第一次实验

1 实验一：点类

1.1 实验目的

复习面向对象编程, 构造函数, 友元函数, 运算符重载等.

1.2 实验内容

定义一个点类, 并进行测试.

1.3 设计与编码

1.3.1 本实验用到的理论知识

1. 面向对象编程
2. 构造函数
3. 友元函数
4. 运算符重载

1.3.2 主函数

Listing 1: 1_Point.cpp

```
1 #include <iostream>
2 #include "Point.h"
3
4 int main()
```

```
{   Point p,p1,p2,p3;
    cin>>p;       //输入0 0  进行测试
    cout<<"p="<<p<<'\n';
    p.move(1,2);          //1 2
    cout<<"new p="<<p<<'\n';
    p1=p;       //1 2
    cout<<"p="<<p<<" p1="<<p1<<'\n';
    p2=p1*3;      //3  6
    cout<<"p1="<<p1<<" p2=p1*3="<<p2<<'\n';
    p3=p1+p2;
    cout<<"p1="<<p1<<" "<<"p2="<<p2<<" p3=p1+p2="<<p3<<
        '\n';
    if (p3>p2) cout<<"p3="<<p3<<" p2="<<p2<<" p3>p2"<<'
        \n';
    if (p1<p2) cout<<"p1="<<p1<<"  p2="<<p2<<" p1<p2"<<
        '\n';
    if (p1==p1) cout<<"p1==p1"<<'\n';

    return 0;
}
```

1.3.3 头文件

Listing 2: Point.h

```
#ifndef _POINT_H
#define _POINT_H
using namespace std;
class Point {
 public:
    Point();
    Point(int, int);
    ~Point(){};
    Point(const Point &p) : x(p.x), y(p.y) {}
```

```
10      void move(int newX,int newY);
11      int getX() const { return x; }
12      int getY() const { return y; }
13      Point& operator=(const Point& p);
14      Point operator+(const Point& p);
15      Point operator*(const int i);
16      bool operator>(const Point& p);
17      bool operator<(const Point& p);
18      bool operator==(const Point& p);
19      friend istream & operator >> (istream &in,  Point
            &p);
20      friend ostream & operator << (ostream &out, const
            Point &p);
21  private:
22      int x, y;
23  };
24  Point::Point()
25  {
26      x = 0;
27      y = 0;
28  }
29  Point::Point(int a, int b)
30  {
31      x = a;
32      y = b;
33  }
34  void Point::move(int newX, int newY)
35  {
36      this->x = newX;
37      this->y = newY;
38  }
39  Point& Point::operator=(const Point& p)
40  {
```

```
41      this->x = p.x;
42      this->y = p.y;
43      return *this;
44  }
45  Point Point::operator+(const Point& p)
46  {
47      Point tmp = Point(x + p.x, y + p.y);
48      return tmp;
49  }
50  Point Point::operator*(const int i)
51  {
52      Point tmp = Point(x * i, y * i);
53      return tmp;
54  }
55  bool Point::operator>(const Point& p)
56  {
57      if (x > p.x && y > p.y)
58      {
59          return true;
60      }
61      else
62      {
63          return false;
64      }
65  }
66  bool Point::operator<(const Point& p)
67  {
68      if (x < p.x && y < p.y)
69      {
70          return true;
71      }
72      else
73      {
```

```
74          return false;
75      }
76  }
77  bool Point::operator==(const Point& p)
78  {
79      if (x == p.x && y == p.y)
80      {
81          return true;
82      }
83      else
84      {
85          return false;
86      }
87  }
88  istream& operator>>(istream& in, Point& p)
89  {
90      cout<<"****************************************"
            <<endl;
91      cout<<"         Please input two integers.       "
            <<endl;
92      cout<<"****************************************"
            <<endl;
93      in>>p.x>>p.y;
94      return in;
95  }
96  ostream& operator<<(ostream& out, Point& p)
97  {
98      out<<"("<<p.getX()<<", "<<p.getY()<<") ";
99      return out;
100 }
101 #endif  //_POINT_H
```

1.4 运行与测试

运行结果如图 1所示.

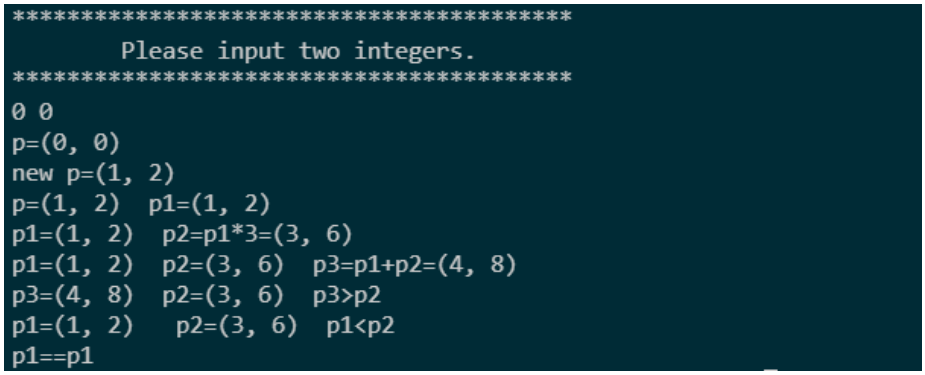

图 1: Result

1.5 总结与心得

通过本题简单复习了类与对象的编程技巧.

2 实验二: 教师, 领导类

2.1 实验目的

复习面向对象编程, 多重继承等.

2.2 实验内容

分别声明 Teacher 类和 Cadre 类, 采用多重继承方式由这两个类派生出新类 Teacher_Cadre 类

2.3 设计与编码

2.3.1 本实验用到的理论知识

1. 面向对象编程
2. 派生类

3. 多重继承

2.3.2 主函数

Listing 3: 2_teacher.cpp

```cpp
#include <iostream>
#include <string>
#include "Teacher.h"
#include "Cadre.h"
#include "Teacher_Cadre.h"
int main()
{
    Teacher_Cadre person("Zhang",67,'M',"prof","
        chairman","u","086-123456", 7654321);
    person.show();
    return 0;
}
```

2.3.3 头文件

Listing 4: Teacher.h

```cpp
#ifndef _TEACHER_H
#define _TEACHER_H
using namespace std;
class Teacher
{
public:
    Teacher(string _name, int _age, char _sex, string
        _title, string _addr, string _tel);
    void display();
protected:
    string name;
    int age;
```

```
12      char sex;
13      string title;
14      string addr;
15      string tel;
16  };
17  Teacher::Teacher(string _name, int _age, char _sex,
        string _title, string _addr, string _tel):
18      name(_name), age(_age), sex(_sex), title(_title),
            addr(_addr), tel(_tel)
19      {}
20  void Teacher::display()
21  {
22      cout<<"name:"<<name<<endl
23          <<"age:"<<age<<endl
24          <<"sex:"<<sex<<endl
25          <<"title:"<<title<<endl
26          <<"address:"<<addr<<endl
27          <<"tel:"<<tel<<endl;
28  }
29  #endif
```

Listing 5: Teacher_Cadre.h

```
1   #ifndef _CADRE_H
2   #define _CADRE_H
3   using namespace std;
4   class Cadre
5   {
6   public:
7       Cadre(string _name, int _age, char _sex, string
            _post, string _addr, string _tel);
8       void display();
9   protected:
10      string name;
11      int age;
```

```
12      char sex;
13      string post;
14      string addr;
15      string tel;
16  };
17  Cadre::Cadre(string _name, int _age, char _sex, string
        _post, string _addr, string _tel):
18      name(_name), age(_age), sex(_sex), post(_post),
            addr(_addr), tel(_tel)
19      {}
20  void Cadre::display()
21  {
22      cout<<"name:"<<name<<endl
23          <<"age:"<<age<<endl
24          <<"sex:"<<sex<<endl
25          <<"post:"<<post<<endl
26          <<"address:"<<addr<<endl
27          <<"tel:"<<tel<<endl;
28  }
29  #endif
```

Listing 6: Cadre.h

```
1  #ifndef _TEACHERCADRE_H
2  #define _TEACHERCADRE_H
3  using namespace std;
4  class Teacher_Cadre: public Teacher, public Cadre
5  {
6  private:
7      double wage;
8  public:
9      Teacher_Cadre(string _name,int _age,char _sex,
           string _title,string _post,string _addr,string
           _tel,float _wage);
10     void show();
```

```
11 };
12 Teacher_Cadre::Teacher_Cadre(string _name,int _age,
       char _sex,string _title,string _post,string _addr,
       string _tel,float _wage):
13     Teacher(_name, _age, _sex, _title, _addr, _tel),
14     Cadre(_name, _age, _sex, _post, _addr, _tel),
15     wage(_wage)
16     {}
17 void Teacher_Cadre::show()
18 {
19     Teacher::display();
20     cout<<"post:"<<Cadre::post<<endl;
21     cout<<"wages:"<<wage<<endl;
22 }
23 #endif
```

2.4 运行与测试

运行结果如图所示.

2.5 总结与心得

通过本题简单复习了继承与派生的相关编程方法, 本题主要与多重继承有关.

3 实验三: Shape

3.1 实验目的

复习面向对象编程, 多态性, 抽象类, 虚函数.

3.2 实验内容

定义一个抽象类 Shape, 由它派上生出 3 个派生类: Circle, Rectangle, Triangle, 用一个函数 printArea 分别输出以上三者的面积, 3 个图形的数据

在定义对象时给定

3.3 设计与编码

3.3.1 本实验用到的理论知识

1. 多态性
2. 抽象类
3. 虚函数

3.3.2 主函数

Listing 7: 3_Shape.cpp

```cpp
#include <iostream>
#include "Shape.h"
int main()
{
    Circle cir(2);
    printArea(cir);
    Rectangle rec(6, 3);
    printArea(rec);
    Triangle tri(7, 5);
    printArea(tri);
    return 0;
}
```

3.3.3 头文件

Listing 8: Shape.h

```cpp
#ifndef _SHAPE_H
#define _SHAPE_H
using namespace std;
const double pi = 3.14156;
```

```
5  class Shape
6  {
7  public:
8      virtual double area() const = 0;
9      virtual void shapeName() const = 0;
10 };
11 class Circle: public Shape
12 {
13 public:
14     Circle(double r);
15     virtual double area() const;
16     virtual void shapeName() const {cout<<"Circle:
           ";}
17 protected:
18     double radius;
19 };
20 Circle::Circle(double r):radius(r){}
21 double Circle::area() const
22 {
23     return pi * radius * radius;
24 }
25 class Rectangle: public Shape
26 {
27 protected:
28     double width;
29     double height;
30 public:
31     Rectangle(double w, double h);
32     virtual double area() const;
33     virtual void shapeName() const {cout<<"Rectangle:
            ";}
34 };
```

```
35 Rectangle::Rectangle(double w, double h):width(w),
       height(h){}
36 double Rectangle::area() const
37 {
38     return width * height;
39 }
40 class Triangle: public Shape
41 {
42 protected:
43     double width;
44     double height;
45 public:
46     Triangle(double w, double h);
47     virtual double area() const;
48     virtual void shapeName() const {cout<<"Triangle:
           ";}
49 };
50 Triangle::Triangle(double w, double h):width(w),
       height(h){}
51 double Triangle::area() const
52 {
53     return 0.5 * width * height;
54 }
55 void printArea(Shape& s)
56 {
57     s.shapeName();
58     cout<<s.area()<<endl;
59 }
60 #endif
```

3.4 运行与测试

运行结果如图 2所示.

图 2: Result

3.5 总结与心得

本实验实现了从抽象类定义派生类, 和定义纯虚函数.

4 实验四: Shape

4.1 实验目的

复习面向对象编程, 多态性, 抽象类, 虚函数, 基类指针数组.

4.2 实验内容

定义一个抽象类 Shape, 由它派上生出 5 个派生类: Circle, Square, Rectangle, Trapezoid, Triangle. 用虚函数分别计算几种图形的面积, 并求它们的和.

4.3 设计与编码

4.3.1 本实验用到的理论知识

1. 多态性
2. 抽象类
3. 虚函数
4. 基类指针数组

4.3.2 主函数

Listing 9: 4_Shape.cpp

```
1 #include <iostream>
```

```cpp
#include "Shape.h"
int main()
{
    Circle cir(1);
    Square squ(4);
    Rectangle rec(2, 3);
    Trapezoid tra(2, 3, 4);
    Triangle tri(5, 6);
    printArea(cir);
    printArea(squ);
    printArea(rec);
    printArea(tra);
    printArea(tri);
    Shape* p[5] = {&cir, &squ, &rec, &tra, &tri};
    cout<<"total area: "<<total(p, 5)<<endl;
    return 0;
}
```

4.3.3 头文件

Listing 10: Shape.h

```cpp
#ifndef _SHAPE_H
#define _SHAPE_H
using namespace std;
const double pi = 3.14156;
class Shape
{
public:
    virtual double area() const = 0;
    virtual void shapeName() const = 0;
};
class Circle: public Shape
{
```

```
13 public:
14     Circle(double r);
15     virtual double area() const;
16     virtual void shapeName() const {cout<<"Circle:
           ";}
17 protected:
18     double radius;
19 };
20 Circle::Circle(double r):radius(r){}
21 double Circle::area() const
22 {
23     return pi * radius * radius;
24 }
25 class Square: public Shape
26 {
27 public:
28     Square(double l);
29     virtual double area() const;
30     virtual void shapeName() const {cout<<"Square:
           ";}
31 protected:
32     double length;
33 };
34 Square::Square(double l):length(l){}
35 double Square::area() const
36 {
37     return length * length;
38 }
39 class Rectangle: public Shape
40 {
41 protected:
42     double width;
43     double height;
```

```
44 public:
45     Rectangle(double w, double h);
46     virtual double area() const;
47     virtual void shapeName() const {cout<<"Rectangle:
           ";}
48 };
49 Rectangle::Rectangle(double w, double h):width(w),
     height(h){}
50 double Rectangle::area() const
51 {
52     return width * height;
53 }
54 class Trapezoid: public Shape
55 {
56 protected:
57     double upper;
58     double lower;
59     double height;
60 public:
61     Trapezoid(double u, double l, double h);
62     virtual double area() const;
63     virtual void shapeName() const {cout<<"Trapezoid:
           ";}
64 };
65 Trapezoid::Trapezoid(double u, double l, double h):
     upper(u), lower(l), height(h){}
66 double Trapezoid::area() const
67 {
68     return 0.5 * height * ( upper + lower );
69 }
70 class Triangle: public Shape
71 {
72 protected:
```

```
73      double width;
74      double height;
75  public:
76      Triangle(double w, double h);
77      virtual double area() const;
78      virtual void shapeName() const {cout<<"Triangle:
            ";}
79  };
80  Triangle::Triangle(double w, double h):width(w),
        height(h){}
81  double Triangle::area() const
82  {
83      return 0.5 * width * height;
84  }
85  void printArea(Shape& s)
86  {
87      s.shapeName();
88      cout<<s.area()<<endl;
89  }
90  double total(Shape* shape[], int count)
91  {
92      double sum = 0;
93      for (int i = 0; i < count; i++)
94      {
95          sum += shape[i]->area();
96      }
97      return sum;
98  }
99  #endif
```

4.4 运行与测试

运行结果如图 3所示.

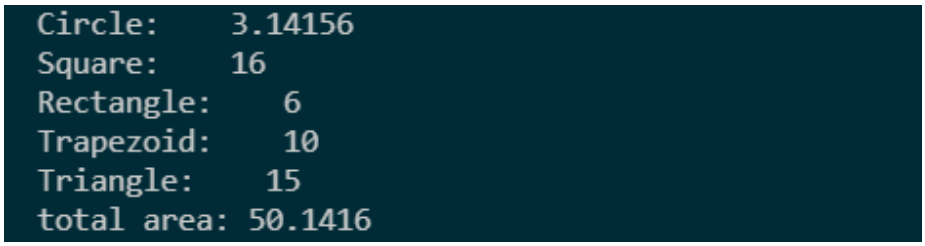

图 3: Result

4.5 总结与心得

本实验应用面向对象编程, 多态性, 抽象类, 虚函数, 基类指针数组. 定义一个抽象类 Shape, 并由它派上生出 5 个派生类. 最后定义函数求和.