C++程序设计

C++ Programming

实验报告 7-8

Experiment Report

专业(Major)： 计算机科学与技术 2502 班

学号(ID)：202512898

姓名(Name)：田佩宁

提交日期(Date)：2025 年 12 月 28 日

成绩(Score)：

东北大学悉尼智能科技学院

Sydney Smart Technology College, Northeastern University

实验编号（Experiment No.）：7

实验名称（Experiment Name）：多态机制的实现

实验目的（Experiment Purpose）：

（1） 掌握虚函数的概念和实现；

（2） 掌握纯虚函数的概念和实现；

（3） 掌握多态机制的概念和实现。

实验内容（Experiment Contents）：

多态性字面意思是指多种形态，是指发出同样的消息被不同类型的对象接收时有可能导致完全不同的行为。利用多态性技术，可以调用同一个函数名的函数，实现完全不同的功能。函数重载、运算符重载和虚函数都可以实现多态。

C++的多态必须满足两个条件：

1、必须通过基类的指针或者引用调用虚函数。

2、被调用的函数是虚函数，且必须完成对基类虚函数的重写。

如下示例代码所示：

```
class Person //成人
{
  public:
  virtual void fun()
   {
       cout << "全价票" << endl; //成人票全价
   }
};
class Student : public Person //学生
{
   public:
   virtual void fun() //子类完成对父类虚函数的重写
   {
       cout << "半价票" << endl;//学生票半价
   }
};
void BuyTicket(Person* p)
{
   p->fun();
}

int main()
{
   Student st;
   Person p;
   BuyTicket(&st);//子类对象切片过去
   BuyTicket(&p);//父类对象传地址
```

第 1 题：定义基本图形 Shape 抽象基类、包含纯虚函数 Area()计算图形面积；二维图形抽象基类 Shape2D 继承 Shape 类，添加纯虚函数 Perimeter()计算二维图形周长；三维图形抽象基类 Shape3D 继承 Shape 类，增加纯虚函数 Volume()计算三维图形体积。

1）Shape2D 派生类矩形 Rectangle、圆形 Circle、椭圆 Ellipse，重写 Area()和 Perimete()函数；

2）Shape3D 派生类球体 Sphere、圆柱体 Cylinder、圆锥体 Cone，重写 Area()和 Volume()函数。

3）实现静态类 SumofShape 计算图形数组的面积、周长和体积之和。（**提示：可以利用 dynamic_cast<Shape2D*>和 dyamic_cast<Shape3D*>分别将 Shape 类型指针转换为 Shape2D 和 Shape3D 指针用于计算面积和体积**）。

下图为部分图形的周长、面积和体积的计算公式，供参考。

1） 椭圆周长 P 与面积 S 公式

$$P = 2\pi b + 4(a - b)$$

$$S = \pi ab$$

其中 a,b 分别是椭圆的长、短半轴。（注意：上述给的椭圆周长公式仅仅近似公式）

2） 球体表面积 S 与体积 V 公式

$$S = 4\pi r^2$$

$$V = \frac{4}{3}\pi r^3$$

3） 圆柱体表面积 S 与体积 V 公式

$$S = 2\pi r(r + h)$$

$$V = \pi r^2 h$$

4） 锥体表面积 S 与体积 V 公式

$$S = \pi r(r + \sqrt{h^2 + r^2})$$

$$V = \frac{1}{3}\pi r^2 h$$

本次实验给出了部分代码，需要将加粗部分的注释要求的功能实现后完成程序的运行。

```
#include<iostream>
#include<cmath>
using namespace std;
const  double  PI = 3.1415926;

//图形的抽象基类
class  Shape {
public:
    virtual  double  Area() = 0;  //求面积,  纯虚函数;
};
//求二维图形的抽象基类
class  Shape2D :public  Shape {
public:
    virtual  double  Perimeter() = 0;  //求周长,  纯虚函数;
};
//求三维图形的抽象基类
class  Shape3D :public  Shape {
public:
    virtual  double  Volume() = 0;  //求体积,  纯虚函数;
};
//派生类：矩形
class  Rectangle : public  Shape2D
{
```

```
public:
    Rectangle(double length, double width) :length_(length), width_(width) {}
    virtual  double Area() {return length_ * width_;}
    virtual  double Perimeter() {return 2 * length_ + 2 * width_;}
private:
    double length_;
    double width_;
};
//派生类：椭圆
class  Ellipse :public  Shape2D
{
public:
    Ellipse(double semimajor,double semiminor):semimajor_(semimajor),
semiminor_(semiminor) {}
    virtual  double Area() {//求椭圆的面积}
    virtual  double Perimeter() {//求椭圆的周长}
private:
    double semimajor_;
    double semiminor_;
};
//派生类：圆
class  Circle : public  Shape2D
{
public:
    Circle(double radius) :radius_(radius) {}
    virtual  double Area() {//求圆的面积}
    virtual  double Perimeter() {//求圆的周长}
private:
    double radius_;
};
//派生类：球体
class  Sphere :public  Shape3D
{
public:
    Sphere(double radius) :radius_(radius) {}
    virtual  double Area() {//求球体的面积}
    virtual  double Volume() {//求球体的体积}
private:
    double radius_;
};
//派生类：圆柱体
class  Cylinder :public  Shape3D
{
public:
    Cylinder(double height, double radius) :height_(height), radius_(radius) {}
    virtual  double Area() {//求圆柱体的面积}
    virtual  double Volume() {//求圆柱体的体积 }
```

```
private:
    double height_;
    double radius_;
};
//派生类：圆锥体
class  Cone : public  Shape3D
{
public:
    Cone(double height, double radius) :height_(height), radius_(radius) {}
    virtual  double Area() {//求圆锥体的面积}
    virtual  double Volume() {//求圆锥体的体积}
private:
    double height_;
    double radius_;
};
class  SumofShape
{
public:
    //静态函数  计算图形数组面积之和
    static  double  SumofArea
    //静态函数  计算图形数组周长之和
    static  double  SumofPerimeter
    //静态函数  计算图形数组的体积之和
    static  double  SumofVolume
private:
    SumofShape() {}  //定义私有构造函数，禁止实例化
};
int  main() {
    Rectangle  rectangle(2, 3);
    Ellipse  ellipse(8, 4);
    Circle  circle(3);
    Sphere  sphere(3);
    Cylinder  cylinder(3, 5);
    Cone  cone(3, 4);
    Shape* shape_array[] = { &rectangle,  &ellipse,  &circle,  &sphere,  &cylinder,
&cone };
    double  sum_of_area = SumofShape::SumofArea(shape_array, 6);
    double  sum_of_perimeter = SumofShape::SumofPerimeter(shape_array, 6);
    double  sum_of_volume = SumofShape::SumofVolume(shape_array, 6);
    cout << "Sum of Area: " << sum_of_area << "\nSum of Perimeter: " << sum_of_perimeter
<< "\nSum of Volume: " << sum_of_volume << endl;
    return  0;
}
```

实验结果（Experiment Results）：

```cpp
#include <iostream>
#include <cmath>
const double PI = 3.1415926;

// 基类 //
class Shape {
public:
    virtual double area() const = 0;
    virtual ~Shape() = default;
};

class Shape2D :public Shape {
public:
    virtual double perimeter() const = 0;   // perimeter n.周长
    virtual ~Shape2D() = default;
};

class Shape3D :public Shape {
public:
    virtual double volume() const = 0;
    virtual ~Shape3D() = default;
};

// 二维 //
class Rectangle :public Shape2D {
    double w, h;
public:
    Rectangle(double width, double height) :w(width), h(height) {}
    double area()       const {return w * h;}
    double perimeter()  const {return 2 * (w + h);}
};

class Circle :public Shape2D {
    double r;
public:
    explicit Circle(double radius) :r(radius) {}        // radius n.半径
    double area()       const {return PI * r * r;}
    double perimeter()  const {return 2 * PI * r;}
};

class Ellipse :public Shape2D{
    double a, b;
public:
    explicit Ellipse(double semimajor, double semiminor) :a(semimajor), b(semiminor) {}
    double area()       const {return PI * a * b;}
    double perimeter()  const {return 2 * PI * b + 4 * (a - b);}
};

// 三维 //
class Sphere :public Shape3D {     // sphere n.球体
    double r;
public:
    explicit Sphere(double radius) : r(radius) {}
    double area()   const {return 4 * PI * r * r;}
    double volume() const {return 4.0 / 3 * PI * r * r * r;}
};

class Cylinder :public Shape3D {   // cylinder n.圆柱体
    double h, r;
public:
    Cylinder(double height, double radius) : h(height), r(radius) {}
    double area()   const {return 2 * PI * r * r + 2 * PI * r * h;}
    double volume() const {return PI * r * r * h;}
};
```

```cpp
class Cone :public Shape3D {       // cone n.圆锥体
    double h, r;
public:
    Cone(double height, double radius) : h(height), r(radius) {}
    double area() const {
        double l = std::sqrt(x: r * r + h * h); // 母线
        return PI * r * r + PI * r * l;
    }
    double volume() const {return PI * r * r * h / 3;}
};

class SumOfShape {
    SumOfShape(){}
public:
    static double totalArea(Shape** v, int len) {
        double sum = 0.0;
        for (int i=0; i<len; i++) {
            sum += v[i]->area();
        }
        return sum;
    }
    static double totalPerimeter(Shape** v, int len) {
        double sum = 0.0;
        for (int i = 0; i < len; i++) {
            Shape2D* shape2d = dynamic_cast<Shape2D*>(v[i]);
            if (shape2d != nullptr) {
                sum += shape2d->perimeter();
            }
        }
        return sum;
    }

    static double totalVolume(Shape** v, int len) {
        double sum = 0.0;
        for (int i = 0; i < len; i++) {
            Shape3D* shape3d = dynamic_cast<Shape3D*>(v[i]);
            if (shape3d != nullptr) {
                sum += shape3d->volume();
            }
        }
        return sum;
    }
};


int main() {
    Rectangle rectangle(width: 2, height: 3);
    Ellipse ellipse(semimajor: 8, semiminor: 4);
    Circle circle(radius: 3);
    Sphere sphere(radius: 3);
    Cylinder cylinder(height: 3, radius: 5);
    Cone cone(height: 3, radius: 4);

    Shape* shape_array[] = {[0]=&rectangle, [1]=&ellipse, [2]=&circle,
        [3]=&sphere, [4]=&cylinder, [5]=&cone,};
    std::cout<<"Sum of Area: "<<SumOfShape::totalArea(v: shape_array, len: 6)<<"\n";
    std::cout<<"Sum of Perimeter: "<<SumOfShape::totalPerimeter(v: shape_array, len: 6)<< "\n";
    std::cout<<"Sum of Volume: "<<SumOfShape::totalVolume(v: shape_array, len: 6)<<"\n";
}
```

```
Sum of Area: 612.327
Sum of Perimeter: 69.9823
Sum of Volume: 398.982
田佩宁 202512898
```

心得体会（Experience）：

通过虚函数与动态绑定，体会到多态的灵活性：基类指针调用派生类方法，实现接口统一与代码复用，深化了对面向对象设计思想的理解。

实验编号（Experiment No.）：8

实验名称（Experiment Name）：文件操作与文件流

实验目的（Experiment Purpose）：

掌握输入输出流、文件的操作方法，包括文件的打开与关闭、ASCII文件的操作、文件操作与文件流、字符串流等。

实验内容（Experiment Contents）：13 章课后习题第 4 题。

建立两个磁盘文件 f1.txt 和 f2.txt，编写程序实现以下工作：

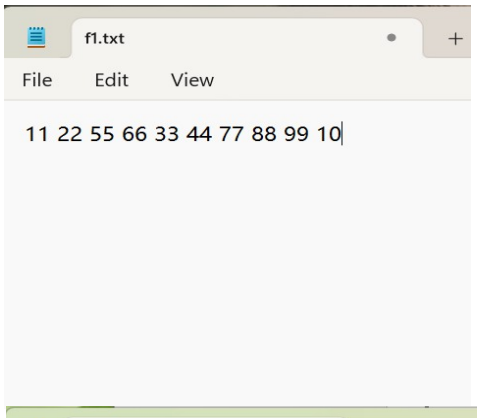
（1）从文件 in.txt 输入 20 个整数，分别存放在两个磁盘文件 f1.txt 和 f2.txt 中（每个文件中放 10 个整数）；

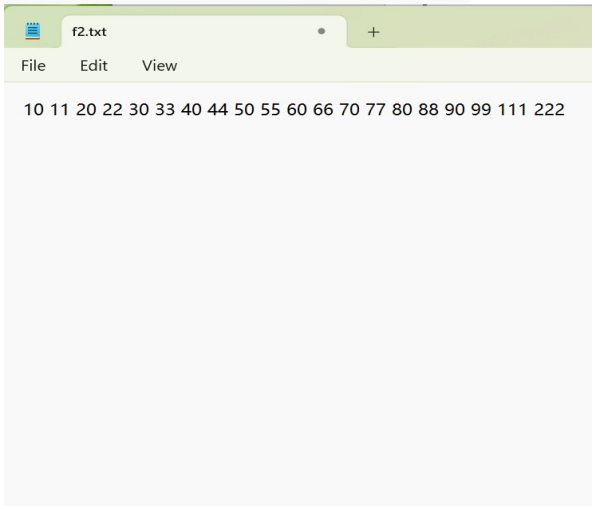
（2）从 f1.txt 读入 10 个数，然后存放到 f2.txt 文件原有数据的后面；

（3）从 f2.txt 中读入 20 个整数，将它们按照从小到大的顺序存放到 f2.txt（不保留原来的数据）。

实验结果（Experiment Results）：

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
using namespace std;

/* 从文件读 n 个整数到 vec */
void readInts(const string& file, vector<int>& vec, size_t n)
{
    ifstream fin(s: file, mode: ios::out);
    int x;
    while (fin >> x && n--) {vec.push_back(x);}
}

/* 向文件追加 vec 中所有整数，空格分隔 */
void appendInts(const string& file, const vector<int>& vec)
{
    ofstream fout(s: file, mode: ios::app);
    for (int x : vec) {fout << x << ' ';}
}

/* 把 vec 中整数覆写到文件，空格分隔 */
void writeInts(const string& file, const vector<int>& vec)
{
    ofstream fout(s: file);
    for (size_t i = 0; i < vec.size(); ++i)
        fout << vec[i] << (i + 1 == vec.size() ? '\n' : ' ');
}

int main()
{
    const string src = "in.txt";
    const string f1  = "f1.txt";
    const string f2  = "f2.txt";

    /* 读入 20 个整数 */
    vector<int> data;
    readInts(file: src, &vec: data, n: 20);
    /* 拆成两份，各 10 个，分别写入 f1.txt 和 f2.txt */
    vector<int> d1(first: data.begin(), last: data.begin() + 10);
    vector<int> d2(first: data.begin() + 10, last: data.end());
    writeInts(file: f1, vec: d1);
    writeInts(file: f2, vec: d2);

    /* 把 f1 的 10 个数追加到 f2 末尾 */
    vector<int> tmp;
    readInts(file: f1, &vec: tmp, n: 10);
    appendInts(file: f2, vec: tmp);

    /* 把 f2 当前全部 20 个整数读回，排序后覆写 */
    vector<int> merged;
    readInts(file: f2, &vec: merged, n: 20);
    sort(first: merged.begin(), last: merged.end());
    writeInts(file: f2, vec: merged);
}
```





心得体会（Experience）：

C++文件流操作简洁高效，fstream 类封装完善。通过本次实验，掌握了文件的读写、状态检测及异常处理，深化了对流式数据处理的理解。