电子科技大学信息与软件工程学院

**实验报告**

点名册序号 304-3 学 号 2015220103022

姓 名 张健顺

（实验） 课程名称 面向对象程序设计(C++)

理论教师 陈安龙

实验教师 陈安龙

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：张健顺 学号：2015220103022 指导教师：陈安龙**

**实验地点：信软楼西304 实验时间：16.10.25**

1. **实验名称：基于MFC的C++桌面应用开发**
2. **实验学时：4**
3. **实验目的：**

充分运用所学的C++的数据封装、继承与派生、多态等全部核心内容，补充完整一个基于MFC的应用，使学生能够基本掌握OOA、OOD方法；熟练掌握OOP方法；初步了解基于Windows平台的桌面图形化应用的开发过程，掌握编程工具的使用；初步掌握建模工具的使用；基本掌握阅读、调试程序的能力。

**四、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，同时显示该形体的特性和关键数据（随机产生），学生复习形体的特性，然后根据给出的关键数据计算形体的面积，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，用以重温，加深印象。

根据上述描述，需要完成：

1. **必做内容**

根据Ken老师的要求，需要至少编写六个类：

* Rect //注意：类名不要使用Rectangle
* Square
* Parallelogram
* Trapezoid
* Diamond
* List

（注：这六个类必须以上述名字命名）

其中，前五个类用于描述五种形体。五种形体不用顶点坐标的表示形式，而只是简单地用它们的特征值表示。例如：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。要求为这五种形体编写相应的类，每个类的设计要求如下：

1. Quadrangle类必须成为抽象类，是其它形体类的祖先。它拥有如下成员：

|  |  |  |
| --- | --- | --- |
| **成员** | **类型** | **是否纯虚** |
| **name** | 数据 | / |
| **area** | 函数 | 是 |
| **draw** | 函数 | 是 |
| **what** | 函数 | 可选 |

1. 其它形体类之间的继承关系请自行拟定。其中，Parallelogram类必须拥有如下虚成员：

* Width(); //返回宽
* Height(); //返回高

1. 每个形体类必须完成如下操作：
2. 设置标志名属性name。五种形体的标志名必须是Parallelogram、Rectangle、Diamond、Trapezoid和Square之一（第一个字母大写，其余小写）；
3. 重载area()成员；
4. 重载draw()成员；
5. 重载Width()成员以返回宽；
6. 重载Height()成员以返回高；
7. 提供Width2()成员以返回第二条平行边长（仅对梯形）
8. List类必须拥有如下成员函数：

* size()；//返回列表中的节点数
* push\_back()；//将数据添加到列表末尾
* operator[]；//返回指定下标的数据
* traverse()；//遍历，需要一个访问函数作为参数
* pop\_back() //用来删除列表的最后一个节点，其原型为：

void pop\_back();

1. 重温过程实际上就是遍历List类对象的过程。
2. **选作内容**

List类实际上是一种称为“容器”的类。除了容纳形体指针，其实它还可以容纳任何类型的对象。在本选作内容中，

1. 请将List类该为模板类，使其能容纳任何类型的对象；
2. 请为List类编写迭代器。要求如下：

* 该迭代器的类名为Iterator;
* 该迭代器必须重载如下运算符：
* ++：迭代器后移
* !=：两个迭代器比较
* \*：返回迭代器指向节点的数据域中的值，即quad指针。
* =：两个迭代器的复制
* +。该运算符函数的原型为：

Iterator operator+(int i);

其功能为：假设当前迭代器（的内部指针）指向了第k个节点（从0开始计数），那么+运算符返回一个迭代器，该迭代器是在当前迭代器的基础上向后移动i个位置得到的，即该迭代器指向了第k+i个节点。例如：

Iterator itr = list.begin() + 5;

那么迭代器itr就指向了从列表的头节点往后数第5个结点。

为List类编写begin()和end()成员。

要求：

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. 初步了解基于Windows平台的桌面图形化应用的开发过程，掌握编程工具的使用；

4）初步掌握建模工具的使用。

1. **实验器材（设备、元器件）：**

PC机一台，Windows 8.1，Visual Studio 2013

1. **实验步骤：**
2. 先在纸上完成系统设计、类设计；
3. 使用任意一款建模工具建立系统的模型，至少画出类图

根据设计，编写各种形体类代码。建议：

1. 请将所有类型声明放在同一个.h文件中，所有类的实现放在同一个.cpp文件中；
2. 先不将以上代码整合到系统中，单独编写测试模块完成所有类的功能测试。
3. 五种形体的名字标识要求使用如下字符串（单词的第一个字母大写）：
   * 四边形：Quadrangle
   * 平行四边形：Parallelogram
   * 梯形：Trapezoid
   * 矩形：Rectangle
   * 菱形：Diamond
   * 正方形：Square

注：以上只是用于属性name的名字表示，而非类名！

1. 在用于测试的main函数中：
   * 不能预先定义形体对象，而采用随机的方式生成形体。这需要使用两个函数：

* 随机数发生器初始化：srand(unsigned(time(NULL))); 该函数只需调用一次；这次调用最好放在main()函数中；
* 随机数发生器：rand();该函数产生0-INT\_MAX之间的伪随机正整数。请思考：如何产生合理范围内的整数？

为了使用以上函数，需要包含以下几个头文件：

cstdlib

ctime

* + 形体的所有关键数据（例如宽和高）也采用随机的方式生成

1. 程序流程
   * 程序首先只考虑学习模式。在该模式下：
     1. 随机产生一个形体及其关键数据，然后“画”出这个形体（即简单地显示这个形体的关键数据）。此后要求学习者输入面积，程序判断输入是否正确，如果不正确则要求继续输入，直到正确为止；
     2. 此后程序询问学习者是否继续。如果是，则重复上一步；
     3. 否则，程序进入复习模式。
   * 暂时不考虑复习流程。

继续第二次实验的编码过程，直到完成所有编码和测试工作。

仿照第二次实验的流程进行List类的编码和测试。

1. 对于List类重载[]说明

如果有：List list;

那么list[2]返回list对象内部链表中第2个（从0开始计数）节点中quad域存储的数据，亦即类型为QUADPTR的指针。

提示：需要用到与遍历相似的算法。

建议：该函数的原型为：

QUADPTR& operator[](int index);

1. 程序流程的完善
   * 程序首先进入学习模式。在该模式下：
2. 随机产生一个形体及其关键数据，并将形体加入到列表中，然后“画”出这个形体（包括显示形体的关键数据）。此后要求学习者输入面积，程序判断输入是否正确，如果不正确则要求继续输入，直到正确为止；
3. 此后程序询问学习者是否继续。如果是，则重复上一步；
4. 否则，程序进入复习模式。
   * 在复习模式下，程序“画”出学习者学过的每一个形体。此后程序结束。

继续第四次实验的编码过程，直到完成List类的编码和测试。

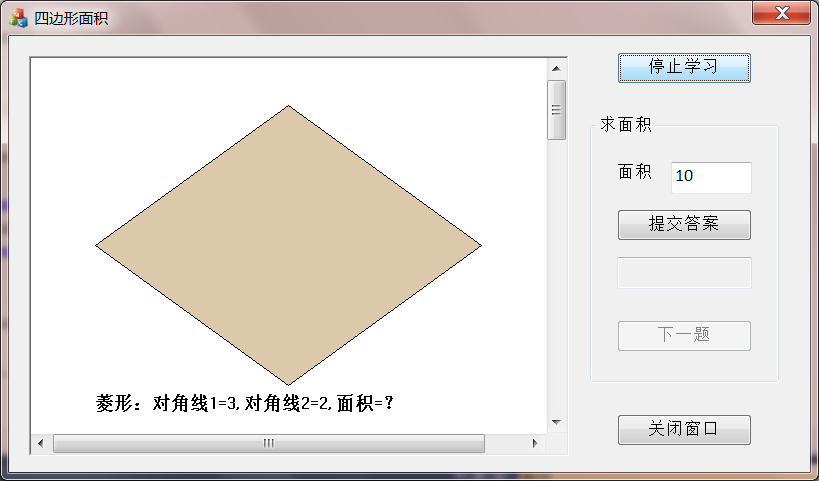
1. 将提供的压缩包解压到任一磁盘上，例如D:，将生成一个experiment文件夹；
2. 将你写好的如下文件拷贝到D:\experiment\experiment文件夹下：
   1. List.h和list.cpp
   2. 形体类的.h和.cpp文件
3. 双击D:\experiment\experiment.sln，启动该解决方案，然后将上述源文件添加到该解决方案中。具体做法是：在“解决方案资源管理器”面板中，右击解决方案的名字，然后选择“添加”->“现有项…”，在弹出的对话框中选择需要的源文件按“添加”按钮。此后在文件列表中拖拽源文件名，将其放到适当的列表项下面。
4. 打开stdafx.h文件，然后将所有你编写的头文件用#include指令添加在最后。注意头文件包含的先后顺序；
5. 保存解决方案；
6. 启动解决方案，按如下描述添加代码：
   1. 在形体类的头文件插入如下一行：

#include "Canvas.h"

* 1. 对每一种形体，删除draw()中原来的调试代码，改成如下一行：

Canvas::drawQuad(this);

* 1. 在experimentDlg.cpp模块前面的位置定义一个List类的全局对象list（必须使用这个名字），并在canvas.cpp中将该list对象说明成是外部的(extern)。在做好的代码中已经用符号对//{{和//}}标识出了你需要添加代码的地方，此后均按此操作；
  2. 在experimentDlg.cpp中编写用于列表遍历时用到的access()函数。它应该是全局的。当然，你也可以把它设计成为CexperimentDlg类的静态成员。
  3. 将随机生成形体的代码编写或粘贴在CexperimentDlg::createAquad()中。在这段代码中，关键一点是一定要将生成形体的指针加入到列表对象中；
  4. 其他需要添加代码的地方在做好的程序中用符号对//{{和//}}标识出来，请搜索这些符号以完成程序；
  5. 生成解决方案然后调试。程序运行的截图如下：



1. 界面操作流程：
   1. 按“开始学习”按钮，画出一个形体，在输入框中输入面积，然后按“提交答案”，如果正确，将显示“OK”；否则显示“答案错误”，此时重新输入直到正确为止；
   2. 按“下一题”重复学习；
   3. 按“结束学习”按钮后，所有学过的形体都会绘制在窗口中；
   4. 按“关闭窗口”退出应用。

继续完成系统的整合调试工作。

继续完成系统的整合调试工作。在完成后对代码进行Review，并考虑如何优化。最后要完成整个系统。此后需撰写实验报告并提交电子档。

注：如有学生提前完成上述工作，那么请完成选作内容。

附件1：示例代码

附件2：系统所需解决方案

附件3：选作内容指导

附件4：选作内容替换代码

1. **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）**

Quad.h

//四边形

#ifndef QUAD\_H

#define QUAD\_H

#include<iostream>

#include"Canvas.h"

using namespace std;

class Quadrangle

{

protected:

string name;

public:

Quadrangle(string \_name="Quadrangle") :name(\_name){};

virtual double area() const = 0;

virtual void draw() const = 0;

virtual string what() const{return name;}

};

typedef Quadrangle \* QUADPTR;

#endif

Rect.h

//矩形

#ifndef RECT\_H

#define RECT\_H

#include"Quad.h"

class Rect : public Quadrangle

{

public:

Rect(int \_width = 5, int \_height = 7) :

Quadrangle("Rectangle"),width(\_width), height(\_height){}

string what()const{ return name; }

double area()const{ return width\*height; }

void draw()const{ Canvas::drawQuad(this); }

int& Width() { return width; };

int& Height(){ return height; };

private:

int width;

int height;

};

#endif

Para.h

//平行四边形

#ifndef PARA\_H

#define PARA\_H

#include"Quad.h"

#include"Canvas.h"

class Parallelogram : public Quadrangle

{

public:

Parallelogram(int \_width = 5, int \_height = 5)

:Quadrangle("Parallelogram"), width(\_width), height(\_height){}

string what()const{ return name; }

double area()const{ return width\*height; }

void draw()const{ Canvas::drawQuad(this); }

virtual int& Width();

virtual int& Height();

protected:

int width;

int height;

};

#endif

Squa.h

//正方形

#ifndef SQUA\_H

#define SQUA\_H

#include"Quad.h"

class Square: public Quadrangle

{

public:

Square(int \_SideLength = 5)

:Quadrangle("Square"), SideLength(\_SideLength){}

string what() const{ return name; }

double area() const{ return SideLength\*SideLength; }

void draw() const{ Canvas::drawQuad(this); }

int& Width() { return SideLength; };

int& Height(){ return SideLength; };

private:

int SideLength;

};

#endif

Trap.h

#ifndef TRAPEZOID\_H

#define TRAPEZOID\_H

#include"Quad.h"

class Trapezoid :public Quadrangle

{

public :

Trapezoid(int \_width=2, int \_width2=3, int \_height=7) :

Quadrangle("Trapezoid"), width(\_width), width2(\_width2), height(\_height){}

string what()const{ return name; }

double area()const{ return (width+width2)\*height/2; }

void draw()const{ Canvas::drawQuad(this); }

int& Width() { return width; };

int& Width2(){ return width2; };

int& Height(){ return height; };

private:

int width;

int width2;

int height;

};

#endif

Diam.h

//菱形\_

#ifndef DIAM\_H

#define DIAM\_H

#include"Quad.h"

class Diamond :public Quadrangle

{

public:

Diamond(int \_SideLength = 5, int \_height = 7)

:Quadrangle("Diamond"), SideLength(\_SideLength), height(\_height){}

string what()const{ return name; }

double area()const{ return SideLength\*height; }

void draw()const{ Canvas::drawQuad(this); }

int& Width() { return SideLength; };

int& Height(){ return height; };

private:

int SideLength;

int height;

};

#endif

List.h

#ifndef LIST\_H

#define LIST\_H

#include "Quad.h"

typedef void(\*ACCESSFUN)(QUADPTR);

class List

{

protected:

struct Node

{

QUADPTR quad;

Node \*next;

Node(QUADPTR q);

} \*head, \*tail;

size\_t len;

public:

List();

List(const List& s);

~List();

List& push\_back(const QUADPTR quad);

void pop\_back();

operator bool() const;

size\_t size() const;

void clear();

List& operator=(const List& l);

List& operator+=(const QUADPTR quad);

QUADPTR& operator[](size\_t index);

void traverse(ACCESSFUN f);

typedef Node\* \_range;

#endif

List.cpp

#include"stdafx.h"

List::List() : head(NULL), tail(NULL), len(0){}

List::List(const List& l) : head(l.head), tail(l.tail), len(l.len){}

List::~List(){clear();}

List::Node::Node(QUADPTR q) : quad(q){}

List& List::push\_back(const QUADPTR quad)

{

Node \*p = new Node(quad);

p->next = NULL;

if (tail == NULL) //first node

head = tail = p;

else

{

tail->next = p;

tail = p;

}

++len;

return \*this;

}

void List::pop\_back()

{

if (tail == NULL) return;

--len;

if (head == tail)

{

delete head;

head = tail = NULL;

return;

}

Node \*p = head, \*q = p->next;

while (q != tail)

{

p = p->next;q = q->next;

}

delete q;

p->next = NULL;

tail = p;

}

List::operator bool() const{return head != NULL;}

List& List::operator=(const List& l)

{

head = l.head;tail = l.tail;return \*this;

}

List& List::operator+=(const QUADPTR quad)

{

push\_back(quad);return \*this;

}

size\_t List::size() const{ return len;}

QUADPTR& List::operator[](size\_t index)

try

{

if (index >= len || len == 0) throw(0);

Node \* p = head;

size\_t i = 0;

while (i < index)

{

p = p->next;

++i;

}

return p->quad;

}

catch (int) { static QUADPTR s = NULL; return s; }

void List::clear()

{

Node \*p = head, \*q;

while (p != NULL)

{

q = p;

p = p->next;

delete q;

}

head = tail = NULL;

len = 0;

}

void List::traverse(ACCESSFUN f)

{

Node \*p = head;

while (p != NULL)

{

f(p->quad);

p = p->next;

}

}

1. **总结及心得体会：**

熟练掌握面向对象的核心知识：封装、继承、多态。只有深入理解了类和对象之间的关系，以及实际问题中各个类之间的关系才能更好地设计出相应的数据模型，对于类中的某些数据成员应当封装起来防止外界的不利因素改变本身具有的属性，通过自定义的一些特有的方法对相应的数据成员进行处理；在继承父类的属性和方法之后，由于子类自身的差异，有时需要重新定义方法或者属性，就不得不对父类的相关成员进行重载；多态又主要分为静态多态和动态多态，静态多态则主要体现在运算符重载，为了使链表的某些操作更加便于人的理解和使用，采用运算符重载的方法达到简化主函数中的代码的目的，至于动态多态，则主要体现在父类指针引用子类对象的相应应用中。

泛型编程在List链表的设计中得到了体现，链表的每一个节点的数据类型不再是我们以往常使用的基本数据类型而是我们自定义的类型，与此同时，节点的数据类型也不是固定的一成不变的类型，而是在我们自定义设计的形状类型中随机产生，在处理不同自定义类型的过程中，List链表的操作又完全一致，所以为了解决这一根本性问题，引入泛型编程，让代码具有可重用性，是程序具有普适性，也便于后期的维护。

**九、对本实验过程及方法、手段的改进建议：**

在开始一个完整的应用开发之前，最好是先建立相关模型，将实际问题抽象化，譬如数据模型，同时要梳理好各类型之间的关系，对于某些特定的功能的实现以及一些资源或者库的调用在开发之前根据需求做出详细的分析，同时也要规范整套的开发流程，落实每一步的具体任务。

对于一个完整的应用，还要对代码文件和资源文件进行规范化的管理，从而提高开发的效率。

对于基于MFC的应用开发方面的知识还是比较欠缺。之前无Windows平台的桌面图形化应用的开发经历，还是显得比较生疏。

对于迭代器的使用，譬如本次试验中List的遍历操作，还有待加强。迭代器其实本身就是泛型编程的体现。

**报告评分：**

**指导教师签字：**