C++概述

- C++与 C 语言的关系

C++是面向对象程序设计思想，而 C 语言是面向过程程序设计思想，C++对 C 语言做了一些改进，与 C 语言是兼容的，是 C 语言的一个子集。

- 类与对象的关系

类是具有相同属性和行为的一组对象的集合。对象是具体存在的事物，明确定义状态和行为。类是一个抽象的概念，对象是类抽象概念的实物表达，对象是类的实例。类是自定义的数据类型，对象是变量。

- 引用类型的定义

int x,&rx=x; //必须对其进行初始化。也就是指定是哪个变量的别名。

- 函数重载

同名不同义，重载是用来描述同名函数具有相同或者相似功能,但数据类型或者是参数不同的函数管理操作的称呼。要掌握判断重载的依据。

类的定义

类中包括数据成员和成员函数，一般数据成员是私有的，成员函数是公有的，默认是私有的。

- 时间类的定义

```
//P54 例 3-6 时间类的定义
#include <iostream>
using namespace std;
class Time
{
private:
    int hour, minute, second;
public://构造函数一般是共有的，否则无法定义对象
    Time(int h=0,int m=0,int s=0 ) :hour(h), minute(m), second(s) {}
    void show()
    {  cout<<hour<<':'<<minute<<':'<<second<<endl;  }
    };
int main()
{
    Time t1;
    Time t2(1,2,3);
    t1.show();
    t2.show();
    return 0;
}
```

思考：如果将 Time 对象转换成一个整数，应该怎么实现？

- 点类的定义

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

```
{
public:
    Point(float x = 0, float y = 0) :x(x), y(y) {}
    void show()
    {   cout << "(" << x << "," << y << ")" << endl;    }
private:
    float x,y;
};
int main()
{
    Point t1,t2[3],*p; //定义了 4 个对象，调用 4 次构造函数。
    Point *p1 = new Point[10];  //生成 10 个对象，调用 10 次构造函数
    Point *p2 = new Point(5,5); //生成 1 个对象，调用 1 次构造函数
    p2->show();
    delete p2;   //析构 1 次
    delete[]p1;  //析构 10 次，如果写成 delete p1,就只会析构一次
    return 0;
}
```

- P79　3-9 3-10 习题
- 常成员函数

  常成员只能调用常成员函数
- 复制构造函数

```
//P61 例 3-14 阅读下列程序，分析出现的问题
#include<iostream>
using namespace std;
class Student
{public:
    Student(int n,char *na,int s)      //构造函数，name 指向分配的空间
    {    no=n;
         name = new char[strlen(na)+1];
         strcpy(name,na);
         score=s;
    }
    ~Student()                              // 析构函数，释放动态分配的空间
    {    if(name != NULL)
         {   delete []name;       }
    }
private:
    int no;
    char *name;
    int score;
};
int main()
{    Student stu1(1,"wangming",90);
```

```
    Student stu2(stu1);                    // 复制对象，另一种写法是什么？
    return 0;
}
```
/*程序运行会出现运行错误。创建 stu1 对象时构造函数分配空间并赋值给 name,执行 Student stu2(stu1)时，执行 stu2.name=stu1.name，由于没有分配新空间给 stu2,所以两个指针指向了同一个空间。析构时对同一个内存空间会释放二次，这就是错误出现的原因。解决办法就是使用“深复制”。“深复制”时，对于对象中的动态成员不能只是简单的赋值，而应该重新动态分配空间，解决方案如下：*/

```
//P61 例 3-14 深复制”解决例 3-13 的问题
#include<iostream>
using namespace std;
class Student
{public:
    Student(int n,char *na,int s)//构造函数，name 指向堆中分配的一空间
    {   no=n;
        name = new char[strlen(na)+1];
        strcpy(name,na);
        score=s;
    }
    Student(const Student& s)
    {   no = s.no;
        score=s.score ;
        name = new char[strlen(s.name)+1];//为新对象重新动态分配空间
        strcpy(name,s.name );
    }
    ~Student()      // 析构函数，释放动态分配的空间
    {    if(name != NULL)
        {   delete []name;     }
}
private:
    int no;
    char *name;
    int score;
};
int main()
{   Student stu1(1,"wangming",90);
    Student stu2(stu1);// 复制对象
    return 0;
}
```
思考：如果要定义重载定义>用于判断二个学生的分数高低，怎么实现？

静态成员、友员、模板

- 静态成员函数（没有 this 指针）

```
//阅读程序，了解静态数据成员与静态成员函数。
#include<iostream>
using namespace std;
class Student
{
    public:
        Student()
        {   count++;}
        ~Student()
        {   --count;}
        static int num()
        {   return count;}
    private:
        static int count;
};
int Student::count = 0;
int main()
{
    Student s1, s2,s3[3],*s4;
    cout<< Student::num() << endl;

    Student s5;
    cout << Student::num() << endl;
    return 0;
}
```

思考：1）如果在主函数中加入以下复合语句

```
    {
        Student   t1,t2;    //复合语句中的局部对象
        cout <<   Student::num() << endl;
    }
```

结果会怎么样？

2）如果再在主函数外加入全局对象的定义 Student s;结果又会怎样？

- 友元

友元可以访问类中的私有成员，友元类中的所有成员函数都是另一个类的友元函数，可以直接访问该类中的私有的和保护的成员。要注意的是，友元是单向的，不可传递的。

- 类模板

```
//P98 例 4-11 一维数组类模板
#include <iostream>
using namespace std;
```

```
#define N 100
template <class T>       //声明模板参数也可以写为 template <typename T>
class Array
{public:
        Array(T aa[],int nn)      //类模板中处理数据的类型用模板参数 T 代替
        {   n=nn;
        for(int i=0;i<n;i++)
                data[i]=aa[i];
        }
        T getmax()                //类模板中处理数据的类型用模板参数 T 代替
        {   T max=data[0];
            for(int i=1;i<n;i++)
                if(data[i]>max)
                    max=data[i];
            return max;
        }
        T getmin()                //类模板中处理数据的类型用模板参数 T 代替
        {   T min=data[0];
            for(int i=1;i<n;i++)
                if(data[i]<min)
                    min=data[i];
            return min;
        }
private:
        T data[N];                //类模板中处理数据的类型用模板参数 T 代替
        int n;
};
int main()
{      int a[]={1,2,3,4,5,6,7,8};
        Array<int> arr1(a,sizeof(a)/sizeof(int));       //定义类模板对象
        cout<<"max="<<arr1.getmax()<<endl;
        cout<<"min="<<arr1.getmin()<<endl;
        double b[]={1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8};
        Array<double> arr2(b,sizeof(b)/sizeof(double)); //定义类模板对象
        cout<<"max="<<arr2.getmax()<<endl;
        cout<<"min="<<arr2.getmin()<<endl;
        return 0;
}
```

思考：以上模板定义的数组个数是固定的，如果是动态的，应该怎么实现？

- 运算符重载

双目运算重载为成员函数时参数的个数为 1 个，双目运算重载为友元函数时参数个数为 2 个，并不是所有的运算符都可以重载，哪些不能重载？

继承性与多态性

继承性与多态性

派生类由基类派生，其成员有自身的成员及派生而来的成员（基类的构造函数和析构函数不能继承），一般情况下是公有派生，如果不加说明就是私有派生。派生类中被继承的成员访问属性会发生改变，这与基类中的成员属性及派生方式有关。派生类与基类之间可以赋值运算。

```
P144 例 6-1 中 set 成员函数，改为无参构造函数
#include<iostream>
#include<string>
using namespace std;
class Student
{public:
        void set()      //改为无参的构造函数
        {   cout<<"请输入学号、姓名、分数："<<endl;
            cin>>num>>name>>score;
        }
        void display()          //成员函数 display 的定义
        {   cout<<"num:"<<num<<endl;
            cout<<"name:"<<name<<endl;
            cout<<"score:"<<score<<endl;
        }
protected:  //受保护的
        int num;
        string name;
        float  score;
};
class Student1:public Student
{ public:
        void set1()    //改为无参的构造函数
        {   set();          //原有类 Student 中有的，继承下来的
            cout<<"请输入年龄、地址："<<endl;
            cin>>age>>addr;
        }
        void display1()     //新类 Student1 中增加的
        {   display();     //原有类 Student 中有的，继承下来的
            cout<<"age:"<<age<<endl;
            cout<<"addr:"<<addr<<endl;
        }
 private:
        int age;         //新增加的数据成员，年龄
        string addr;     //新增加的数据成员，地址
};
int main()       //主函数也要有相应的改变
{   Student1 stu;
    stu.set1();
    stu.display1();
```

```
    return 0;
}
```

- 有子对象的派生类的构造函数

```
//例 6-5 有子对象的派生类的构造函数。(只保留了构造函数部分，其它成员函数删除了)
#include<iostream>
#include<string>
using namespace std;
class Date
{public:
    Date(int y,int m,int d):year(y),month(m),day(d)
    { cout<<"Date"<<endl;}  //这个输出是为了方便看构造函数的执行顺序
private:
    int year,month,day;
};
class Student
{public:
    Student(int n,string na,float s):num(n),name(na),score(s)
    {cout<<"Student\n"; } //增加了输出是为了方便看构造函数的执行顺序
protected:
        int num;
        string name;
        float  score;
};
class Student1:protected Student
{ public:
    Student1(int n,string na,float s,int a,string ad)://参数 a 没用
            Student(n,na,s),birthday(1996,12,12),addr(ad){ }
  private:
        Date birthday;    //子对象
        string addr;
};
int main()
{
    Student1 stu(20201,"wangli",100,18,"NanJing");//年龄的参数不需要
    return 0;
}
```

如果有析构函数，程序的运行结果应该怎样？

```
//例 6-7 阅读程序，了解派生类析构函数的执行顺序
#include<iostream>
using namespace std;
class Meba
{public:
```

```
        Meba(){cout<<"Meba"<<"    ";}
        ~Meba(){cout<<"~Meba"<<"    ";}
};
class Mebb
{public:
        Mebb(){cout<<"Mebb"<<"    ";}
        ~Mebb(){cout<<"~Mebb"<<"    ";}
};
class A
{public:
        A(){cout<<"A"<<"    ";}
        ~A(){cout<<"~A"<<"    ";}
protected:
    Meba a;//基类中有子对象
};
class B: public A
{public:
        B(){cout<<"B"<<"    ";}
        ~B(){cout<<"~B"<<"    ";}
private:
        Mebb b;//派生类中有子对象
};
int main()
{   B b;
        return 0;
}
```

- 派生类的定义

```
//例 6-22 定义一个点类，再定义一个圆类。因为一个圆是点的放大，所以用继承来实现。
#include<iostream>
#include<cmath>
using namespace std;
class Point
{protected:
    double x, y;
public:
    static double PI;                          //静态数据成员，类共享
public:
    Point(double a=0, double b=0):x(a),y(b){} //构造函数
    void moveTo(double a, double b){ x = a, y = b; }    //点的移动
};
double Point::PI = 3.14159265;                  //静态数据成员在类外赋值
class Circle : public Point
{  double radius;
```

```
public:
    Circle(const Point& p=Point(), double r=0):
          Point(p),radius(r){}                          //构造函数
  void moveTo(double a, double b){ x = a, y = b; }    //移动点
  void modifyRadius(double r){ radius = r; }             //修改半径
  double getArea()const{ return radius*radius*Point::PI;}//计算面积

};
int main()
{   Point p(0,0);
    Circle c(p,10);
    c.moveTo(2,2);
    c.modifyRadius(100);
    cout<<c.getArea()<<endl;
    return 0;
}
```

- 虚基类

例 6-13 虚基类的构造函数

```
#include<iostream>
#include<string>
using namespace std;
class Base
{public:
        Base(string na){name=na;cout<<name<<endl;}
private:
        string name;
};
class Derived11:virtual public Base
{public:
        Derived11(string na1,string na2):Base(na1) //base 不会真的构造
        {name=na2;cout<<name<<endl;}
private:
        string name;
};
class Derived12:virtual public Base
{public:
      Derived12(string na1,string na2):Base(na1) //base 不会真的构造
     {name=na2;cout<<name<<endl;}
private:
        string name;
};
class Derived2:public Derived11,public Derived12
{public:
        Derived2(string na1,string na2,string na3,string na4):
```

```
            Base(na1),Derived11(na1,na2),Derived12(na1,na3)//base 在这儿真的构造
          { name=na4;cout<<name<<endl;}
    private:
            string name;
    }
    int main()
    {   Derived2 d("Base","Derived11","Derived12","Derived2");
        return 0;
    }
```

思考：如果不加 virtual，程序要怎么改，运行结果是什么？

如有以下三个类，其继承关系如下：

```
#include <iostream>
#include <string>
using namespace std;
class Teacher                                   //声明类 Teacher(教师)
{
public:
     Teacher(string nam,int a, string t)              //构造函数
     {name=nam;       age=a;       title=t;}
  protected:
     string name;      int age;      string title;      //姓名、年龄、职称
};
class Student                                    //定义类 Student(学生)
{public:
    Student(string nam,char s,float sco)
    { name=nam;         sex=s;         score=sco;}   //构造函数
    protected:
        string name;     char sex;     float score;      //姓名、性别、成绩
 };
class Graduate:public Teacher,public Student //声明多重继承的派生类 Graduate(研究生)
 {public:
    Graduate(string nam,int a,char s, string t,float sco,float w):   //构造函数
          Teacher(nam,a,t),Student(nam,s,sco),wage(w) { }
    void show( )                                                    //输出研究生的有关数据
     {cout<<"name:"<<Student::name<<endl;//用作用域运算符解决二义性问题
      cout<<"age:"<<age<<endl;
      cout<<"sex:"<<sex<<endl;
      cout<<"score:"<<score<<endl;
      cout<<"title:"<<title<<endl;
      cout<<"wages:"<<wage<<endl;
      }
  private:
     float wage;                          //工资
```

```
 };
int main( )
{ Graduate grad1("Wang-li",24,'f',"assistant",89.5,1234.5);
   grad1.show( );
   return 0;
}
下面通过增加一个虚基类来解决二义性问题
#include <iostream>
#include <string>
using namespace std;

class Person                              //定义一个虚基类
{public:
      Person(string na):name(na){}
protected:
      string name;
};
class Teacher:virtual public Person                    //声明类 Teacher(教师)
{
public:
      Teacher(string na,int a, string t):Person(na)    //构造函数
      {age=a;         title=t;}
   protected:
      int age;       string title;                     //姓名、年龄、职称
};
class Student:virtual public Person                    //定义类 Student(学生)
{
public:
      Student(string na,char s,float sco):Person(na)
    {       sex=s;          score=sco;}                //构造函数
    protected:
         char sex;     float score;       //姓名、性别、成绩
 };
class Graduate:public Teacher,public Student//声明多重继承的派生类 Graduate(研究生)
 {public:
    Graduate(string nam,int a,char s, string t,float sco,float w):   //构造函数
           Person(nam),Teacher(nam,a,t),Student(nam,s,sco),wage(w) { }
    void show( )                                                    //输出研究生的有关数据
     {cout<<"name:"<<name<<endl;           //用虚基类解决了二义性问题
       cout<<"age:"<<age<<endl;
       cout<<"sex:"<<sex<<endl;
       cout<<"score:"<<score<<endl;
       cout<<"title:"<<title<<endl;
       cout<<"wages:"<<wage<<endl;
```

```
    }
  private:
    float wage;                          //工资
 };
int main( )
 {Graduate grad1("Wang-li",24,'f',"assistant",89.5,1234.5);
  grad1.show( );
  return 0;
}
```

- 多态性

  编译时的多态由函数重载来实现，运行时时多态由虚函数来实现

```
//例 6-17 一个多态性的程序实例
#include<iostream>
using namespace std;
class Point
{public:
        Point(double a=0, double b=0){x=a,y=b;}
    virtual double area(){return 0;}
protected:
        double x, y;
};
class Circle:public Point
{public:
    Circle(double a=0,double b=0, double r=0):Point(a,b),radius(r){}
    double  area(){ return radius * radius * 3.1415926; }
protected:
    double radius;
};
class Cylinder:public Circle
{public:
        Cylinder(double a=0,double b=0,double r=0,double h=0):
    Circle(a,b,r),height(h){}
        double area() {return 2*Circle::area()+2*3,1415926*radius*height;}
 protected:
        float height;
};
int main()
{   Point p(1,1);
    cout<<p.area()<<endl;
    Circle c(1,1,100);
    cout<<c.area()<<endl;
    Cylinder cy(1,1,10,10);
    cout<<cy.area()<<endl;
```

```
        Point *pRef=&c;
        cout<<pRef->area()<<endl; //输出的是圆的信息，不是点的信息
        pRef=&cy;
        cout<<pRef->area()<<endl;//输出的是圆柱体的信息，不是点的信息
        return 0;
    }
```

思考：

如果定义一个函数

```
void outArea(Point &p)
{
    cout << p.area()<<endl;
}
```

代入不同层的对象，可以得到相应层对象的面积。实现了多态性。

- 抽象类

  抽象类是包含有纯虚函数的类，不可以实例化。

文件

文件分为文本文件和二进制文件，文件的读写追加操作。

```
//例 读取文本文件 test.txt 中的一个整型数据给变量 score
#include<iostream>
#include<string>
#include<fstream>
using namespace std;
int main()
{
    string errInfo;
    ifstream infile;
    int score;
    infile.open("test.txt", ios::in);
    if (!infile)
        errInfo = "error";
    infile >> score;   //读数据给变量 score
    cout << score<<endl;
    return 0;
}
```

思考：请编写一个函数，实现从文件中读取数据

```
void readdata(ifstream *f,int &score)
{
    *f >> score;
}
```

怎么调用呢？

异常处理

如果上例需要异常处理，那么，

rdaddata(&infile,score)在 try 中，后面紧跟着 catch。

考虑如果发生异常，请 throw 异常信息。