第五章 类和对象

面向对象程序设计

基本特征

1.抽象性
2.封装性
3.继承性
4.多态性

类的定义

类的定义格式

```
class 类名
{
public:
        公有数据成员和成员函数的定义和说明
private:
        私有数据成员和成员函数的定义和说明(可省略)
protected:
        保护数据成员和成员函数的定义和说明
};
```

注意事项：若在类体外定义成员函数，则需要在类体内给出它们的原型说明，在类体外的函数名面前加上类名和域作用符“::”，以表示所属的类

定义类时的注意事项

1.不允许对类中数据成员进行初始化，仅允许用static修饰
2.不允许本类的变量和数组作为本类的成员，仅允许本类的指针和引用作为本类的成员
3.一个已定义类的底线可以作为另一个类的数据成员

对象的定义和引用

对象的定义

1.在定义类的同时定义对象

```
class Date{
public:
```

```
        void SetDate(int y=2006,int m=1,int d=1) //仅对月份进行正确性检查
        {year=y;month=m>0 && m<13?m:1;day=d;}
        int IsLeapYear()
        {return (year%400==0) ||(year%4==0 && year %100!=0);}
        int GetMonth(){return month;}
        int GetDay(){return day;}
        void Print(){cout<<year<<'.'<<month<<'.'<<day<<endl;}
private:
        int year,month,day;

}d1,d2;
```

上面的d1和d2是Date的对象

2.先定义类，再定义对象（可以是对象数组，对象的引用红孩儿指向对象的指针）

class 类名 对象名表

PS:在类外定义成员函数的方式为
**返回类型 类名 函数名()

对象成员的引用

直接引用，格式如下

```
Student s,*sp=&s;&rs=s; //定义对象，Student是类名

//则引用如下
s.Print()      //使用对象引用共有成员函数Print
sp→Print()     //使用指针引用...
rs.Print()     //使用引用变量引用...
```

对象的初始化

构造函数

1.构造函数的作用

主要用于为对象分配存储空间，对对象的**数据成员**进行初始化

2.构造函数的特点

可以重载

2.1 构造函数无返回值
2.2 构造函数名与本类类名相同
2.3 构造函数的参数个数没有限制，允许重载
2,4 构造函数可以带有一个数据成员初始化表，必须咋形式参数表的圆括号之后，函数体花括号之前，且用一个冒号开始

3.构造函数的一般形式

构造函数名([形式参数表]){:数据成员初始化表}函数体

例如，日期类的构造函数可写成：

```
Date(int y = 2000,int m = 1,int d = 1):year(y),month(m),day(d){}
//或者
Date(int y = 2000,int m = 1,int d = 1){year=y;month=m;day=d}
```

4.构造函数的注意事项

4.1 如果用户没有定义构造函数，系统会自动生成无参的、函数体为空的构造函数，仅为对象分配存储空间，不能起到为对象的数据成员进行初始化的作用
4.2 参数个数为0或者全体形参都带有缺省值的构造函数都是缺省的构造函数

**自动调用的无法初始化！！

5.自动调用构造函数的情况（重点）

5.1 遇到对象定义时自动调用构造函数
5.2 遇到表达式new类名或者new类名[长度]时自动调用缺省构造函数，遇到表达式new类名(实参表)时自动调用构造函数
5.3 遇到表达式 类名(实参表)时自动调用构造函数

析构函数

1.析构函数的作用

主要用于执行该函数中各语句和释放对象所占用的内存空间

2.析构函数的特点

不能重载！！
2.1 析构函数名为本类的类名前面加一个“~”符号

2.2 析构函数不允许带有任何参数，在一个类中析构函数只有 **一个** 且不可以重载
2.3 析构函数无返回值

3.析构函数的注意事项

3.1 如果用户没有定义析构函数，系统会自动生成一个函数体为空的析构函数，可以释放除了用new运算分配的任何对象的存储空间，一般不需要显示定义
3.2 当类中包含由new运算分配存储空间的活动时，必须给出析构函数的明确定义，并用delete运算释放这些存储空间

4.自动调用析构函数的情况

3.1 遇到复合语句执行结束、函数执行结束或程序执行结束时调用析构函数，释放复合语句、函数或整个程序中的局部对象或全局对象占用的内存空间
3.2 在执行delete指向对象的指针或delete[]指向对象数组的指针时调用析构函数，释放用new运算是动态分配的对象占用的存储空间

构造函数和析构函数的调用顺序

1.通常相反

例子如下

```
class Point{                    //点Point类的定义
        double x,y;
public:
        Point(double a=0,double b=0)                        //构造函数
        {x=a;y=b;cout<<"构造点("<<x<<","<<y<<")\n";}
        void Print() {cout<<"点("<<x<<","<<y<<")";}         //显示点信息函数
        double Distance(Point &pr)                          //计算两点间的距离
函数
        {   double dx=x-pr.x,dy=y-pr.y;
            return sqrt(dx*dx+dy*dy);
        }
        ~Point(){cout<<"析构点("<<x<<","<<y<<")\n";}
};

void main()
{
        Point ob1,ob2(-3,4);
        ob1.Print();cout<<"到";ob2.Print();cout<<"的距离=";
        cout<<ob1.Distance(ob2);
        cout<<endl;
        }
```

输出结果为:
构造点(0,0)
构造点(-3,4)
点(0,0)到点(-3,4)的距离=5
析构点(-3,4)
析构点(0,0)

2.对象的生存期会改变析构函数的调用顺序

见课本P133-135

拷贝构造函数

1.拷贝构造函数的作用

> 主要是用一个已定义的对象初始化一个被创建的同类对象（为了防止指针悬挂）

2.拷贝构造函数的特点

> 2.1 函数名与类名相同，无返回值
> 2.2 只有一个参数，必须是一个本类对象的引用

3.自动调用拷贝构造函数函数的情况

> 3.1 遇到一个已定义的对象初始化一个新的同类对象时
> 3.2 在调用一个函数式，要把实参对象按值传递给相应形参对象需要调用
> 3.3 在遇到把对象作为函数返回值时，调用

一维整数数组类必须使用拷贝构造函数
例子如下:

```
//还是上面的Point类
Point::Point(Point &p)
{
        x=p.x;y=p.y;
        cout<<"拷贝函数被调用"<<endl;
}
```

静态成员

static在C语言中的作用 > static修饰类的成员

友元

（选择题出现，认识为主）

1.定义：
在C++中，我们使用类对数据进行了隐藏和封装，类的数据成员一般都定义为私有成员，成员函数一般都定义为公有的，以此提供类与外界的通讯接口。但是，有时需要定义一些函数，这些函数不是类的一部分，但又需要频繁地访问类的数据成员，这时可以将这些函数定义为该函数的友元函数。
除了友元函数外，还有友元类，两者统称为友元。友元的作用是提高了程序的运行效率（即减少了类型检查和安全性检查等都需要时间开销），但它破坏了类的封装性和隐藏性，使得非成员函数可以访问类的私有成员。

友元类

1.定义：
友元类的所有成员函数都是另一个类的友元函数，都可以访问另一个类中的隐藏信息（包括私有成员和保护成员）
当希望一个类可以存取另一个类的私有成员时，可以将该类声明为另一类的友元类。

2.定义格式：
`friend class 类名`
（其中，`friend`和`class`是关键字，类名必须是已定义的类）

3.注意事项：

- 友元关系不能被继承
- 友元关系是单向的，不具有交换性（得看声明）
- 友元关系不具有传递性（得看声明）
  4.代码示例：

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

class Address; //提前声明Address类

//声明Student类
class Student{
public:
 Student(char *name, int age, float score);
public:
 void show(Address *addr);
private:
 char *m_name;
```

```
 int m_age;
 float m_score;
};

//声明Address类
class Address{
public:
 Address(char *province, char *city, char *district);
public:
 //将Student类声明为Address类的友元类
 friend class Student;
private:
 char *m_province; //省份
 char *m_city; //城市
 char *m_district; //区（市区）
};

//实现Student类
Student::Student(char *name, int age, float score): m_name(name),
m_age(age), m_score(score){ }
void Student::show(Address *addr){
    cout<<m_name<<"的年龄是 "<<m_age<<", 成绩是 "<<m_score<<endl;
    cout<<"家庭住址: "<<addr->m_province<<"省"<<addr->m_city<<"市"<<addr-
>m_district<<"区"<<endl;
}

//实现Address类
Address::Address(char *province, char *city, char *district){
    m_province = province;
    m_city = city;
    m_district = district;
}

int main(){
 Student stu("小明", 16, 95.5f);
 Address addr("陕西", "西安", "雁塔");
    stu.show(&addr);

 Student *pstu = new Student("李磊", 16, 80.5);
 Address *paddr = new Address("河北", "衡水", "桃城");
    pstu -> show(paddr);

 return 0;
}
```

友元函数

1.定义：
在当前类以外定义的、不属于当前类的函数也可以**在类中声明**，但要在前面加 friend 关

键字，这样就构成了友元函数。友元函数可以是不属于任何类的非成员函数，也可以是其他类的成员函数。
友元函数可以访问当前类中的所有成员，包括 public、protected、private 属性的。

2.代码示例：

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

class Student{
public:
 Student(char *name, int age, float score);
public:
 friend void show(Student *pstu); //将show()声明为友元函数
private:
 char *m_name;
 int m_age;
 float m_score;
};

Student::Student(char *name, int age, float score): m_name(name),
m_age(age), m_score(score){ }

//非成员函数
void show(Student *pstu){
    cout<<pstu->m_name<<"的年龄是 "<<pstu->m_age<<", 成绩是 "<<pstu-
>m_score<<endl;
}

int main(){
 Student stu("小明", 15, 90.6);
 show(&stu); //调用友元函数
 Student *pstu = new Student("李磊", 16, 80.5);
 show(pstu); //调用友元函数

 return 0;
}
```

运行结果：
小明的年龄是 16，成绩是 95.5
家庭住址：陕西省西安市雁塔区
李磊的年龄是 16，成绩是 80.5
家庭住址：河北省衡水市桃城区

例题

6.#include<iostream,h>

```
class Base{
    private:
        int a;
    public;
```



```
        Base(){a=10;cout<< "基类缺省构造函数\n";}
        Base(int i){a=i;cout<< "基类缺带参构造函数\n";}
        ~Base(){cout<< "基类析构函数\n";}
        virtual void print(){cout<< "a="<<a<<endl;}
};
class Derived:public Base{
    private:
        int b;
    public
        Derived(){b=20;cout<< "派生类缺省构造函数\n";}
        Derived(int i,int j):Base(i),b(j){cout<< "派生类带参构造函数\n";}
        ~Derived(){cout<< "派生类析构函数\n";}
    void print(){Base::print(); cout "b="<<b<<endl;}
};
void main(){
    Base *pa,b;
    Derived d(1,2);
    pa=&b;
    pa->print();
    pa=&d;
    pa->print();
}
```

}
输出结果：

6.
基类缺省构造函数
基类带参构造函数
派生类带参构造函数
a=10
a=1
b=2
派生类析构函数
基类析构函数
基类析构函数