类

类的内容是C++中的一个重点内容，同时也是901考试的重点内容。每年都会有两道（至少）读程序题会涉及到类，这两道题也一般是与其他人拉开差距的关键。因此需要注意。

类的声明

```
class 类名称 {
public:
    公有成员（外部接口）
private:
    私有成员（只允许本类中的函数访问，而类外部的任何函数都不能访问）
protected:
    保护成员（与private类似，差别表现在继承与派生时）
};
```

每个类可以没有成员，也可以定义多个成员，成员可以是数据、函数、或类型别名。

类成员有三种形式`public`、`private`、`protected`。

所有成员必须在类内部声明，一旦类定义完成后，就没有任何方式可以增加成员了。

构造函数：创建一个类类型的对象时，编译器会自动使用一个构造函数来初始化对象。构造函数是一个特殊的与类同名的成员函数，用于给每个数据成员设置适当的初始值。

构造函数的名称与类的名称是完全相同的，并且不会返回任何类型，也不会返回 void。构造函数可用于为某些成员变量设置初始值。

构造函数格式：

```
class Line {
public:
    // 无参数构造函数
    Line() {}
    // 带参数构造函数
    Line(double len) {}
    // 使用初始化列表来初始化字段
    Line(double wid, double len): width(wid), length(len) {}
    ...
}
```

析构函数是类的一种特殊的成员函数，它会在每次删除所创建的对象时执行。

析构函数的名称与类的名称是完全相同的，只是在前面加了个波浪号（~）作为前缀，它不会返回任何值，也不能带有任何参数。析构函数有助于在跳出程序（比如关闭文件、释放内存等）前释放资源。

析构函数格式：

```
class Line {
public:
    // 构造函数
    Line() {}
    // 析构函数
    ~Line() {}
    ...
}
```

类成员函数

我们定义一个类`Box`

```
class Box {
public:
    double length;
    double height;
    double breadth;
    double getVolume();  // 返回体积
}
```

其中我们定义了一个计算体积的函数`getVolume()`，那我们应该如何实现它呢，有两种方式：可以定义在类定义内部，或者单独使用**范围解析运算符 ::** 来定义。

```
class Box {
public:
    double length;
    double height;
    double breadth;
    double getVolume() {  // 返回体积
        return length * height * breadth;
    }
}
```

或者：

```
class Box {
public:
    double length;
    double height;
    double breadth;
    double getVolume();  // 返回体积
}

double Box::getVolume() {
    return length * height * breadth;
}
```

类访问修饰符

数据封装是面向对象编程的一个重要特点，它防止函数直接访问类类型的内部成员。类成员的访问限制是通过在类主体内部对各个区域标记 **public、private、protected** 来指定的。关键字 **public、private、protected** 称为访问修饰符。

一个类可以有多个 `public`、`protected` 或 `private` 标记区域。每个标记区域在下一个标记区域开始之前或者在遇到类主体结束右括号之前都是有效的。成员和类的默认访问修饰符是 `private`。

公有成员（public）

公有成员在程序中类的外部是可访问的。

私有成员（private）

私有成员变量或函数在类的外部是不可访问的，甚至是不可查看的。只有类和友元函数可以访问私有成员。默认情况下，类的所有成员都是私有的。

保护成员（protected）

保护成员变量或函数与私有成员十分相似，但有一点不同，保护成员在派生类（即子类）中是可访问的。

继承中的特点：

有public, protected, private三种继承方式，它们相应地改变了基类成员的访问属性。

- 1.**public 继承：**基类 public 成员，protected 成员，private 成员的访问属性在派生类中分别变成：public, protected, private
- 2.**protected 继承：**基类 public 成员，protected 成员，private 成员的访问属性在派生类中分别变成：protected, protected, private
- 3.**private 继承：**基类 public 成员，protected 成员，private 成员的访问属性在派生类中分别变成：private, private, private

但无论哪种继承方式，上面两点都没有改变：

- 1. `private` 成员只能被本类成员（类内）和友元访问，不能被派生类访问；
- 2. `protected` 成员可以被派生类访问。

拷贝构造函数

拷贝构造函数是一种特殊的构造函数，它在创建对象时，是使用同一类中之前创建的对象来初始化新创建的对象。拷贝构造函数通常用于：

- 通过使用另一个同类型的对象来初始化新创建的对象。
- 复制对象把它作为参数传递给函数。
- 复制对象，并从函数返回这个对象。

如果在类中没有定义拷贝构造函数，编译器会自行定义一个。

```
class Line() {
public:
    Line() {}
    // 拷贝构造函数
    Line(Line &obj) {}
    ~Line() {}
    ...
}
```

类的静态成员

我们可以使用 **static** 关键字来把类成员定义为静态的。当我们声明类的成员为静态时，这意味着无论创建多少个类的对象，静态成员都只有一个副本。

静态成员在类的所有对象中是共享的。如果不存在其他的初始化语句，在创建第一个对象时，所有的静态数据都会被初始化为零。我们不能把静态成员的初始化放置在类的定义中，但是可以在类的外部通过使用范围解析运算符 :: 来重新声明静态变量从而对它进行初始化。

例题：

例1：

```
#include<iostream>
using namespace std;
class MyClass
{
    int a,b;
    public :
        MyClass(int x=0,int y=0);
        ~MyClass();
};
MyClass::MyClass(int x,int y):a(x),b(y)
{
    cout<<"Constructor a+b="<<a+b<<endl;
}
MyClass::~MyClass()
{
    cout<<"Destructor"<<a+b<<endl;
}
int main()
{
    MyClass x,y(10,20),z(y);
    return 0;
}
```

结果如下：

```
Constructor a+b=0
Constructor a+b=30
Destructor30
Destructor30
Destructor0
```

例2：

```
#include<iostream>
using namespace std;
class B0
{
    public :
            virtual void display(){cout<<"B0::display()."<<endl;}
            B0(){cout<<"B0 called.\n";}
            ~B0(){cout<<"~B0 called."<<endl;}
};
class B1:public B0
{
    public :
            void display(){cout<<"B1::display()."<<endl;}
            B1(){cout<<"B1 called.\n";}
            ~B1(){cout<<"~B1 called.\n";}
};
class D1:public B1
{
    public :
            void display(){cout<<"D1::display().\n";}
            D1(){cout<<"D1 called.\n";}
            ~D1(){cout<<"~D1 called.\n";}
};
void fun(B0 *ptr)
{
    ptr->display();
}
int main()
{
    B0 b0,*p;
    D1 d1;
    p=&b0;
    fun(p);
    p=&d1;
    fun(p);
    return 0;
}
```

结果如下：

```
B0 called.
B0 called.
B1 called.
D1 called.
B0::display().
D1::display().
~D1 called.
~B1 called.
~B0 called.
~B0 called.
```

例3：

```
#include<iostream>
using namespace std;
class MyClass
{
    int x;
    public :
        MyClass(int a)
        {
            x=a;
            cout<<"Constructor"<<x<<endl;
        }
        ~MyClass()
        {
            cout<<"Destructor"<<x<<endl;
        }
};
int main()
{
    MyClass Commonobj(0);
    static MyClass Staticobj(1);
    return 0 ;
}
MyClass GlobalObj(9);
```

结果如下：

```
Constructor9
Constructor0
Constructor1
Destructor0
Destructor1
Destructor9
```

例4：

```
#include<iostream>
using namespace std;
```

```
class A
{
    public :
        A(){cout<<"A's cons."<<endl;}
        virtual ~A(){cout<<"A's des."<<endl;}
        virtual void f(){cout<<"A's f()."<<endl;}
        void g(){f();}
};
class B:public A
{
    public :
        B(){f();cout<<"B's cons."<<endl;}
        ~B(){cout<<"B's des."<<endl;}
};
class C:public B
{
    public :
        C(){cout<<"C's cons."<<endl;}
        ~C(){cout<<"C's des."<<endl;}
        void f(){cout<<"C's f()."<<endl;}
};
int main()
{
    A *a=new C;
    a->g();
    delete a;
    return 0;
}
```

结果如下：

```
A's cons.
A's f().
B's cons.
C's cons.
C's f().
C's des.
B's des.
A's des.
```

例5：

```
#include<iostream>
using namespace std;
class A
{
    public :
        A(char *s){cout<<s<<endl;}
};
class B:virtual public A
{
    public :
```

```
        B(char *s1,char *s2):A(s1){cout<<s2<<endl;}
};
class C:virtual public A
{
    public :
        C(char *s1,char *s2):A(s1){cout<<s2<<endl;}
};
class D:public B,public C
{
    public :
        D(char *s1,char *s2,char *s3,char *s4):B(s1,s2),C(s1,s3),A(s1)
        {
            cout<<s4<<endl;
        }
};
int main()
{
    D *p=new D("Class A","Class B","Class C","Class D");
    return 0;
}
```

结果如下:

```
Class A
Class B
Class C
Class D
```

例6:

```
#include<iostream>
using namespace std;
class Base
{
    int Y;
    public :
        Base(int y=0){Y=y;cout<<"Base("<<Y<<")\n";}
        ~Base(){cout<<"~Base()\n";}
        void print(){cout<<Y<<"";}
};
class Derived:public Base
{
    int Z;
    public :
        Derived(int y,int z):Base(y)
        {
            Z=z;
            cout<<"Derived("<<y<<","<<z<<")\n";
        }
        ~Derived(){cout<<"~Derived()\n";}
        void print(){Base::print();cout<<Z<<endl;}
};
```

```
int main()
{
    Derived d(10,20);
    d.print();
    return 0;
}
```

结果如下：

```
Base(10)
Derived(10,20)
1020
~Derived()
~Base()
```

例7：

```
#include<iostream>
using namespace std;
class Base
{
    int i;
    public :
        Base(int n){i=n;cout<<"Constructing base class"<<i<<endl;}
        ~Base(){cout<<"Destructing base class"<<i<<endl;}
        void show(){cout<<i<<",";}
        int Geti(){return i;}
};
class Derived:public Base
{
    int j;
    Base aa,bb;
    public :
        Derived(int n,int m,int p):Base(m),bb(0),aa(p)
        {
            j=n;
            cout<<"Cnstructing derived class"<<j<<endl;
        }
        ~Derived(){cout<<"Destructing derived class"<<j<<endl;}
        void show()
        {
            Base::show();
            cout<<j<<","<<aa.Geti()<<endl;
        }
};
int main()
{
    Derived obj(8,13,24);
    obj.show();
    return 0;
}
```

结果如下：

```
Constructing base class13
Constructing base class24
Constructing base class0
Cnstructing derived class8
13,8,24
Destructing derived class8
Destructing base class0
Destructing base class24
Destructing base class13
```

例8：

```
#include<iostream>
using namespace std;
class A
{
    public :
        A(){cout<<"A 构造函数\n";fun();}
        virtual void fun(){cout<<"A::fun()函数\n";}
};
class B:public A
{
    public :
        B(){cout<<"B 构造函数\n";fun();}
        void fun(){cout<<"B::fun()函数\n";}
};
int main()
{
    B b;
    return 0;
}
```

结果如下：

```
A 构造函数
A::fun()函数
B 构造函数
B::fun()函数
```

例9：

```
#include<iostream>
using namespace std;
class Base
{
    int x;
    public :
        Base(int i){x=i;cout<<"Constructor of Base"<<x<<endl;}
        ~Base(){cout<<"Destructor of Base"<<x<<endl;}
```

```
        void show(){cout<<"x="<<x<<endl;}
};
class Derived:public Base
{
    Base d;
    public :
        Derived(int i):Base(i),d(9)
        {
            cout<<"Constructor of Derived"<<endl;
        }
        ~Derived(){cout<<"Destructor of Derived"<<endl;}
};
int main()
{
    Derived obj(5);
    obj.show();
    return 0;
}
```

结果如下：

```
Constructor of Base5
Constructor of Base9
Constructor of Derived
x=5
Destructor of Derived
Destructor of Base9
Destructor of Base5
```

例10：

```
#include<iostream>
using namespace std;
class AA
{
    int aa;
    public :
        static int count;
        AA(int a=0):aa(a)
        {
            count++;
            cout<<"Constructor A:"<<aa<<endl;
        }
        AA(AA &pa)
        {
            count++;
            aa=pa.aa;
            cout<<"Copy constructor A:"<<aa<<endl;
        }
        ~AA()
        {
            count--;
```

```
            cout<<"Destructor A:"<<aa<<endl;
        }
};
int AA::count;
int main()
{
    cout<<AA::count<<endl;
    AA a,b(3),c(2),*d;
    cout<<a.count<<endl;
    d=new AA(b);
    cout<<d->count<<endl;
    delete d;
    cout<<a.count<<endl;
}
```

结果如下：

```
0
Constructor A:0
Constructor A:3
Constructor A:2
3
Copy constructor A:3
4
Destructor A:3
3
Destructor A:2
Destructor A:3
Destructor A:0
```

例11：

```
#include<iostream>
using namespace std;
class A
{
    protected:
        int x,y;
    public :
        A(){x=3;y=4;cout<<"A() Constructor!"<<endl;}
        A(int m,int n)
        {
            x=m;y=n;
            cout<<"A(m,n) Constructor!"<<endl;
        }
        ~A(){cout<<"A() Destructor!"<<endl;}
};
class B:public A
{
    protected:
        int i,j;
    public :
```

```
        B(int a,int b):A(a,b)
        {
            i=3;j=4;
            cout<<"B(3,4)Constructor!"<<endl;
        }
        B(int a,int b,int m,int n):A(m,n)
        {
            i=a;
            j=b;
            cout<<"B(a,b,m,n)Constructor!"<<endl;
        }
        ~B(){cout<<"B()Destructor!"<<endl;}
        void print(){cout<<x<<y<<i<<j<<endl;}
};
int main()
{
    B d(1,2),e(5,6,7,8);
    d.print();
    e.print();
    return 0;
}
```

结果如下：

```
A(m,n) Constructor!
B(3,4)Constructor!
A(m,n) Constructor!
B(a,b,m,n)Constructor!
1234
7856
B()Destructor!
A() Destructor!
B()Destructor!
A() Destructor!
```

例12：

```
#include<iostream>
#include<cstring>
using namespace std;
class St
{
    char str[100];
    public :
        St(char *s)
        {
            strcpy(str,s);
            cout<<"Cons:"<<str<<endl;
        }
        ~St(){cout<<str<<":Des"<<endl;}
};
St e("str5");
```

```
static St f("str6");
int main()
{
    St a("str1");
    static St b("str2");
    {
        St c("str3");
    }
    St d("str4");
    return 0;
}
```

结果如下：

```
Cons:str5
Cons:str6
Cons:str1
Cons:str2
Cons:str3
str3:Des
Cons:str4
str4:Des
str1:Des
str2:Des
str6:Des
str5:Des
```

例13：

```
#include<iostream>
using namespace std;
class A
{
    protected:
        int x,y;
    public :
        A(int m=5,int n=6){x=m;y=n;}
};
class B:public A
{
    protected:
        int i,j;
    public :
        B(int a,int b){i=a;j=b;}
        B(int a,int b,int m,int n):A(m,n){i=a;j=b;}
        void print(){cout<<x<<y<<i<<j<<endl;}
};
int main()
{
    B d(9,2);
    d.print();
    B e(7,8,3,4);
```

```
    e.print();
    return 0;
}
```

结果如下：

```
5692
3478
```

例14：

```
#include<iostream>
using namespace std;
class A
{
    int *a;
    public :
        A(int aa=0)
        {
            a=new int(aa);
            cout<<"Constructor"<<*a<<endl;
        }
        ~A()
        {
            if(*a=2)
                cout<<"A's Destructor"<<endl;
            else
                cout<<"Destructor"<<endl;
        }
};
int main()
{
    A x(2);
    A *p=new A(5);
    delete p;
}
```

结果如下：

```
Constructor2
Constructor5
A's Destructor
A's Destructor
```

例15：

```
#include<iostream>
using namespace std;
class base
{
    public :
```

```
        void who(){cout<<"base class"<<endl;}
};
class derived1:public base
{
    public :
        void who(){cout<<"derived1 class"<<endl;}
};
class derived2:public base
{
    public :
        void who(){cout<<"derived2 class"<<endl;}
};
int main()
{
    base obj1;
    obj1.who();
    derived1 obj2;
    obj2.base::who();
    derived2 obj3;
    obj3.base::who();
    return 0;
}
```

结果如下：

```
base class
base class
base class
```

例16：

```
#include<iostream>
using namespace std;
class A
{
    int *a;
    public :
        A(int x)
        {
            a=new int (x);
            cout<<"Constructor"<<*a<<endl;
        }
        ~A()
        {
            if(*a==3)
                cout<<"Destructor"<<endl;
            else
                cout<<"Destructor"<<*a<<endl;
            delete a;
        }
};
int main()
```

```
{
    A x(3),*p;
    p=new A(5);
    delete p;
}
```

结果如下：

```
Constructor3
Constructor5
Destructor5
Destructor
```