C++程序设计精要教程

华中科技大学

第4章 C++的类

◆4.1 类的声明及定义

类保留字：class、struct或union可用来声明和定义类。

- 类的声明由保留字class、struct或union加上类的名称构成。
- 类的定义包括类的声明部分和类的由{ }括起来的主体两部分构成。
- 类的实现通常指类的函数成员的实现，即定义类的函数成员。

```
class 类型名;//前向声明
class 类型名{
private:
    私有成员声明或定义;
protected:
    保护成员声明或定义;
public:
    公有成员声明或定义;
};
```

第4章 C++的类

◆4.1 类的声明及定义

定义类时应注意的问题：

- **使用private、protected和public保留字标识主体中每一区间的访问权限，同一保留字可以多次出现；**
- **同一区间内可以有数据成员、函数成员和类型成员，习惯上按类型成员、数据成员和函数成员分开；**
- **成员在类定义体中出现的顺序可以任意，函数成员的实现既可以放在类的外面，也可以内嵌在类定义体中；但是数据成员的定义顺序与初始化顺序有关。**
- **若函数成员在类定义体外实现，则在函数返回类型和函数名之间，应使用类名和作用域运算符“::”来指明该函数成员所属的类。**

第4章 C++的类

◆4.1 类的声明及定义

定义类时应注意的问题：

- 在类定义体中允许对所数据成员定义默认值，若在构造函数的“：”和函数体的“{”之间对其进行了初始化，则默认值无效，否则用默认值初始化；
- 类定义体的最后一个花括号后要跟有分号作为定义体结束标志。
- 构造函数和析构函数都不能定义返回类型。
- 如果类没有自定义的构造函数和析构函数，且有非公开实例数据成员等情形，则C++为类生成默认的参数表无参的构造函数和析构函数。
- 构造函数的参数表可以出现参数，因此可以重载。

第4章 C++的类

◆4.1 类的声明及定义

定义类时应注意的问题：

- **构造函数和析构函数：是类封装的两个特殊函数成员，都有固定类型的隐含参数this 。**
- **构造函数：函数名和类名相同的函数成员。**
- **析构函数：函数名和类名相同且带波浪线的参数表无参函数成员。**
- **定义变量或其生命期开始时自动调用构造函数，生命期结束时自动调用析构函数。**
- **同一个对象仅自动构造一次。构造函数是唯一不能被显式（人工，非自动）调用的函数成员。**

第4章 C++的类

◆4.1 类的声明及定义

定义类时应注意的问题：

- 构造函数用来为对象申请各种资源，并初始化对象的数据成员。构造函数有隐含参数this,可以在参数表定义若干参数，用于初始化数据成员。
- 析构函数是用来毁灭对象的，析构过程是构造过程的逆过程。析构函数释放对象申请的所有资源。
- 析构函数既能被显式调用，也能被隐式（自动）调用。由于只有一个固定类型的this，故不可能重载，只能有一个析构函数。
- 若实例数据成员有指针，应当防止反复析构（用指针是否为空做标志）。
- 联合也是类，可定义构造、析构以及其它函数成员。

第4章 C++的类

例4.1】定义字符串类型和字符串对象。

```
#include <alloc.h>
struct STRING {
    typedef  char *CHARPTR;      //定义类型成员
    CHARPTR  s;                  //定义数据成员
    int  strlen( );              //声明函数成员，求谁的长(有this)
    STRING(CHARPTR);             //声明构造函数，有this
    ~STRING( );                  //声明析构函数，有this
};
int STRING::strlen( ){           //用运算符::在类体外定义
    int  k;
    for(k=0; s[k]!=0; k++);
    return k;
}
```

第4章 C++的类

```
STRING::STRING(char *t){//用::在类体外定义构造函数,无返回类型
    int  k;
    for(k =0; t[k]!=0; k++);
    s=(char *)malloc(k+1);          //s等价于this->s
    for(k=0; (s[k]=t[k])!=0; k++);
}
STRING::~STRING( ) {//用::在类体外定义析构函数,无返回类型
    free(s);
}
struct  STRING  x("simple");      //struct可以省略
void main( ){
    STRING  y("complex"), *z=&y;
    int  m=y.strlen( );                  //当前对象包含的字符串的长度
    m=z->strlen( );
}   //返回时自动调用y的析构函数
```

第4章 C++的类

◆4.1 类的声明及定义

程序不同结束形式对对象的影响：

- **exit退出：局部自动对象不能自动执行析构函数，故此类对象资源不能被释放。静态和全局对象在exit退出main时自动执行收工函数析构。**
- **abort退出：所有对象自动调用的析构函数都不能执行。局部和全局对象的资源都不能被释放，即abort退出main后不执行收工函数。**
- **return返回：隐式调用的析构函数得以执行。局部和全局对象的资源被释放。**

```
int main( ){ …; if (error) return  1; …;}
```

- **提倡使用return。如果用abort和exit，则要显式调用析构函数。另外，使用异常处理时，自动调用的析构函数都会执行。**

第4章 C++的类

例4.2本例说明exit和abort的正确使用方法。

```
#include <process.h>
#include  “string.cpp”      //不提倡这样include：因为string.cpp内有函数定义
STRING x("global");          //自动调用构造函数初始化x
void main(void){
    short error=0;
    STRING y("local");       //自动调用构造函数初始化y
    switch(error) {
    case  0: return;         //正常返回时自动析构x、y
    case  1: y.~STRING( );  //为防内存泄漏，exit退出前必须显式析构y
             exit(1);
    default: x.~STRING( );  //为防内存泄漏，abort退出前须显式析构x、y
             y.~STRING( );
             abort( );
    }
}
```

第4章 C++的类

◆4.1 类的声明及定义

接受与删除编译自动生成的函数：default接受, delete：删除。

例4.4使用delete禁止构造函数以及default接受构造函数。

```
struct A {
    int x=0;
    A( ) = delete;                  //删除产生构造函数A( )
    A(int m): x(m) { }
    A(const A&a) = default;//接受编译生成的拷贝构造函数A(const A&)
};
void main(void) {
    A x(2);                                  //调用程序员自定义的单参构造函数A(int)
    A y(x);                         //调用编译生成的拷贝构造函数A(const A&)
    //A u;                          //错误：u要调用构造函数A( )，但A( )被删除
    A v( );                         //正确：说明外部无参非成员函数v，且返回类型为A
}// “A v( );”等价于“extern  A v( );”
```

第4章 C++的类

◆4.2 成员访问权限及其访问

- 封装机制规定了数据成员、函数成员和类型成员的访问权限。包括三类：
  - private：私有成员，本类函数成员可以访问；派生类函数成员、其他类函数成员和普通函数都不能访问。
  - protected：保护成员，本类和派生类的函数成员可以访问，其他类函数成员和普通函数都不能访问。
  - public：公有成员，任何函数均可访问。
- 类的友元不受这些限制，可以访问类的所有成员。另外，通过强制类型转换可突破访问权限的限制。
- 构造函数和析构函数可以定义为任何访问权限。不能访问构造函数则无法用其初始化对象。

第4章 C++的类

【例4.5】为女性定义FEMALE类。

```
class  FEMALE{          //缺省访问权限为private
   int  age;        //私有的，自己的成员和友员可访问
public:                 //访问权限改为public
   typedef char *NAME;    //公有的，都能访问
protected:              //访问权限改为protected
   NAME nickname;  //自己的和派生类成员、友员可访问
   NAME getnickname( );
public:                 //访问权限改为public
   NAME name;           //公有的，都能访问
};
```

第4章 C++的类

```
FEMALE::NAME FEMALE::getnickname( ){
    return  nickname; //自己的函数成员访问自己的成员
}
void  main(void){      //main没有定义为类FEMALE的友员
    FEMALE  w;
    FEMALE::NAME(FEMALE::*f)( ); //原书报不可访问错
    FEMALE::NAME n;
    n=w.name;                  //任何函数都能访问公有name
    n=w.nickname;              //错误，main不得访问保护成员
    n=w.getnickname( );  //错误，main不得调用保护成员
    int  d=w.age;              //错误，main不得访问私有age
    f=&FEMALE::getnickname; //错误,不得取保护成员地址
}
```

第4章 C++的类

◆4.3 内联、匿名类及位段

- 函数成员的内联说明：
  - 在类体内定义的任何函数成员都会自动内联。
  - 在类内或类外使用inline保留字说明函数成员。
- 内联失败：有分支类语句、定义在使用后，取函数地址，定义(纯)虚函数。
- 内联函数成员的作用域局限于当前代码文件。
- 匿名类函数成员只能在类体内定义(内联)。
- 函数局部类的函数成员只能在类体内定义(内联)，某些编译器不支持局部类。

第4章 C++的类

◆4.3 内联、匿名类及位段

- **对于没有对象的匿名联合，C++兼容C的用法：**
  - **没有对象的全局匿名联合必须定义为static，局部的匿名联合不能定义为static ；**
  - **匿名联合内只能定义公有数据成员；**
  - **数据成员和联合本身的作用域相同；**
  - **数据成员共享存储空间。**

```
#include <iostream>
static union { int x, y, z; };
//int y=5;              //错：本作用域已定义y
void main(void){
   x=3; std::cout<<y; //输出3
}
```

相当于定义：
static int x;
static int &y=x;
static int &z=x;
x,y,z作用于当前文件

第4章 C++的类

◆4.3 内联、匿名类及位段

- **位段成员：按位分配内存的数据成员。**
  - **class、struct和union都能定义位段成员；**
  - **位段类型必须是字节数少于整数类型的类型，如：**
    char，short，int，long long, enum（实现为int：简单类型）
  - **相临位段成员分配内存时,可能出现若干位段成员共处一个字节，或一个位段成员跨越多个字节。因按字节编址，故位段无地址。**
- **位段用法：**
  - **用于生产过程控制的各种开关、指示灯等；**
  - **布尔运算、图象处理等；**
  - **位段成员不能取地址，因现代计算机按字节编址。**

第4章 C++的类

◆4.4 new和delete

- 内存管理的区别：
  - C不必实现函数malloc、free；C++必须实现运算符new、delete。
  - 内存分配：malloc为函数，参数为值表达式；new为运算符，操作数为类型表达式，先底层调用malloc，然后调用构造函数；
  - 用“new 类型表达式 {}”可使分配的内存清零，若“{}”中有数值可用于初始化。
  - 内存释放：free为函数，参数为指针类型值表达式，直接释放内存；delete为运算符，操作数为指针类型值表达式，先调用析构函数，然后底层调用free。
- 如为简单类型(没有构造、析构函数)分配和释放内存，则new和malloc、delete和free没有区别，可混合使用：比如new分配的内存用free释放。
- 无法用malloc代替new初始化, new调用的构造函数可维护多态。
- 注意delete的新参类型应为const void*，因为它可接受任意指针实参。

第4章 C++的类

◆4.4 new和delete

- **new <类型表达式> //后接()或{ }用于初始化或构造。{}可用于数组元素**
  - **类型表达式：int *p=new int; //等价int *p=new int(0);**
  - **数组形式仅第一维下标可为任意表达式，其它维为常量表达式：int (*q)[6][8]=new int[x+20][6][8];**
  - **为对象数组分配内存时，必须调用参数表无参构造函数**
- **delete <指针>**
  - **指针指向非数组的单个实体：delete p; 可能调析构函数。**
- **delete [ ]<数组指针>**
  - **指针指向任意维的数组时：delete [ ]q;**
  - **如为对象数组，对所有对象(元素)调用析构函数。**
  - **若数组元素为简单类型，则可用delete <指针>代替。**

第4章 C++的类

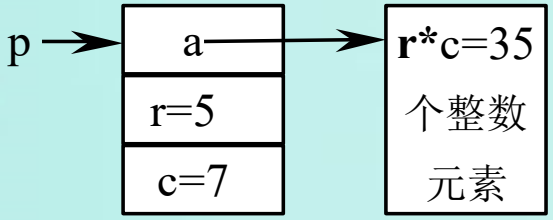
【例4.13】定义二维整型动态数组的类。

```
#include <alloc.h>
#include <process.h>
class  ARRAY{                          //class体的缺省访问权限为private
    int   *a, r, c;
public:                                //访问权限改为public
    ARRAY(int x, int y);
    ~ARRAY( );
};
ARRAY::ARRAY(int x, int y){
    a=new int[(r=x)*(c=y)];            //可用malloc：int为简单类型
}
```

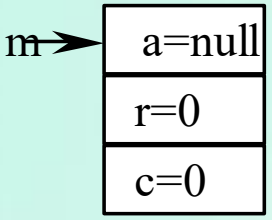
第4章 C++的类

```
ARRAY::~ARRAY( ){    //a指向的简单类型int数组无析构函数
   if(a){ delete [ ]a; a=0;} //可用free(a), 也可用delete a
}
ARRAY  x(3, 5);            //开工函数构造，收工函数析构x
void main(void){
   int error=0;
   ARRAY  y(3, 5), *p;     //退出main时析构y
   p=new ARRAY(5, 7);   //不能用malloc，ARRAY有构造函数
   delete  p;                  //不能用free，否则未调用析构函数
}                                 //退出main时，y被自动析构
//程序结束时，收工函数析构全局对象x
```

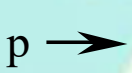
第4章 C++的类



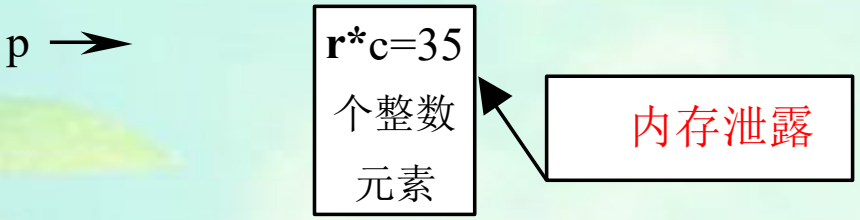
(a) p=new ARRAY(5,7)



(b) m=(ARRAY*)malloc(sizeof(ARRAY))



(c) delete p



(d) free (p)

第4章 C++的类

◆4.4 new和delete

- **new还可以对已经析构的变量重新构造。可以减少对象的说明个数，提高内存的使用效率。(不是所有C++编译器都支持)**

```
STRING  x ("Hello!"), *p=&x;
x. ~STRING ( );
new (&x) STRING ("The World");
new (p) STRING ("The World");
```

- **这种用法可以节省内存或栈的空间。**

第4章 C++的类

◆4.5 隐含参数this

- this指针是一个特殊的指针，它是普通函数成员隐含的第一个参数，其类型是指向要调用该函数成员的对象的const指针。
- 当对象调用函数成员时，对象的地址作为函数的第一个实参首先压栈，通过这种方式将对象地址传递给隐含参数this。
- 构造函数和析构函数的this参数类型固定。例如A::~A()的this参数类型为

  A*const this; //析构函数的this指向可写对象，但this本身是只读的
- 注意：可用*this来引用或访问调用该函数成员的普通、const或volatile对象；类的静态函数成员没有隐含的this指针；this指针不允许移动。

第4章 C++的类

【例4.16】在二叉树中查找节点。

```
#include <iostream>
class  TREE{
    int   value;
    TREE  *left, *right;
public:
    TREE (int); //this类型: TREE * const this
     ~TREE( );     //this类型: TREE * const this，析构函数不能重载
      const TREE *find(int)const; //this类型: const TREE * const this
};
TREE::TREE(int value){          //隐含参数this指向要构造的对象
    this->value=value;          //等价于TREE::value=value
    left=right=0;               //C++提倡空指针NULL用0表示
}
```

第4章 C++的类

```
TREE::~TREE( ){            //this指向要析构的对象
   if(left) { delete left; left=0; }
   if(right) { delete right; right=0; }
}
const TREE* TREE::find(int v) const {//this指向调用对象
   if(v==value) return this;//this指向找到的节点
   if(v<value) //小于时查左子树，即下次递归进入时新this=left
      return  left!=0?left->find(v):0;
   return  right!=0?right->find(v):0; //否则查右子树
}
TREE root(5);                          //收工函数将析构对象root
void main(void){
   if(root.find(4)) std::cout<<“Found\n”;
}
```

第4章 C++的类

◆4.6 对象的构造与析构

- 类可能会定义只读和引用类型的非静态(static)数据成员，在使用它们之前必须初始化；若无默认值，该类必须定义构造函数初始化这类成员。
- 类A还可能定义类B类型的非静态对象成员，若对象成员必须用带参数的构造函数构造，则A的对象必须定义有初始化构造函数(自定义的类A的构造函数，传递实参初试化类B的非静态对象成员：缺省的无参的A( )只调用无参的B( ))。
- 构造函数的初始化位置在参数表的“:”后，所有数据成员都必须在此初始化，未列出的成员用其默认值值初始化，未列出且无默认值的非只读、非引用、非对象成员的值根据对象存储位置可取随机值（栈段）或0及nullptr值（数据段）。
- 按定义顺序初始化或构造数据成员（大部分编译支持）。

第4章 C++的类

◆4.6 对象的构造与析构

- 如未定义或生成构造函数，则可以用”{}”的形式初始化。联合仅需初始化第一个成员。
- 对象数组的每个元素都必须初始化，默认采用无参构造函数初始化。
- 单个参数的构造函数能自动转换单个实参值成为对象
- 若类未自定义构造函数，且类包含私有实例数据成员等调价满足时，编译会自动生成构造函数。
- 一旦自定义构造函数，将不能接受编译生成的构造函数，除非用default等接受。
- 用常量对象做实参，总是优先调用参数为&&类型的构造函数；用变量等做实参，总是优先调用参数为&类型的构造函数。

第4章 C++的类

【例4.17】包含只读、引用及对象成员的类。

```
class A{
    int  a;
public:
    A(int x) { a=x;} //重载构造函数，自动内联
    A( ){ a=0; }       //重载构造函数，自动内联
};
class B{
    const  int  b;        //数据成员不能在定义的同时初始化
    int  c, &d, e, f;     //b,d,g,h只能在构造函数体前初始化
    A    g, h;   //数据成员按定义顺序b, c, d, e, f, g, h初始化
```

第4章 C++的类

```
public: //类B构造函数体前未出现h，故h用A( )初始化
   B(int y): d(c), c(y), g(y) ,b(y), e(y){//自动内联
      c+=y;            f=y;
   }//f被赋值为y
};
void main(void){
   int x(5);                  //int x=5等价于int x(5)
   A a(x), y=5;               //A y=5等价于A y(5)，请和上一行比较
   A *p=new A[3]{ 1, A(2)}; //初始化的元素为A(1), A(2), A(0)
   B b(7), z=(7,8);           //B z=(7,8)等价于B z(8),等号右边必单值
   delete [ ]p;               //防止内存泄漏：new产生的所有对象必须用delete
}                             //故(7,8)为C的扩号表达式，(7,8)=8
```