2020春季学期C++整理4.0

一.利用构造函数对类对象进行初始化

1.对象的初始化：

我们知道在程序中我们通常要对变量赋初值——即初始化。对于一般的变量，我们可以这样来初始化：

`int x = 0;double y = 0.0`

对于一个类，我们也需要对他进行初始化工作，但是可以发现的是，如果我们按照以下方式初始化：

```
class Time
{
    public :
        int t = 0;
        int m = 0;
        int s = 0;
};
```

显然这是**不对的！！**

也就是说，我们**不能在类的声明中对对象进行初始化操作。**

但是类比结构体，我们知道，对于一个成员变量都是公用`public`类型的类时，我们可以这样做：

```
class Time
{
    public:
        int t;
        int m;
        int s;
};
Time ti = {14,56,30};
```

这和结构体的处理是相似的。

但是：如果成员是`private`或者`protected`型时，就不能这样初始化了。所以到底该咋办呢？？

2.利用构造函数实现数据成员初始化

为了解决上述问题，C++提供了构造函数来解决这一问题。**构造函数是一种特殊的成元函数**，与其他函数不同，**不需要用户来调用它，用户也不能调用它，而是在建立对象是自动执行。**构造函数的名字必须和类名称相同，**不能任意命名**！它**不具有任何类型，也没有返回值。**

注意：

- 如果定义了构造函数，则编译器不生成默认的无参数的构造函数。
- **对象生成时构造函数自动被调用。对象一旦生成，就再也不能在 其上执行构造函数。**
- 一个类可以有多个构造函数~~麻烦要死~~

给个例子：

```
class Time
{
    public:
        int t;
        int m;
        int s;
    public:
    Time()
    {
        t = 0;
        m = 0;
        s = 0;
    }
};
```

上述操作后，`t,m,s`的初始值都为0。

3.带参数的构造函数

如上所示，如果我们希望对不同的对象赋予不同的初值，那采用上面的方法就不合适了。我们给出带参数的构造函数，这样在调用不同对象的的构造函数时，就可以从外面将不同的数据传递给构造函数，实现不同的初始化。

一般结构为：**构造函数名（类型1 形参1，类型2 形参2）**

前面我们讲过，用户自己不能调用构造函数。所以说没有办法像之前对待一般函数一样传递实参（比如`fun(a,b)`）实参是在定义对象时给出的。

```
class Time
{
    public:
        time(int,int,int) //声明带参数的构造函数
        int t;
        int m;
        int s;
};
Time :: time(int x,int y,int z) //在类外定义带参数的构造函数
{
    t = x;
    m = y;
    s = z;
}
int main()
{
    Time t1(12,34,23); //建立对象t1并初始化相关变量的值
}
```

4.使用默认参数的构造函数

构造函数中的参数的值既可以通过实参传递，也可以指定为某些默认值，即如果用户不指定实参值，编译系统就会使形参的值为默认值。

比如下面的代码：

```
class Time
{
```

```
    public:
        time(int x = 10,int y = 12,int z = 20) //声明构造函数时指定默认参数
        int t;
        int m;
        int s;
};
Time :: time(int x,int y,int z) //在类外定义带参数的构造函数
{
    t = x;
    m = y;
    s = z;
}
int main()
{
    Time t1(12,34,23); //建立对象t1并初始化相关变量的值
    Time t2();//无实参
    Time t2(12,34);//给定两个实参
    Time t4(13); //给定一个实参
}
```

注意：第四行的代码也可以简写成 `time(int = 10,int = 10,int = 10)` 形参我们可以忽略

5.样例说明

```
class Complex {
    private :
    double real, imag;
    public:
    Complex( double r, double i = 0); };
    Complex::Complex( double r, double i) { real = r; imag = i;
}
Complex c1; // error, 缺少构造函数的参数
Complex * pc = new Complex; // error, 没有参数
Complex c1(2); // OK
Complex c1(2,4), c2(3,5);
Complex * pc = new Complex(3,4);
```

同时上面提到过，一个类中可以存在多个构造函数：参数类型和个数不同（**这里需要注意一下，对于默认参数的构造函数一个类中只允许存在一个！为了避免歧义**）

```
class Complex
{
    private :
    double real, imag;
    public:
    void Set( double r, double i );
    Complex(double r, double i );
    Complex (double r );
    Complex (Complex c1, Complex c2);
};
Complex::Complex(double r, double i)
{
    real = r; imag = i;
}
Complex::Complex(double r)
{
```

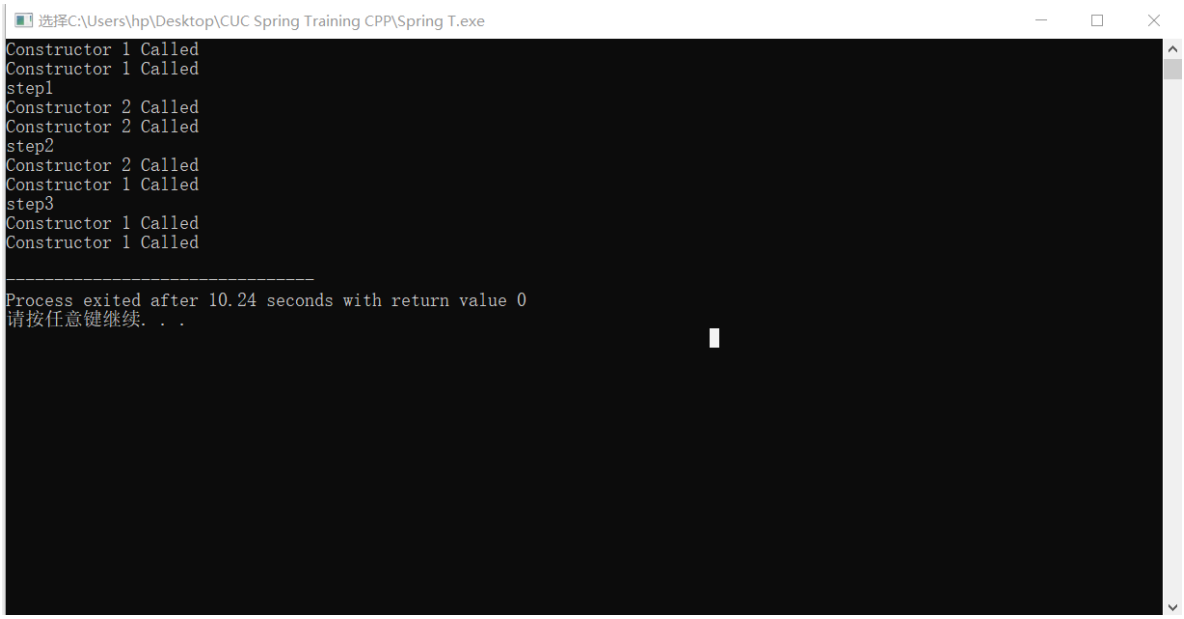
```
    real = r;
    imag = 0;
}
Complex::Complex (Complex c1, Complex c2);
{
    real = c1.real+c2.real;
    imag = c1.imag+c2.imag;
}
Complex c1(3),c2 (1,0),c3(c1,c2);
// c1 = {3, 0}, c2 = {1, 0}, c3 = {4, 0};
```

二.构造函数在数组中的应用

用例子来解释:

```
class CSample
{
    int x;
    public:
    CSample() //函数1
    {
        cout << "Constructor 1 Called" << endl;
    }
    CSample(int n) //函数2
    {
        x = n;
        cout << "Constructor 2 Called" << endl;
    }
};
int main()
{
    CSample array1[2]; //调用两次函数1
    cout << "step1"<<endl;
    CSample array2[2] = {4,5}; //调用两次函数2
    cout << "step2"<<endl;
    CSample array3[2] = {3}; //先调用函数2，再调用函数1
    cout << "step3"<<endl;
    CSample * array4 = new CSample[2]; //直接两次函数1
    delete []array4; return 0;
}
```

输出结果如下:




```
class Test
{
    public: Test( int n) { } //(1)
    Test( int n, int m) { } //(2)
    Test() { } //(3)
};
Test array1[3] = { 1, Test(1,2) }; // 三个元素分别用(1),(2),(3)初始化
Test array2[3] = { Test(2,3), Test(1,2) , 1}; // 三个元素分别用(2),(2),(1)初始化
Test * pArray[3] = { new Test(4), new Test(1,2) };//两个元素分别用(1),(2) 初始化
```

三.复制构造函数

我们有时候需要用到多个完全相同的对象，如果对这些对象进行分别处理，就需要建立多个相同的对象，并进行相同的初始化，太麻烦了！！！

C++为我们提供了对象的复制机制，**用一个已有的对象复制出多个完全相同的对象**

`Box box2(box1)`

其作用是去用box1去初始化box2

一般类型表示：**类名 对象2 （对象1）**既是用对象1复制出对象2.

我们可以看出，其括号中的参数是一个对象而不是变量，所以在建立对象的时候调用一个特殊的构造函数——**复制构造函数**

形式如下：

```
Box::Box(const Box& b)
{
    height = b.height;
    width = b.width;
    len = b.len;
}
```

复制函数也是一个构造函数，但他只有一个参数，这个参数是本类的对象，而且采用对象的引用的形式（一般约定加`const`来声明，使得参数值不能改变）。此复制函数的作用是将实参对象的各成员值一一赋值给新的对象中对应的成员。

我们对`Box box2(box1)`分析：

实参box1的地址传递给形参b（b就成为box1的引用）在执行复制构造函数的函数体时，将box1对象中的各数据成员值赋给box2中的各个数据成员。

同时，C++也给出了这样的复制形式：

`Box box2 = box1`

一般形式为：类名 对象1 = 对象2;

也可以在一个语句中对多个对象操作：

`Box box2 = box1;box3 = box3`

附加说明 对普通构造函数和复制构造函数的区分：

- 在形式上：

  类型（形参） //普通构造函数的声明

  类型（类名 & 对象名） //复制构造函数的声明

- 在建立对象时实参类型不同

  `Box box1(12,13,14)` //实参为整形，调用普通构造函数

  `Box box2(Box1)` //实参是对象名，调用复制构造函数

- 在什么情况下被调用

  普通构造函数在成程序中建立对象时被调用

  复制函数在用已有对象复制一个新对象的时候被调用。

  - 程序中需要建立一个新的对象，并用另一个同类对象对他进行初始化。
  - 函数的参数为类的对象
  - 函数的返回值是类的对象

```
Box f()
{
    Box box1(12,13,14);
    return box1;
}
int main()
{
    Box box2;
    box2 = f();
    return 0;
}
```

四.析构函数

析构函数也是一个特殊的成员函数，名字与类名相同，在前面加'~'， 没有参数和返回值，一 个类最多只能有一个析构函数。（~~定义上和复制构造函数差不多~~）**析构函数对象消亡时即自动被调用。**可以定义析构函数来在对象消亡前做善后工作，比如释放分配的空间等。如果定义类时没写析构函数，则编译器**生成缺省析构函数。** 缺省析构函数什么也不做。**如果定义了析构函数，则编译器不生成缺省析构函数**

```
class String
{   private :
    char * p;
```

```
    public:
    String ()
    {
        p = new char[10];
    }
    ~ String () ;
};
String ::~ String()
{
    delete [] p;
}
```

对于析构函数和数组：

```
class Ctest
{
    public:
    ~Ctest(){ cout<< "destructor called" << endl; }
};
int main ()
{
    Ctest array[2];
    cout << "End Main" << endl;
    return 0;
}
输出:End Main
    destructor called
    destructor called
```

注意：若new一个对象数组，那么用delete释放时应该写 []。否则只delete一个对 象(调用一次析构函数

```
Ctest * pTest;
pTest = new Ctest; //构造函数调用
delete pTest; //析构函数
```

```
pTest = new Ctest[3]; //构造函数调用3次
delete [] pTest; //析构函数调用3次
```

对于析构函数和构造函数调用的顺序，个人感觉更像是个栈，先构造的后析构，后构造的先析构，就像是栈的后进先出。