華中科技大學

面向对象程序设计

第三章　C++的类

许向阳

xuxy@hust.edu.cn

学习内容

3.1 类的声明及定义

3.2 访问权限

3.3 构造函数与析构函数

3.4 new和delete

3.5 组合类和对象成员的初始化

3.6 类与const

3.7 隐含参数this

3.8 对象作为参数和函数返回

3.1 类的声明及定义

类保留字： class、struct或union可用来声明和定义类。

类的定义：

```
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;
};
```

类的实现： 类的函数成员的实现，即定义类的函数成员。

类的申明： class 类型名;

3.1 类的声明及定义


```
#include <iostream>
using  namespace std;
class Student  {                    //一般 类名的首字母 大写
  private:
        int    number;              //一般数据成员名称小写
        char  name[15];
        float  score;
  public:              //  成员函数的组成单词的首字母一般大写
        void Init(int number1,char *name1,float score1);  //声明
        void Modify(float  score1)   //声明与定义
        {  score=score1;  }
        void  Print();                      // 在体外定义Init 与 Print
};
```

3.1 类的声明及定义


```
void Student::Init(int number1, char *name1, float score1)
{                          // :: 作用域解析运算符
        number = number1;
        strcpy_s(name,name1);
        score = score1;
}

void Student::Print()
{
   cout<< " name : " <<name << " score: " << score << "\n";
}
```


```
int main(int argc, char* argv[])
{
        int     t_number;
        char   t_name[15];
        float   t_score;
        Student  stu1, stu2;
        cout<< "Enter student 1 number : ";   cin >> t_number;
        cout<< "Enter student 1 name : ";             cin >> t_name;
        cout<< "Enter student 1 score : ";            cin >> t_score;
        stu1.Init(t_number, t_name, t_score);
        stu2.Init(t_number,t_name,t_score);
        stu2.Modify(87);
        stu1.Print();              // 访问对象的成员用 . 运算符
        stu2.Print();
        return 0;
}
```

对象的定义和申明

class 类对象的定义和使用，与struct 结构变量的用法相似

```
Student   stu1;       // 对象定义
Student    *p;        // 对象指针
Student    ar[10];    // 对象数组
Student   *par[10];// 对象指针数组
Student    **s2;      // 二级指针
```

stu1. ***

P→***

ar[i]. ***

3.1 类的声明及定义

- private、protected和public 标识每一区间的访问权限
- private、protected和public 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；
- 成员可以任意顺序出现
- 函数成员的实现既可放在类体外，也可在类体中；
- 若函数成员在类的定义体外实现，
  使用“类名::”指明该函数成员所属的类；
- 类的定义体花括号后要有分号作为结束标志。

3.2 访问权限

private：私有成员

仅可被本类的函数成员访问

不能被派生类、其它类和普通函数访问

protected：受保护成员

可被本类和派生类的函数成员访问

不能被其它类函数成员和普通函数访问

public：公有成员

可被任何函数成员和普通函数访问

class定义的类缺省访问权限为private

struct和union定义的类缺省访问权限为public。

3.2 访问权限

全局变量　　作用域的差别

局部变量

类中的变量：公有、受保护、私有

访问：

取值、赋值、引用、调用、取地址、取内容等。

3.2 访问权限


```
class Student
{
private:
        int  number;
        float score;
public:
        char  name[15];
public:
        void Init(int number1,char *name1,float score1);  //声明
        void Modify(float score1)   //声明与定义
        {
                score=score1;
        }
        void Print();                    // 在体外定义init 与 Print
};
```

3.2 访问权限


```
int main(int argc, char* argv[])
{
        Student  stu1, stu2;

        strcpy_s(stu1.name,”xu” );

        stu1.number = 123;

//    error C2248: ‘number' : cannot access private member declared in class 'Student'

         return 0;
}
```

3.2 访问权限

用强制类型转换
方法修改常变量

用强制类型转换方法
访问私有成员？

```
int xx;
cin >>xx;
const int yy = xx;
cout << “yy=” << yy<<endl;
*(int *)&yy = 30;
cout << “yy=” << yy<<endl;
// 显示30
```

```
int main(int argc, char* argv[])
{     Student  stu1, stu2;
      stu1.number = 123;
//    error C2248: ‘number' : cannot access private member declared in class 'Student'
}
```

```
*(int *)&stu1 =123;                  // 用强制类型转换 number
*((float *) &stu1+1)=99.5;           // 未直接私有成员number
                                     // 访问 score
```

3.2 访问权限

能否用非正规方法访问私有成员？

定义一个结构，字段与类相同，然后转换为该结构类型

```
struct  TROJAN_HORSE {
     int  number;
    float score;
    char  name[15];
};

((TROJAN_HORSE ) &stu1) ->score = 99.9;
```

有意识地绕开了编译器对访问权限的检查

3.2 访问权限

华中科技大学

类型成员

在一个类中定义类型

```
public:
    typedef   char *  NAME;    // NAME 等同 char *
protected:
    NAME  nickname;
    NAME  getnickname();

Student::NAME  p;    //  定义一个NAME类型的变量
   // 若将类型定义所属权限修改为私有/保护, 则上一语句非法
Student::NAME  Student::getnickname() { ……}
```

3.3 构造函数与析构函数

在定义一个类时，可以给数据成员赋初值

```
class Student { ……
    int  number=101;
    char name[15] = "hello";
    float score =95;  ……
};
Student  stu1, stu2;
```

stu1、stu2 中数据成员的初值相同。

3.3 构造函数与析构函数

用struct 定义的类时，可以给数据成员赋初值

```
struct Student { ……
    int  number=101;
    char name[15] = "hello";
    float score =95;  ……
};
struct Student  stu1;
struct Student  slist[10];
```

slist[0]、……、slist[9] 中的各成员的初值均相同。

3.3 构造函数与析构函数

若希望Student的两个对象的数据成员值发生变化，怎么做？

```
Student  stu1, stu2;
stu1.Init(t_number1, t_name1, t_score1);
stu2.Init(t_number2, t_name2, t_score2);
```

- 调用函数Init，覆盖类定义中给数据成员所赋初值；
- 定义对象与 给对象的成员赋值 用两个语句表达；

  有无更简洁的表达方法？

3.3 构造函数与析构函数

构造函数

对象的初始化，给成员变量赋值

- 函数名与类名相同
- 没有返回类型（void 也不行）
- 可以有多个构造函数，但参数应有所不同
- **自动调用**
- 对于局部对象，执行到声明对象的代码时，被自动调用
- 对于全局对象，在执行main()之前被自动调用

思考：自动化调用是如何实现的？

3.3 构造函数与析构函数


```
class Student {
private:
        int  number;
        char name[15];
        float score;
public:
        Student(int number1,char *name1,float score1);
        void Modify(float score1)
        {     score=score1;     }
        void Print();
};
```

3.3 构造函数与析构函数


```
Student::Student(int number1,char *name1,float score1)
{       number = number1;
        strcpy_s(name,name1);
        score=score1;
}
      ..........
cout<< "Enter student 1 number : "; cin >> t_number;
cout<< "Enter student 1 name : ";   cin >> t_name;
cout<< "Enter student 1 score : ";  cin >> t_score;

Student stu1(t_number,  t_name,t _score);
stu1.Print();
Student stu1{t_number,  t_name,t _score};
Student stu1={t_number,  t_name,t _score};
```

3.3 构造函数与析构函数


```
    Student stu1(t_number, t_name, t_score);
002655D3 51                   push        ecx
002655D4 F3 0F 10 45 D0       movss       xmm0,dword ptr [t_score]
002655D9 F3 0F 11 04 24       movss       dword ptr [esp],xmm0
002655DE 8D 45 DC             lea         eax,[t_name]
002655E1 50                   push        eax
002655E2 8B 4D F4             mov         ecx,dword ptr [t_number]
002655E5 51                   push        ecx
002655E6 8D 4D B0             lea         ecx,[stu1]
002655E9 E8 D9 BB FF FF       call        Student::Student (02611C7h)
    stu1.Print();
002655EE 8D 4D B0             lea         ecx,[stu1]
002655F1 E8 92 BE FF FF       call        Student::Print (0261488h)
```

缺省的以对象为参数的构造函数

```
Student stu2(stu1);
```

```
mov   eax,dword ptr [ebp-5Ch]
mov   dword ptr [ebp-0BCh],eax
mov   ecx,dword ptr [ebp-58h]
mov   dword ptr [ebp-0B8h],ecx
mov   edx,dword ptr [ebp-54h]
mov   dword ptr [ebp-0B4h],edx
mov   eax,dword ptr [ebp-50h]
mov   dword ptr [ebp-0B0h],eax
```

```
mov   ecx,dword ptr [ebp-4Ch]
mov   dword ptr [ebp-0ACh],ecx
mov   edx,dword ptr [ebp-48h]
mov   dword ptr [ebp-0A8h],edx
```

将 stu1中变量空间中的内容都拷贝到stu2变量所在的位置。

由于中间定义有字符数组，16个字节，要拷贝4次；另有int,float,

总共拷贝 6*4 = 24字节

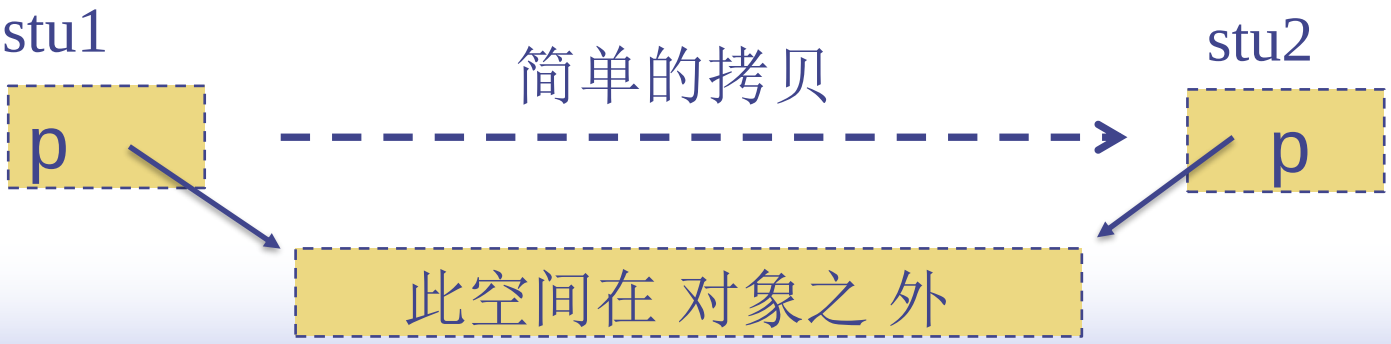
缺省的以对象为参数的构造函数

Student stu2(stu1);

简单地将 stu1中变量空间中的内容都拷贝到stu2的变量中。

浅拷贝 存在的问题

设 Student 中 有变量 char *p;

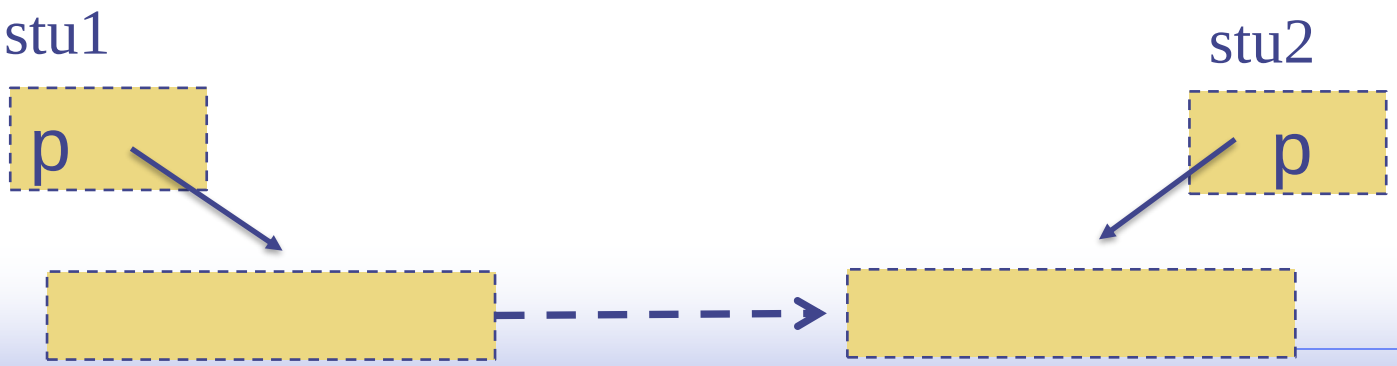


为实现深拷贝，自己编写以对象引用为参数的构造函数

```
Student(Student   &a)
{   ……}
```

Student stu2(stu1);

设 Student 中 有变量 char *p;



3.3 构造函数与析构函数

为实现深拷贝，编写以对象引用为参数的构造函数

```
Student(Student   &a)    // 复制构造函数
{   ……}

Student(Student   *a)    // 普通的构造函数
{   ……}
```

为什么不支持以对象为形参的构造函数？

```
Student(Student   a)    // 非法的复制构造函数
{   ……}
```

3.3 构造函数与析构函数

0个 或多个构造函数

```
Student() { …}                    // 无参数的构造函数

Student(int  x) { … }   // 普通构造函数

Student(Student  &a)    // 复制构造函数
{  ……}
```

在没有定义构造函数时，默认有一个无参数的构造函数，为对象分配相应的空间。

定义了有参数的构造函数，
而没有定义无参数的构造函数，
在定义对象时，必须有相应的参数，不能缺省。

```
Student stu1(t_number, t_name,t _score);

Student stu2;    // 没有合适的默认构造函数可用
```

析构函数

对象撤销时，释放空间或其他处理

- 函数名与类名相同
- 函数名为类名前加 ~
- 没有返回类型
- 无参数函数
- 只能有一个析构函数
- 可以自动调用，也可以在程序中显式调用
- 对象的生命周期结束时，被自动调用

3.3 构造函数与析构函数


```
class Student {
    ……
public:
    ~Student();
};

Student::~Student()
{
   cout<<" Student object " << name << " is destructed "<<endl;
}
```

实验一下：析构函数何时被调用？
为什么会这样？

3.3 构造函数与析构函数


```
void  f(……)
{
    char t_name[15]="zhang";
    Student   zhang(10, t_name, 90) ;
    ……

    Student *wang = new Student(20, t_name, 80) ;
}
```

对象 zhang 在函数 f 结束时，自动析构；
对象指针 wang 指向的对象不会自动析构。

实验一下：析构函数何时被调用？
为什么会这样？

总结

- 构造函数和析构函数与类名相同
- 两者的访问权限一般应为 public，否则无法自动调用
- 可多个构造函数，一个析构函数
- 析构函数无参数
- 都无返回类型
- 构造函数只能在定义对象时自动调用
- 析构函数可以自动和手动调用

何时自动析构，何时要手动析构？

类对象生命周期结束时，会自动调用析构函数。

3.4 new和delete


```
class Student{
private:
        int  number;
        char *name;     // 原来是 char name[15];
        float score;    // 在构造函数中要为name分配空间
public:
        Student(int number1,char *name1,float score1);
        ~Student();
        void Modify(float score1)
        {     score=score1;     }
        void Print();
};
```

3.4 new和delete

- 使用 malloc

```
char   *name;
 #include <malloc.h>
 name = (char *)malloc(15);
```

- 使用 new

```
 name = new char[15];
```

生成一个类对象，怎么办？

```
class  ARRAY  { …. };
ARRAY  *p;
```

如何为 p 分配指向的空间？

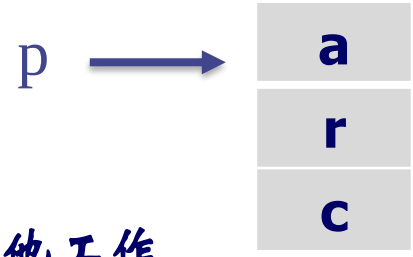
p 指向的ARRAY又如何初始化？

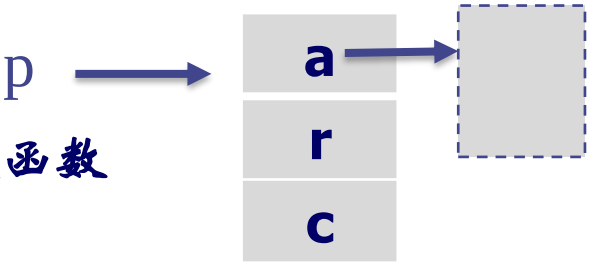
3.4 new和delete


```
class  ARRAY{          //class体的缺省访问权限为private
private:
    int  r , c;        // 行数, 列数
    int  *a,           //  数组元素存放区
public:
    ARRAY(int x, int y)   {
        r=x;  c=y;
        a=new int[x*y];          // int型可用malloc
    }
   ~ARRAY( )     {
        if  (a)  {
           delete [ ]a;   //可用free, 也可用delete a
           a=0;
        }
    }
};
```

3.4 new和delete

```
int main(void){
    ARRAY  *p;
    p = (ARRAY *)malloc(sizeof(ARRAY));
            //只为p分配了指向的空间，未进行其他工作
    free(p);

    p=new ARRAY(5, 7);
            //  分配了空间，并调用ARRAY的构造函数
    delete  p;
            //调用析构函数，释放了空间
    return 0;
}
```





p	0x00380830
a	0xfeeefeee
r	-17891602
c	-17891602

p	0x00380830
a	0x00380ac0
r	5
c	7

3.4 new和delete

華中科技大學

- **new <类型表达式>**

```
int   *p;
int   *pa;
p  = (int *)malloc(sizeof(int));
pa = (int *)malloc(sizeof(int) * 10);
```

数组指针

pa[0], pa[1],.....

```
ARRAY  *q = new ARRAY(5,7);
ARRAY  *qa = new ARRAY[5];
```

数组指针

如果有有参数的构造函数，则必须同时要有无参的构造函数

qa[0], qa[1], …… 都是ARRAY对象

3.4 new和delete

- 对象数组初始化

```
int     x[5];    //  整数数组   x[0],x[1],x[2]…
ARRAY   q[5];    //  对象数组   q[0],q[1],q[2]…
                 //  使用无参数的构造函数
int     x[5]={0,1,2,3,4};
ARRAY   q[5]={{对象q[0]的构造参数},……};
int     x[5] {0,1,2,3,4};
ARRAY   q[5] {{对象q[0]的构造参数},……};
int     x[5]={0,1};
ARRAY   q[5]={{对象q[0]的构造参数}};
        q[1],…,q[4] 要采用无参数的构造函数
```

3.4 new和delete

- 对象数组初始化

```
int     x[5];   //  整数数组   x[0],x[1],x[2]…
ARRAY   q[5];   //  对象数组   q[0],q[1],q[2]…
                //  使用无参数的构造函数
```

```
ARRAY   q[5]={{对象q[0]的构造参数},……};
ARRAY   q[5]={ARRAY(对象q[0]的构造参数),……};
```

3.4 new和delete

```
int     ia[10];    //  整数数组    ia[0],ia[1],ia[2]…
ARRAY   oa[10];    // 对象数组    oa[0],oa[1],oa[2]…

int     *ap = new int[10];     // 数组指针
ARRAY   *aq = new ARRAY[10];   // 数组指针
ARRAY   *sq = new ARRAY(5,7);
```

- delete <指针>　　delete sq;
  - 指针指向非数组的单个实体
  - 如sq指向对象，则自动调用析构函数，再释放对象所占的内存。

- delete [ ] <数组指针>　　delete [] aq;
  - 指针指向任意维的数组时使用
  - 对所有对象(元素) 自动调用析构函数。
  - 若数组元素为简单类型，则可用delete <指针>代替。

3.4 new和delete

```
int     *pa[10];        // 指针数组，有10个指针排在一起
ARRAY   *qa[10];        // 指针数组
```

3.4 new和delete


```
int     *pa[10];  // 指针数组，有10个指针排在一起
int     (*q)[10]; // 数组指针，指向一个数组
```

(1) 定义了一个变量 q

(2) q 是一个指针

(3) 将（*q）视为 A，则有 int A[10],这是一个数组

(4) q是指向长度为10的整型数组的指针

```
q = new  int[3][10];
```

分配的字节数为 3*10*4 。

q=q+1; // q 增加 40个字节。将A[10] 视为一个整体

3.4 new和delete

对于 简单类型(没有构造、析构函数)指针分配和释放内存

- new和malloc、delete和free没有区别
- 可混合使用，如new分配的内存用free释放。

3.5 对象的初始化

类中的各种数据成员如何初始化？

```
Class A {   ……};
Class  B
{
      int  x;
      const int y;     // 只读成员
      int  &z;         //  引用成员
      static int u;    //  静态成员
      A    a;          //   对象成员
      A    *p;
      A    &q;         //   引用成员，引用的是一个对象
     B(……) { ……....}
};
```

能够都在构造函数中{……}初始化吗？

3.5 对象的初始化

类中的各种数据成员如何初始化？

```
Class A {   ……};
Class  B
{     int  x;
       const int y;
       int  &z;
       static int u;
       A    a;
       A    *p;
       A    &q;
      B(int t1,int t2,……): y(t),  z(t2), a(…),  q(…)
          { ……..}
};
```

在构造函数体前初始化：
只读成员、引用成员、对象成员

- 对象指针可在构造函数中初始化
- 静态成员在类外初始化

3.5 对象的初始化

Class A { ……}; 类中的各种数据成员如何初始化？

```
Class  B
{     B(int t1,int t2,……): y(t),  z(t2), a(…),  q(…)
        { ……….}
};
```

在构造函数体前初始化：只读成员、引用成员、对象成员

- 函数说明之后
- { } 之前
- ： 分隔
- 各数据成员以逗号分隔
- 用构造函数的形式给各变量赋初值， 如 y(t)
- 可以用列表的形式{} 赋初值， 如 y{t}
- 不能采用 = 来初始化 : y=t // error

3.5 对象的初始化

在类的定义中，可以对非静态数据成员进行初始化。

```
class B {
  public:
     int   x=10;              // 初始化 x=10
     const  int   y=20;   //  初始化 y= 20
     int   &z = x;            //  z 中的内容为 x的地址
     A    a{…..};             //   类似于构造函数，用{ } 代替( )
     A    *p = new A(…);
     A    &q=a;
};
```

等效于申明后，将初始化赋值放在各构造函数体前。

3.5 对象的初始化

另一种初始化方法

```
class A {
  public:
     int   x {10};       // 初始化 x=10;
     int   &y {x};       //  y 中的内容为 x的地址
    const int  z{20};
    int   u=30;
    int   v(40);         //  ERROR
    A     a(……);         //  ERROR
    A     a{……};
};
```

3.5 对象的初始化

类成员不能像一般的变量那样用 ( ) 来初始化

```
class A {
  public:
     int   x1 ;          // x1 未初始
     int   x2 (10);       // Error;
     int   &y1;          //  y1未初始化
     int   &y2 (x2);  //   error
     const int u1 ;     // u1 未初始
     const int u2 {20};  // u2 初始化为 20
 };

int  x2(10);    // 非类的成员变量，等效 x2=10;
```

3.5 对象的初始化

- 如果常变量、引用变量在定义时未初始化，则在构造函数前进行初始化 。

  Why?

  类比：普通的常变量、引用变量（即不是一个类的数据成员），是应该在申明时就初始化的。因而不应该在构造函数内初始化的。 特殊！

- 类中有数据成员是一个对象时，也要在构造函数前进行初始化。**Why?**

  构造函数只能自动调用，不能写调用构造函数语句。

  数据成员是对象指针时，则可在构造函数中初始化。

3.5 对象的初始化

在类的定义中，可以对非静态数据成员进行初始化。

```
class A {
  public:

     static  int  v =30;   // 错误语句，静态数据成员独立于
                           //  对象而存在
    static const int w=40;  // 有const 约束，可赋值
    const static int  t =50;
};
```

3.5 对象的初始化


```
class ARRAY  {.....};
```

单个ARRAY对象 a1，5行7列的矩阵

```
ARRAY   a1(5,7);        // 调用 ARRAY的构造函数初始化
```

对象数组 a2，10个对象，给定每个对象的行列数

```
ARRAY   a2[10] = {ARRAY(5,7), ARRAY(3,4), … ,ARRAY(2,2)};
```

对象指针

```
ARRAY  *p = new ARRAY(5,7);     //  一个对象的指针
ARRAY  *pa = new ARRAY[10];     //  指向一个对象数组
ARRAY  *q = & a1;               //  a1 是前面定义的一个对象
ARRAY  *qa = a2;                //  a2 是前面定义的一个对象数组
```

用法类比：对象数组 VS 结构数组

对象指针 VS 结构指针

3.5 对象的初始化

```
int   x=5;
int   y;
y=x;
ARRAY   a1(5,7);        // 调用 ARRAR的构造函数初始化
                        //  ARRAY(int x, int y)
ARRAY   a2;             //  在ARRAY中要有无参数的构造函数
                        //  ARRAY( ) { r=0;c=0;a=0; }
```

能否像整型变量那样，将 a1中各数据成员的值拷贝到了a2中？

a2 = a1;

这 种拷贝存在的问题是什么？

浅拷贝

3.5 对象的初始化

组合类对象的初始化

```
class  Date
{
private:
        int  day, month, year;
public:
        Date(int dd, int mm, int yy)
        {
            day = dd;
           month = mm;
           year = yy;
        }
        void Print()
        {
            cout<<year << " - " << month << " - " << day<<endl;
        }
};
```

3.5 对象的初始化

类中含有其他类的对象
组合类对象的初始化

```
class Student
{
private:
        int  number;
        char name[15];
        Date birthday;   // 何时初始化 birthday ?
        float score;
public:
        Student(int number1,char *name1,float score1,int dd,int
mm, int yy);
        ~Student();
        void Modify(float score1);
        void Print();
};
```

3.5 对象的初始化


```
Student::Student(int number1,char *name1,float score1,int
dd,int mm, int yy) : birthday(dd,mm,yy)
{
        number = number1;
        strcpy(name,name1);
        score=score1;
}
```

构造函数体之前初始化，冒号分隔

3.5 对象的初始化

数据成员初始化方法：

- 在定义数据成员时赋初值，等价于在构造函数体前赋初值；
- 在构造函数中赋初值；
- 在构造函数体前赋初值；
- 在定义对象时，自动调用构造函数初始化；
- 按定义的先后次序初始化，与出现在初始化位置列表的次序无关；
- 普通数据成员没有出现在初始化位置时，若所属对象为全局、静态或new的对象，将具有缺省值0；
- 基类和非静态对象成员没有出现在初始化位置时，此时必然调用无参构造函数初始化对其初始化；
- 如果类仅包含公有成员且没有定义构造函数，则可以采用同C兼容的初始化方式，即可使用花括号初始化数据成员；联合类型的对象只须初始化一个成员(共享内存)；

3.6 类与const

常数据成员及其初始化

```
class  A {
  private:
        const int  x;
        int   y;
  public:
        A( int y);
};
```

方法1： **const int x=10;**

方法2： **const int x;**

构造函数体前初始化，
与对象成员一样

A::A(int y) : x(10) { this->y=y; } // 构造函数

3.6 类与const

常成员函数 （const member function）

函数中不允许修改对象的非静态数据成员。

目的：避免对数据成员的误修改

```
class  Date
{
  private:
        int day, month, year;
  public:
        int GetDay()  const
        {       // year = 2014;      错误语句
                return day;
        }
};
```

3.6 类与const

常成员函数 （const member function）

函数中不允许修改对象的非静态数据成员。

```
int GetDay()  const
{       // year = 2014;      错误语句
```

- 可以修改静态成员；
- 不得调用其他非 const的成员函数

```
   //错误   f();    //   void f() {..可以修改数据成员....}
            return day;
   }
};
```

3.6 类与const

常对象

整个对象的数据成员都不允许修改。
在定义常对象时一定要初始化。

类名 const 对象名(实参表);

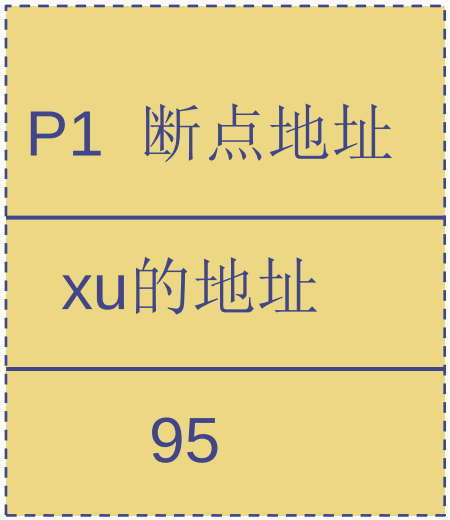
或

const 类名 对象名(实参表);

const Date National_day(1, 10, 1949);

3.7 隐含参数this

```
Student(int number1,char *name1,float score1);
void Modify(float score1)
        {     score=score1;     }

Student   wang(123, “wangji”,97);
Student   xu(125,”xuxiangyang”,90);

      xu.Modify(95);
P1:
      wang.Modify(100);
P2:
```

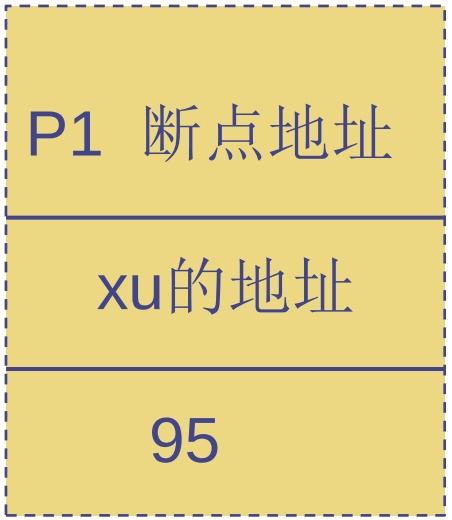


有两个对象，是修改哪个对象中的变量？

VS2019 用 ECX来传递对象的地址

3.7 隐含参数this

```
void Modify(float score1){
      score=score1;
}

void Modify(float score) {
     this->score=score;
    // *this.score=score; 等价语句
    // Student::score=score;
}

Student  xu(125,"xuxiangyang",90);

     xu.Modify(95);
P1:
```



3.7 隐含参数this

- this是一个隐含的const指针，不能移动或对该指针赋值。

  this =p; // =**左操作数必须是左值**

- 普通函数成员的第一个参数，指向调用该函数成员的对象。

  *this代表当前被指向的对象。

- 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，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**。由于析构函数的参数表必须为空，this参数又无类型变化，故析构函数不能重载。

- 类的**静态函数成员**没有隐含的this指针。

3.7 隐含参数this

在树中找节点值为v 的节点
this 相当于树的根指针

```
class  TREE{
    int   value;
    TREE  *left, *right;
public:
    TREE (int);
    ~TREE( );                  //this类型: TREE * const this
    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
}
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) return  left!=0?left->find(v):0;  //查左子树
    return  right!=0?right->find(v):0;     //查右子树，调用时新this=left
}
```

3.8 对象作为参数和函数返回

- 以对象为参数的函数　　*** f (Student s) { …}
- 以对象引用为参数的函数　　*** f (Student &s) { …}
- 以对象为返回结果　　Student g (….) { …}
- 以对象引用返回结果　　Student & g (….) { …}

一个类中不能有函数名相同、参数相同的函数

若有 1、 f (Student s) { …}　2、 f (Student &s) { …}

Student s1(…);

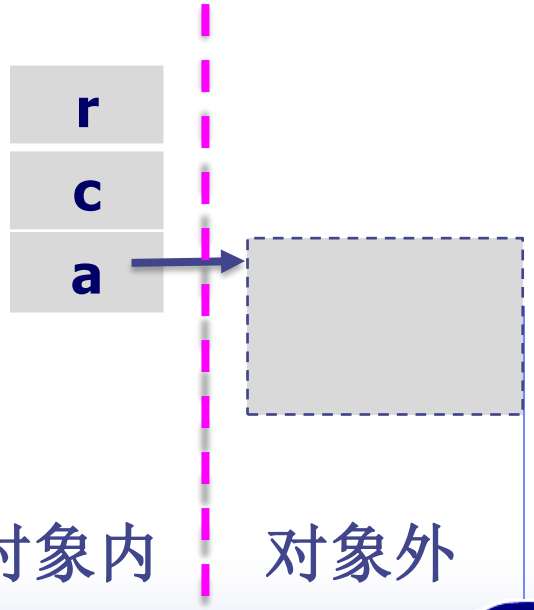
f(s1);　会调用 函数1，还是函数2?

对重载函数的调用不明确

3.8 对象作为参数和函数返回

```
class  ARRAY{
public:
    int  r , c;              // 行数, 列数
    int  *a,                 //  数组元素存放区
public:
    ARRAY(int x, int y)   {
        r=x;  c=y;
        a=new int[x*y];
    }
   ~ARRAY( )     {
        if  (a)  {
           delete [ ]a;
           a=0;
        }
    }
};
```

增加一个函数：
求矩阵中各元素的和



3.8 对象作为参数和函数返回

```
class ARRAY{
public:
    int  r , c;              // 行数, 列数
    int *a,                  // 数组元素存放区
public:
    ARRAY(int x, int y)   { ..... }
   ~ARRAY( )     { ..... }
    int SumElements( )
    { ..... }
};
```

增加一个函数：
求矩阵中各元素的和

- 能否不用成员函数实现一个给定的矩阵中各元素的和？

3.8 对象作为参数和函数返回

方法1： 以对象指针为参数

```
ARRAY  a1(5,7);

int  SumElements(const ARRAY  *received)
{   int  sum=0;
    int  nums = received->r * received->c;
    for (int i=0; i<nums;i++)
        sum += received->a[i];
    return sum;
}

s= SumElements(&a1);  //received 指向 a1 对象
```

函数正确的前提：ARRAY 的成员的访问权限是 public

3.8 对象作为参数和函数返回

方法2： 以对象引用为参数

```
ARRAY  a1(5,7);

int  SumElements(const ARRAY  &received)
{   int  sum=0;
    int  nums = received.r * received.c;
    for (int i=0; i<nums;i++)
        sum += received.a[i];
    return sum;
}

s= SumElements(a1);  //received 指向 a1 对象
```

函数正确的前提：ARRAY 的成员的访问权限是 public

3.8 对象作为参数和函数返回

方法3： 以对象为参数

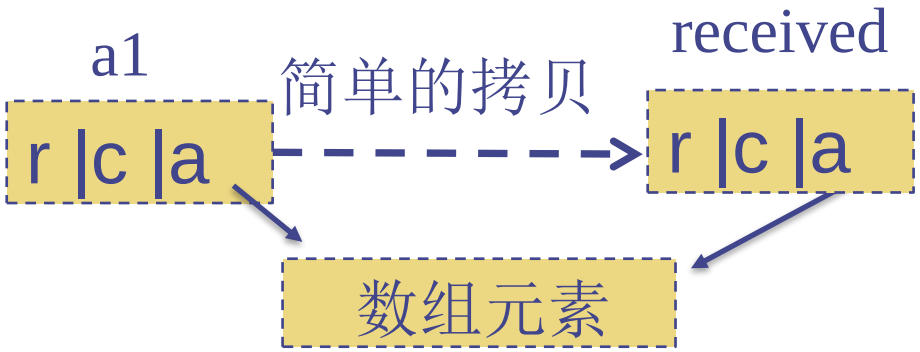
```
ARRAY  a1(5,7);

int  SumElements(const ARRAY  received)
{   int  sum=0;
    int  nums = received.r * received.c;
    for (int i=0; i<nums;i++)
        sum += received.a[i];
    return sum;
}

s= SumElements(a1);
```

函数运行本身未出错，但最后程序崩溃。

为什么？如何解决？

3.8 对象作为参数和函数返回

```
ARRAY  a1(5,7);
s= SumElements(a1);
```



参数传递的实现机制

- 将 **a1** 存储空间中的内容 直接拷贝到 **received**所在空间
- 是一种浅拷贝
- 执行的是默认的以对象为参数的构造函数
- 在**SumElements** 执行结束时，析构对象 **received**
- 在定义**a1**的函数运行结束时，析构对象 **a1**
- 两次释放同一空间

3.8 对象作为参数和函数返回

以对象为参数的复制构造函数

由一个对象复制出一个独立的对象。

classname (const classname &obj) { ……}

在 ARRAY 类中增加成员函数

```
ARRAY (const ARRAY &obj)
{
        r=obj.r;
        c=obj.c;
        a = new int[r*c];
        memcpy(a,obj.a,sizeof(int)*r*c);
}
```

3.8 对象作为参数和函数返回

以对象引用为参数的复制构造函数：

对象复制构造函数 的参数一定要是对象引用！

classname (const classname &obj) { ……}

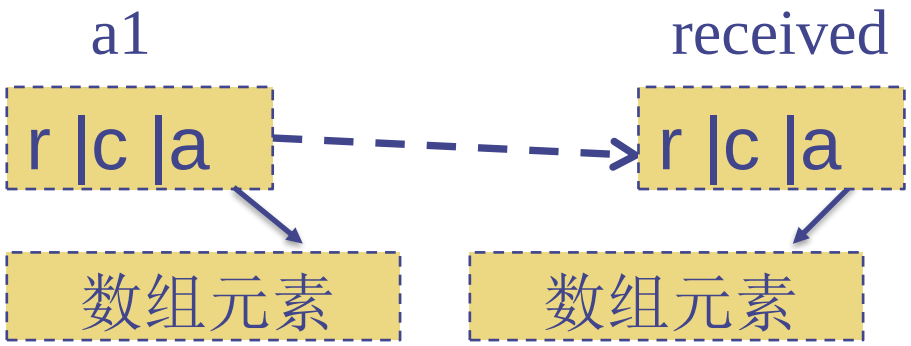
为什么？

不能是 **classname (classname obj) { ……}** 的形式

3.8 对象作为参数和函数返回

int SumElements(const ARRAY received)

ARRAY a1(5,7);
s= SumElements(a1);



参数传递的实现机制

- **调用对象复制构造函数，由a1 构造 received**
- **是一种深拷贝**
- **在SumElements 执行结束时，析构对象 received**
- **在定义a1的函数运行结束时，析构对象 a1**

3.8 对象作为参数和函数返回

复制构造函数被调用的场景：

- ValuePassing(y)　// 对象y被作为参数传递给函数
- Classname　x=y;　//在声明x时，对象y被用来初始化x
- Classname　y = func();　　// 用返回对象来初始化 y

前2种情况，对象y的引用将被传递给复制函数

第3种情况，返回对象的引用将被传递给y的复制函数

特别注意：　Classname　x=y;

不等价于：　Classname　x;

x=y;　// =是赋值运算，不是初始化，浅拷贝

3.8 对象作为参数和函数返回

从函数返回对象

函数的返回类型声明为一个类类型；

return 该类的一个对象。

```
ARRAY  ReturnArray()
{
        ARRAY  temp(10,10);
        temp.SetValue();
        return temp;
}
ARRAY  a2;
a2 = ReturnArray();
```

问答：

- 在函数运行结束时，**temp**会不会被析构？
- 如何实现返回对象？
- 与值参为对象有何对应关系？

与 **ARRAY a2 = ReturnArray();** 的差别

3.8 对象作为参数和函数返回


```
ARRAY  ReturnArray()
{   ……   }
ARRAY a2 = ReturnArray();    // 正常


ARRAY  a3;
a3 = ReturnArray();      // 析构  a3时异常
                         // 赋值 =不会调用对象赋值构造函数
```

3.8 对象作为参数和函数返回


```
ARRAY &operator=(const ARRAY &ca)
{    delete [] a;
     r = ca.r;
     c = ca.c;
     a = new int[r*c];
     memcpy(a,ca.a,sizeof(int)*r*c);
     return *this;
}
ARRAY   a1;  ……  ARRAY   a2;  ……
a2=a1;                  // 将=运算符重载，看成一个函数 f
                        //  a2.f(a1);
                        //等同于执行传统的函数  f(&a2，&a1)
a2 = ReturnArray( );
```

3.8 对象作为参数和函数返回


```
ARRAY &operator=(const ARRAY &ca)
{
     ……..
    return *this;
}
ARRAY   a1;  ……  ARRAY   a2;  ……
a2=a1;                a2 = ReturnArray( );
```

将=运算符重载，看成一个函数 f

等同于执行传统的函数 f(&a2，&a1)

a2的地址就对应 this 指针；a1的地址就对应参数 ca

该函数并不需要返回结果，

为什么要将函数定义为 ARRAY & ……？

3.8 对象作为参数和函数返回


```
ARRAY &operator=(const ARRAY &ca)
{
    ……..
   return *this;
}
```

- void operator=(const ARRAY &ca);
- ARRAY operator=(const ARRAY &ca);
- ARRAY &operator=(const ARRAY &ca);

均可以实现 a2=a1;

但是，对于 a3 = a2 = a1; 等同 f(a3,f(a2,a1));

若 f 无返回值，上述语句则不能支持

3.8 对象作为参数和函数返回


```
ARRAY  &operator=(const ARRAY &ca)
{
     ……..
    return *this;
}
```

对于 a3 = a2 = a1; 等同 f(a3,f(a2,a1));

对于 (a3 = a2) = a1; f(a3,a2) = a1;

```
ARRAY  const & operator=(const ARRAY &ca)
{     ……..
    return *this;
}
```

在函数返回用时，增加const 修饰，结果如何？

3.8 对象作为参数和函数返回

- 设计思考
- 如何实现 2 个相同大小的矩阵加法，返回一个新矩阵？

```
ARRAY   a1(5,7);
ARRAY   a2(5,7);
ARRAY   ARAAY_ADD(a1, a2) ;    // 不是一个类的成员函数
A1.ARRAY_ADD(a2);

ARRAY  ARRAY_ADD(ARRAY  a2) { ……. }
```

3.8 对象作为参数和函数返回


```
ARRAY(const ARRAY & ca)  { … }     // 对象复制构造函数
ARRAY  &operator=(const ARRAY &ca) // 对象赋值函数

ARRAY & ARRAY_ADD(const ARRAY  &a2)
{ ……. }
```

3.9 内联/位段/局部类/类的存储空间

内联：函数前加 **inline**

位段：有几个二进制位来表示某种信息

```
class  SWITCH{
public:
   int   power:3;
   int   water:5;
   int   gas:4;
}
```

```
SWITCH temp;
temp.power = 6;
temp.water = 15;
temp.gas = 8;
```

7E

有何优点？

VS 三个独立变量 VS 一个变量

3.9 内联/位段/局部类/类的存储空间

局部类：在一个函数中定义的类

类的存储空间

#pragma pack(n)

指明各数据成员的地址对齐方式

#pragma pack(1) // 紧凑方式

默认是松散方式，

一个 **int** 类型的变量其地址被 **4**整除。

补充说明

在 .c 文件中 与 .cpp 中 struct 用法上的差别

- 在.cpp 文件中，struct 用法 与class 相同

  可以有构造函数、析构函数、权限说明等；

  差别：class 默认的访问权限是 private;

  struct 默认的访问权限是 public;

- 在 .c 文件中

  struct 中不能有函数；不能有权限说明；

  在定义结构变量时，要写成 struct *** x 之类的形式；

  当然可以用typedef , 将 struct *** 赋予新的名字。

总结

- 类的声明及定义
- 数据成员、函数成员的声明和定义
- 访问权限 private, protected, public
- 构造函数与析构函数
- new和delete
- 组合类和对象成员的初始化
- 常对象、常成员函数
- 隐含参数this
- 对象作为参数和函数返回
- 复制构造函数

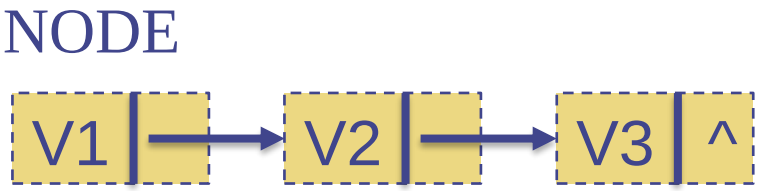
类的设计思考

栈（整型数的栈）的设计：

要求：用链表来存放各元素；

进栈、出栈、构造、析构等等操作

```
class  STACK {
        ????          *head;
public:
        STACK() { head = 0; }
        ~STACK();
        int push(int v);
        int pop(int& v);
};
```



类的设计思考


```
class NODE {
public:          // 在STACK中，要访问各成员
        int  val;
        NODE* next;
        NODE(int v);   // 在push 时，要生成一个新节点
};

 class  STACK {
        NODE        *head;
 public:
        STACK() { head = 0; }
        ~STACK();
        int push(int v);
        int pop(int& v);
 };
```

NODE

问题：NODE中的
信息未能隐藏

类的设计思考

華中科技大學

```
class  STACK {
      class NODE {
      public:
            int  val;
            NODE* next;
            NODE(int v);
      };                        // 可写成 class NODE {…..} *head;
      NODE     *head;
public:
            STACK() { head = 0; }
            ~STACK();
            int push(int v);
            int pop(int& v);
};
```

类的设计思考

实现信息隐藏

在非STACK函数中，不能直接使用 NODE

```
void f () { NODE  p(10)    // NODE 未申明的标识符
            STACK::NODE p(10);   // 无法访问私有类
           // 若有 public:  class NODE …… 则可访问
```

```
class  STACK {
     class NODE {
     public:
          int  val;    NODE* next;        NODE(int v);
     };
    NODE     *head;
public:
          STACK() { head = 0; }      ~STACK();
          int push(int v); int pop(int& v);  };
```

练习

试设计一个队列类 Queue, 并测试各项功能。
队列是一个先进先出(FIFO：First In First out) 的数据结构。
队列元素（整数）的存储：
　　用一个整数型的数组；
　　数组环形使用
对一个队列常用的操作有：
　　在队列尾增加一个元素
　　在队列头取一个元素
　　判断队列是否为空
　　判断队列是否已满
　　依次显示队列所有元素等