskFormat();

void DiskCompare();

可以利用函数指针数组将这些函数组合在一起。

```
void ( * OpDisk[ ] )( ) = { DiskCopy, DiskFormat, DiskCompare } ;
```

调用方法:

```
( * OpDisk[ 2 ] )( ) // 等于调用DiskCompare()
```

六、C++ 的程序结构

1. 全局变量与局部变量

变量在整个程序中都是可见的, 称为全局变量。变量只能在一个函数中可知, 并在程序的每个函数中都是可见的。全局变量由编译器建立, 并且初始化为 0。

(2) 局部变量: 在函数内部定义的变量。仅在函数内部是可见。

2. 静态局部变量

在局部变量前加上“static”关键字。存放在内存中的全局数据区。函数结束时, 静态局部变量不会消失, 每次该函数调用时, 不会为其重新分配空间。直到程序运行结束。

3. 外部存储类型

一般稍微大一点的程序都由多个源文件组成。只有一个源文件具有 main() 构成一个程序的多个源文件之间, 通过声明数据或函数为外部的(extern)来进行沟通。

```
(1) cpp: int gCount = 100; void main() { }
```

在一个具有多个参数的函数中, 用户不能只为特定参数做缺省值设置,除非该参数右边参数都设置了缺省值。

注意: 参数缺省值设置不可以同时出现在函数原型与函数定义中。

(5) 函数名重载

在 C++ 中允许函数名称重复使用。在一个有效范围内可以有两个或两个以上的函数名称与返回值完全一样。重载函数至少要在参数个数或参数类型上有所不同。

(6) 函数指针

① 函数指针声明: 函数指针必须使得该指针与所指向的函数具有相同的参数表与返回值。

```
static int gCount = 100; // 全局变量
```

```
void main() { }
```

(2) 静态函数: 在函数前加上“static”关键字。存放在内存中的全局数据区。对于一个文件没有什么区别, 对于多个文件是有区别的。全局静态变量使得该变量成为定义该变量的源文件所独享。

5. 数组

(1) 数组的定义: 一个数组中的所有元素具有相同的数据类型。可以声明任何基本数据类型的数组, 包括用户自定义类型的对象数组。在 C++ 中的类对象数组)。在 C++ 中的数据数组是固定大小的, 总是保持同样的大小, 生命周期同变量定义。编程时, 若要定义一个很大的数组, 宜采用 new 进行分配。

(2) 访问数组的元素: 长度为 n 的数组, 其下标范围为 0 ~ (n - 1)。

3. 初始化数组

(1) 数组的初始化: 在数组被定义时, 使包含程序立即能使用的值。

全局数组和全局静态数组若没有明确初始化, 系统自动初始化 0, 而且在程序进入主函数之前就完成了, 而局部数组没有初始化除非显式初始化。

2. 初始化字符串数组

```
char sName[ 4 ] = " neu" ;
```

(3) 省略数组的大小: int iCounts[] = { 1, 2, 3, 4, 5 } ;

(4) 间接函数传递数组: 将数组传递给函数, 实际上是把数组的地址传给函数。

```
int iCounts[ 10 ] = { 1, 1, 2, 2, 3, 3, 4, 4, 5, 5 } ; // 全局数组
int sum( int iC[ ], int ) ;
int sum( iCounts, 5 ) ;
```

七、指针

1. 指针概念

(1) 指针类型: 基本类型相对应, int, float, char, double 等, 每一种都有相应的指针类型。

2. 指针变量的定义:

```
int * ip; const int * icp; char * pszName;
```

(3) 建立指针: 变量存在于内存中的某位置(地址)。用 & 操作符来取得变量的地址, 指针变量用于存放地址...

(4) 间接引用指针: 间接引用指针时, 可获得由该指针指向的变量内容。

```
int iCount = 18; int * iPr = &iCount;
cout << * iPr << endl;
```

通过指针，可以修改指向的变量内容。

```
int iCount = 18; int * iPr = &iCount;
*iPr = 23; //实际是修改 iCount 的值
```

2. 指针运算

指针可以进行加减运算。可以把数组起始地址赋给一指针，通过移动指针（加减指针）来对数组元素进行操作。

```
int sum = 0, iCounts[] = { ... };
iPr = iCounts; //用数组名给指针初始化
for( int i = 0; i < 10; i++ )
    iCounts[ i ] = i * 2; sum += *( iPr + i );
cout << " sum = " << sum << endl;
```

指针是具有某个数据类型的地址。指针运算都是以数据类型为单位进行展开的。以

上例子 iPr 是整型指针。iPr + + 使指针指向下一个，地址值增加了 4。

对于不同类型的指针，指针 + +，实际地址增加是不一样的。

对于整型指针，指针 + +，实际地址增加是 4。

对于字符指针，指针 + +，实际地址增加是 1。

对于长整型指针，指针 + +，实际地址增加是 4。

3. 指针与数组

数组名可以初始化指针，数组名就是数组第一个元素地址。

```
int a[ 100 ];
int * iPr = a;
第 i 个元素：a[ i ] 等价于 * ( a + i ) 等价于 iPr[ i ] 等价于 * ( iPr + i )。
第 i 个元素的地址：&a[ i ] 等价于 a + i 等价于 iPr + i 等价于 &iPr[ i ]。
```

4. new 与 delete

它们都不用头文件声明，new 的操作数为数据类型，它可以带初始化值或单元个数。它返回一个具有操作数的数据类型的指针。delete 的操作数是 new 返回的指针。

```
char * pszName; pszName = new char[ 4 ]; ...;
```

delete pszName;

5. 字符指针

```
(1) 字符数组
char szBuffer[] = "hello";
(2) 字符指针
char * pszName, * p; pszName = new char[ 4 ];
if( pszName != NULL )
{
    p = pszName;
    strcpy( p, "new" ); //字符串的赋值
}
```