电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2017221302006

姓 名 周玉川

（实验） 课程名称 面向对象程序设计C++

理论教师 赵白银

实验教师 赵白银

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：周玉川 学号：2017221302006 指导教师： 赵太银**

**实验地点：信软学院楼西400 实验时间：2019.12.11、2019.12.18**

**一、实验名称：微型C++控制台应用开发**

**二、实验学时：8**

**三、实验目的：** 充分运用所学的C++的全部核心内容，开发一个微型的C++控制台应用程序，使学生能够：

1） 熟练掌握面向对象的核心知识：封装、继承、多态，基本掌握泛型编程；

2） 基本掌握OOA、OOD方法；熟练掌握OOP方法；

3） 掌握C++应用的开发过程，掌握编程工具的使用；

4） 初步掌握建模工具的使用；

5） 基本掌握阅读、调试程序的能力。

**四、实验原理：**

本次实验基于MFC开发C++桌面应用程序，实现可视化操作。

**五、实验内容：**

一位小学教师Ken希望完成这样的任务：针对于小学生正在学习四边形(quadrangle)的特性，编写一个小软件，能够随机在屏幕上显示矩形(rectangle)、正方形(square)、平行四边形(parallelogram)、梯形(trapezoid)和菱形(diamond)五种形体之一，同时显示该形体的轮廓形状和关键数据（随机产生），学生根据给出的关键数据计算形体的面积并输入，软件判断其结果的正确性。在学习过程中，软件记录产生的每一个形体，在学生选择不再继续后，将其学习的过程重放一遍（及显示所有学过的形体数据、形状及结果），用以重温，加深印象。

根据上述描述，编写程序，完成Ken老师的需求。

编码要求：

1. 理清形体概念之间的关系，然后用（多）继承来实现这些关系。
2. 使用抽象类来表示顶层概念。
3. 矩形、平行四边形用长和高表示；正方形用边长表示；梯形用两条平行边长和高表示；菱形用两条对象线长表示。
4. 使用链表模板来存储数据。
5. 为链表模板设计迭代器。

要体现多态。

**六、实验器材（设备、元器件）：**

1. 项目需用仪器设备名称：PC机一台。
2. 所需软件环境：任意C++开发平台。

**七、实验步骤：**

1. **先编写形体类头文件**
2. 形体类diamond，菱形

|  |
| --- |
| #pragma once  #include<string>  #include<iostream>  #include "parallelogram.h"  using namespace std;  /\*菱形\*/  class Diamond : public Parallelogram  {  public:  Diamond(int l = 5, string tag = "Diamond") :Parallelogram(l, l, tag) {};  ~Diamond() {};  void draw() {  std::cout << what() << ":宽=" << Width()  << ",高=" << Height() << std::endl;;  };  double area() const { return (double)width \* height / 2; };  string what() const { return tag; };  int Width() { return width; };  int Height() { return height; };  Diamond& operator= (const Diamond& di)  {  tag = di.tag;  width = di.width;  height = di.height;  return \*this;  }  }; |

1. 形体类Parallelogram，平行四边形

|  |
| --- |
| #pragma once  #include<string>  #include<iostream>  #include "quadrangle.h"  using namespace std;  /\*平行四边形\*/  class Parallelogram : public Quadrangle  {  public:  Parallelogram(int w = 5, int h = 7, string nm = "Parallelogram") :width(w), height(h), Quadrangle(nm) {};  Parallelogram(const Parallelogram& q) : width(q.width), height(q.height), Quadrangle(q.tag) {};  ~Parallelogram() {};  void draw() const { std::cout << "draw Parallelogram" << std::endl;; };  double area() const { return width \* height; };  void draw() {  std::cout << what() << ":宽=" << Width()  << ",高=" << Height() << std::endl;;  };  int& Width() { return width; }; //获取矩形的宽。如果是梯形，还需要一个成员int& Width2()来获取另一条平行边的长度  int& Height() { return height; };//获取矩形的高  Parallelogram& operator=(const Parallelogram& pa)  {  tag = pa.tag;  width = pa.width;  height = pa.height;  return \*this;  }  protected:  int width;  int height;  }; |

1. 形体类Quadrangle四边形，为抽象类，其子类必须覆盖其抽象函数

|  |
| --- |
| #pragma once  #include <string>  /\*四边形\*/  class Quadrangle  {  protected:  std::string tag;  public:  Quadrangle(std::string t) : tag(t) {};  std::string what() const { return tag; };  virtual double area() const = 0; // 纯虚函数  virtual void draw() = 0; // 纯虚函数  };  typedef Quadrangle\* QUADPTR; |

1. 形体类Rectangle矩形

|  |
| --- |
| #pragma once  #include<string>  #include<iostream>  #include "parallelogram.h"  using namespace std;  /\*矩形\*/  class Rectangle : public Parallelogram  {  public:  Rectangle(int w = 4, int h = 8, string nm = "Rectangle") :Parallelogram(w, h, nm) {};  Rectangle(const Rectangle & rec) :Parallelogram(rec) {};  ~Rectangle() {};  string what()const { return Parallelogram::what(); }  double area()const { return Parallelogram::area(); }  void draw() {  std::cout << what() << ":宽=" << Width()  << ",高=" << Height() << std::endl;;  };  int& Width() { return width; }  int& Height() { return height; };  Rectangle& operator=(const Rectangle& re)  {  tag = re.tag;  width = re.width;  height = re.height;  return \*this;  }  }; |

1. 形体类Square，正方形

|  |
| --- |
| #pragma once  #include<string>  #include<iostream>  #include "rectangle.h"  using namespace std;  /\*正方形\*/  class Square : public Rectangle  {  public:  Square(int w = 6, string nm = "Square") :Rectangle(w, w, nm) {};  ~Square() {};  void draw() {  std::cout << what() << ":宽=" << Width()<< std::endl;;  };  double area() const { return width \* width; };  string what() const { return Rectangle::what(); };  int& Width() { return width; };  Square& operator=(const Square& sq)  {  tag = sq.tag;  width = sq.width;  height = sq.height;  return \*this;  }  }; |

1. 形体类梯形Trapezoid

|  |
| --- |
| #pragma once  #include<string>  #include<iostream>  #include "quadrangle.h"  using namespace std;  /\*梯形\*/  class Trapezoid : public Quadrangle  {  public:  Trapezoid(int w1 = 5.0, int w2 = 7.0, int h = 8.0, string tag = "Trapezoid") :topWidth(w1), bottomWidth(w2), height(h), Quadrangle(tag) {};  ~Trapezoid() {};  void draw(){ std::cout << what()<<":上底=" <<TopWidth()<<",下底="<<BottomWidth()  <<",高="<<Height()<< std::endl;; };  double area() const { return (double)(bottomWidth + topWidth) \* height / 2; };  string what() const { return tag; };  int& BottomWidth() { return bottomWidth; };  int& TopWidth() { return topWidth; };  int& Height() { return height; };  Trapezoid& operator=(const Trapezoid& tr)  {  tag = tr.tag;  bottomWidth = tr.bottomWidth;  topWidth = tr.topWidth;  height = tr.height;  return \*this;  }  protected:  int topWidth;  int bottomWidth;  int height;  }; |

1. **编写链表List头文件**

完成类List构造函数，析构函数以及函数的编写。

|  |
| --- |
| #pragma once  #include "quadrangle.h"  #include<iostream>  struct Node  {  QUADPTR quad;  Node\* next;  Node(QUADPTR q) :quad(q), next(nullptr) {};  };  typedef bool predicate\_t(QUADPTR);  class List  {  private:  Node\* head, \* tail;  size\_t len;  public:  // 默认构造函数  List()  {  head = nullptr;  tail = nullptr;  len = 0;  };  // 复制构造函数  List(const List& list);  // 析构函数  ~List();  bool isEmpty() const { return len == 0; }; // 判断链表为空  void empty(); //清空链表  List& operator+=(QUADPTR p);  List& operator+=(const List& list); //拼接两个链表  List& operator=(const List& p);  QUADPTR operator[](size\_t index);  friend List operator+(const List& list, QUADPTR p);  friend List operator+(const List& list1, const List& list2);  List& push\_back(QUADPTR p);  void traverse(predicate\_t pred);  size\_t size() const { return len; };  };  List& List::operator+=(QUADPTR p)  {  return push\_back(p);  };  //拼接两个链表  List& List::operator+=(const List& list)  {  if (list.isEmpty())  return \*this;  Node\* t = list.head;  while (t != list.tail)  {  push\_back(t->quad);  t = t->next;  }  return push\_back(t->quad);  };  List& List::operator=(const List& p)  {  empty(); //先释放  return \*this += p; //再复制  };  QUADPTR List::operator[](size\_t index)  {  if (index < 0 || index >= size())  {  return nullptr; //超出范围返回空  }  if (index == (size() - 1))  {  return tail->quad;  }  Node\* t = head;  while (index--)  {  t = t->next;  }  return t->quad;  };  List operator+(const List& list, QUADPTR p)  {  List t(list);  return t += p;  };  List operator+(const List& list1, const List& list2)  {  List t(list1);  return t += list2;  };  List::List(const List& list)  {  if (!list.isEmpty())  {  empty();// 释放  head = new Node(list.head->quad);  Node\* list\_beCopy\_iterator = list.head, \* list\_copy\_iterator = head, \* list\_new\_node;  if (list.head != list.tail)  {  while (list\_beCopy\_iterator != list.tail)  {  list\_new\_node = new Node(list\_beCopy\_iterator->quad);  list\_beCopy\_iterator = list\_beCopy\_iterator->next;  list\_copy\_iterator->next = list\_new\_node;  list\_copy\_iterator = list\_new\_node;  }  tail = list\_copy\_iterator;  tail->next = nullptr;  }  else  {  tail = head;  }  len = list.size();  }  else  {  List();  }  }  List::~List()  {  empty();  }  List& List::push\_back(QUADPTR p)  {  Node\* tmp = new Node(p); // 为加入的Node结构体分配一个新的内存  if (tail == nullptr)  { // 判断链表是否为空  head = tmp; //链表为空时，把head和tail都置为tmp  tail = tmp;  }  else  {  tail->next = tmp; // 将链表的尾部指针的next指向新申请的Node结构体  tail = tmp; // 将链表的尾部置为新申请的Node结构体  }  len += 1;// 加上新加入的size  return \*this;  }  void List::traverse(predicate\_t pred)  {  if (isEmpty())  {  std::cout << "链表是空的" << std::endl;  }  else  {  int count = 0;  Node\* temp = head;  while (temp != nullptr && count < len)  {  std::cout << "第" << ++count << "个" << std::endl;  pred(temp->quad);  temp = temp->next;  }  }  };  void List::empty()  {  if (!isEmpty())  {  Node\* before = head, \* after;  while (before != tail)  {  after = before->next;  delete before;  before = after;  }  delete before;  head = nullptr;  tail = nullptr;  len = 0;  }  }; |

1. **编写Mission头文件以及main函数**

完成生成随机形体类以及记录到List列表并结束时复习的功能，还要 出现相关提示。

|  |
| --- |
| #pragma once  #include<iostream>  #include<cstdlib>  #include<ctime>  #include<cstring>  #include "diamond.h";  #include "List.h";  #include "parallelogram.h";  #include "quadrangle.h";  #include "rectangle.h";  #include "square.h";  #include "trapezoid.h";  #define RAND\_MAX 30 //定义形体类长度的最大数  #define NUM 5 //形体类个数  using namespace std;  class Mission {  public:  Mission() { cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Missiion Start!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl; };  void Start();  ~Mission() { cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Missiion End!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl; };  };  bool access(QUADPTR p) {  p->draw();  cout << "area=" << p->area() << endl;  return true;  }  void Mission::Start()  {  int width, height, topWidth, number;  double answer;  QUADPTR q = new Trapezoid();  List list;  char cmd;  srand((unsigned)time(NULL));  cout << "================ C++实验微型C++控制台应用开发 =================" << endl;  cout << "=====++++++++++++2017221302006 周玉川+++++++++++ =============" << endl;  while (true)  {  cout << "================q、Q：结束测试 ================================" << endl;  cout << "================其他字符：继续测试 ============" << endl;  cout << "user#";  cmd = getchar();  if (cmd == 'q' || cmd == 'Q') {  cout << "============临走之前，复习一下 ============" << endl;  list.traverse(access);  cout << "============再见 ============" << endl;  break;  }  while (cmd != '\n' && getchar() != '\n');  width = rand() % RAND\_MAX;  height = rand()% RAND\_MAX;  topWidth = rand()% RAND\_MAX;  number = rand() % NUM;  /\*0：Diamond  \*1：Parallelogram  \*2：Rectangle  \*3：Square  \*4：Trapezoid  \*/  switch (number) {  case 0:  q = new Diamond(width);  break;  case 1:  q = new Parallelogram(width, height);  break;  case 2:  q = new Rectangle(width, height);  break;  case 3:  q = new Square(width);  break;  case 4:  q = new Trapezoid(width, topWidth, height);  break;  };  q->draw();  cout << "计算随机生成图形的面积，并输入你的答案：" << endl << "answer:";  cin >> answer;  while (getchar() != '\n');  if (answer == q->area())  {  cout << "恭喜你，答对了" << endl;  }  else  {  cout << "很遗憾，答错了" << endl;  }  list.push\_back(q);  }  } |

**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

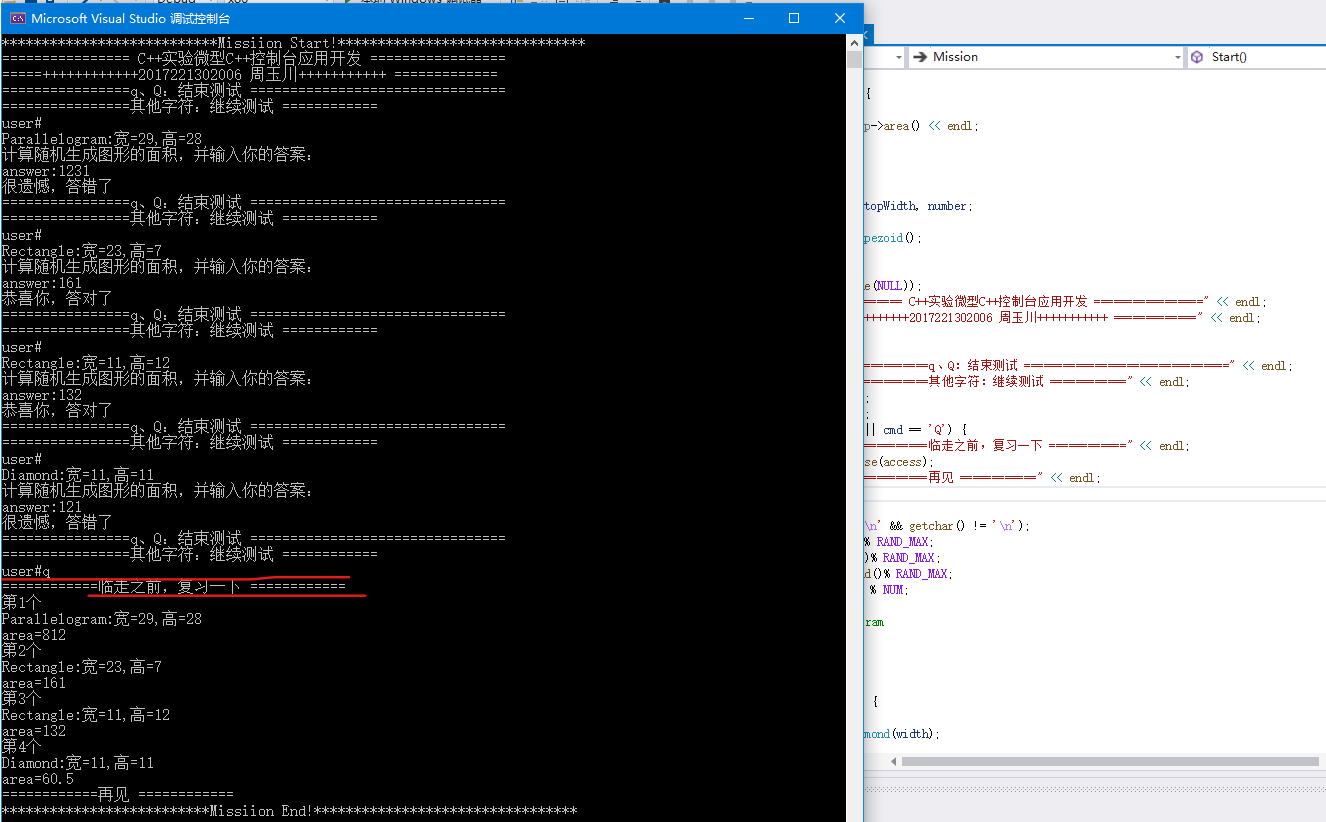
输出结果图，随机出形体类，数据结果判断对错，把这次形体类加入链表中，复习的时输出。q退出时复习一遍，从链表中取出数据。

图8-1 结果图

**九、总结及心得体会：**

通过这次形体类的实验，把学过的C++的主要内容，包括继承，虚函数，运算符重载等全部实践并复习了一遍，相当于一次课程大终结，这次实验贯穿整个学习C++的过程，实验成功结束，也标志着课程的圆满成功即将结束。

在赵老师的带领下学习过c++的基本知识和概念后，在本实验中解决了实际问题，明白了类，继承关系以及接口的作用，并会这种面向对象的思想会一直影响我今后的专业生涯。

学习C++的过程中，实践也就是上手码代码给我很大的满足感，对今后面试时现场写代码也有极大的帮助。

**十、对本实验过程及方法、手段的改进建议：**

一次实验似乎很少，但是已经够起到作用了，还是希望实践多一点。

**报告评分：**

**指导教师签字：**