**电子科技大学**

**实验报告**

**学生姓名：贾子阳 学号：2012224010018 指导教师：钱伟中**

**实验地点：学院楼机房 实验时间：周五晚**

**一、实验室名称：**软件实验室

**二、实验项目名称：**面向对象程序设计—泛型编程

**三、实验学时：**4

**四、实验原理：**

所谓泛型编程，就是以独立于任何特定类型的方式编写代码。这使得程序员在编写充分可重用代码时有了趁手的工具。

泛型其实是一种形式的静态多态，实现方式是类型参数化。一旦类型本身被参数化，那么我们就可以跃过类型限制带来的鸿沟，用一个类或者函数操纵多种类型不同的对象，并且不需要知道实现的细节。这无疑为代码的编写和维护带来的巨大的好处。

在C++中，泛型编程主要依托模板(template)来实现。例如：

template<typename T>

T abs(T a) { return a > 0 ? a : -a; }

其中，T就是参数化的类型。编译器会根据提供的值类型来自动生成对应版本的函数。

除了函数模板，C++还提供了类模板，为泛型编程打下了坚实的基础。

**五、实验目的：**

本实验通过编写一些简单的程序，使学生掌握泛型编程，并能基本运用模板的语法，培养理论联系实际和自主学习的能力，提高程序设计水平。

**六、实验内容：**

前面设计的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。在本实验中，

1. 请将List类改为模板类，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：
   1. 该迭代器的类名为Iterator;
   2. 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。

* 1. 为List类编写begin()和end()成员。

1. 如果时间足够，那么可以选做如下实验：
   1. 将List类的存储实现从单链表改为双向链表。修改过后的链表仍然拥有head和tail指针；
   2. 为List类添加逆向迭代器。该迭代器的行为与正向迭代器非常相似，只不过操作方向正好相反。这需要你为List类再添加两个成员：
      * + rbegin()：返回一个指向List尾部的迭代器；
        + rend()：返回一个指向List头部的迭代器。

**七、实验器材（设备、元器件）：**

PC机一台，装有C/C++语言集成开发环境。

**八、实验步骤：**

//以下修改基于实验三

/\*

没有改动的文件：

Canvas.h Diamond.h experiment.h experimentDlg.h Parallelogram.h Quadrangle.h Rectangle.h resource.h Square.h stdafx.h targetver.h Trapozoid.h

diamond.cpp experiment.cpp parallelogram.cpp rectangle.cpp square.cpp stdafx.cpp trapezoid.cpp

\*/

//experimentDlg.cpp

//list.h

#pragma once

#include <iostream>

#include <string.h>

#include <string>

#include <time.h>

using std::string;

using namespace std;

#include "Canvas.h"

#include "Quadrangle.h"

#include "Parallelogram.h"

#include "Rectangle.h"

#include "Diamond.h"

#include "Trapezoid.h"

#include "Square.h"

//list

template <typename T>

class List {

public:

friend class Node ;

class Node {

public:

T quad; //pro: qPtr

Node \*next;

Node(){}

~Node(){}

};

friend class Iterator;

class Iterator {

private :

public:

List <T> \* plist;

typename List <T> ::Node \*p;

Iterator (const List <T> & list ) :plist(const\_cast <List <T> \* > (&list)),p(list.head){}

Iterator ( Iterator & itr ) :plist( itr.plist) , p(itr.p){}

Iterator () : plist(NULL) , p(NULL) {}

Iterator & begin() {

return \*this;

}

Iterator end () {

return Iterator();

}

Iterator & operator = (const Iterator & itr ) {

plist = itr.plist;

p = itr.p;

return \*this;

}

Iterator & operator ++ () {

p = p->next;

return \*this;

}

bool operator != (const Iterator & itr) {

return p != itr.p;

}

T & operator \* () {

return p->quad;

}

Iterator operator + (int i) {

Iterator temp = \*this;

for (int j =0;j != i;j++){++temp;}

return temp;

}

};

protected:

void pushBack(T Quad ,Node \* node) {

Node \*newNode = node;

newNode->next=NULL;

this->tail->quad = Quad;

this->tail->next = newNode;

this->tail=this->tail->next;

this->num++;

return;

}

public:

Node \*head,\*tail ;

int num;

Iterator \*lItr;

List(){

num = 0;

head=new Node;

tail=head;

tail->next=NULL;

lItr = new Iterator;

lItr->p = head;

}

~List() {}

int size (){return num;}

void pop\_back(){

Node \*p =this->head;

int NUM = 1;

for (;NUM<=num-1;p=p->next){

NUM++;

}

delete(p);

num--;

return;

}

void traverse () {

Node \*p =this->head;

int NUM = 1;

for (;NUM<=this->num;p=p->next){

cout<<"-----------------------"<<NUM<<endl;

p->quad->draw();

cout<<"the area is :"

<<p->quad->area()

<<"\n"

<<endl;

NUM++;

}

return;

}

List & operator += (T Quad) {

Node \* node = new Node;

this->pushBack(Quad,node);

return \*this;

}

List & operator = (List & l) {

int n = l.num;

Node \*node = l.head;

for (int i = 1;i<=n;i++){

\*this+=(node->quad);

n++;

}

return \*this;

}

QUADPTR & operator [] (int index) {

Node \*node = this->head;

if (index<0){

throw int (index) ;

}

else if (this->num<=index){

throw long (index);

}

else {

for (int i =0 ;i<index;i++) {

node = node->next;

}

}

return node->quad;

}

Iterator begin () {

lItr->p = head;

return lItr->begin() ;

}

Iterator end () {

lItr->p = head;

return lItr->end();

}

};

//Canvas.cpp

// Canvas.cpp : 实现文件

//

#pragma once

#include "stdafx.h"

#include "experiment.h"

#include "Canvas.h"

//{{ 外部说明添加到这里

extern List <qPtr> list;

//}}

// Canvas

Canvas \* Canvas::pCanvas = NULL;

bool Canvas::isLearning = false;

IMPLEMENT\_DYNCREATE(Canvas, CScrollView)

Canvas::Canvas()

{

pCanvas = this;

quadMap.insert(KV("Parallelogram", para));

quadMap.insert(KV("Rectangle", rect));

quadMap.insert(KV("Square", sqr));

quadMap.insert(KV("Diamond", diam));

quadMap.insert(KV("Trapezoid", trap));

}

static CString formatString[] =

{

\_T("宽=%lf, 高=%lf, 面积="),

\_T("对角线1=%lf, 对角线2=%lf, 面积="),

\_T("平行边1=%lf, 平行边2=%lf, 高=%lf, 面积=")

};

Canvas::~Canvas()

{

}

BEGIN\_MESSAGE\_MAP(Canvas, CScrollView)

END\_MESSAGE\_MAP()

// Canvas 绘图

void Canvas::OnInitialUpdate()

{

CScrollView::OnInitialUpdate();

//CSize sizeTotal;

// TODO: 计算此视图的合计大小

//sizeTotal.cx = sizeTotal.cy = 100;

//SetScrollSizes(MM\_TEXT, sizeTotal);

}

union \_QUADPTR\_

{

Quadrangle \*q;

Parallelogram \*p;

Rect \*r;

Diamond \*d;

Square \*s;

Trapezoid \*t;

};

void Canvas::OnDraw(CDC\* pDC)

{

CDocument\* pDoc = GetDocument();

// TODO: 在此添加绘制代码

CRect rect;

GetClientRect(&rect);

CBrush br(RGB(220, 200, 170));

CBrush \* oldBrush = reinterpret\_cast<CBrush\*>(pDC->SelectObject (&br));

int xc, yc;

int vw = rect.Width (), vh = rect.Height ();

int sw = static\_cast<int>(vw \* 0.75 / 2.0), sh = static\_cast<int>(vh \* 0.75 / 2.0);

SIZE ad[5][4] =

{ { { -sw + 50, -sh }, { sw + 50, -sh }, { sw - 50, sh }, { -sw - 50, sh } },

{ { -sw, -sh }, { sw, -sh }, { sw, sh }, { -sw, sh } },

{ { -sh, -sh }, { sh, -sh }, { sh, sh }, { -sh, sh } },

{ { 0, -sh }, { sw, 0 }, { 0, sh }, { -sw, 0 } },

{ { -sw + 120, -sh }, { sw - 80, -sh }, { sw, sh }, { -sw, sh } },

};

double ratio = 1.0;

int yOffset = 120;

if (isLearning)

{

xc = vw / 2;

yc = vh / 2;

}

else

{

xc = 80;

yc = 60;

ratio = 0.3;

}

POINT vert[4];

int shape;

List<qPtr>::Iterator itr;

List<qPtr>::Iterator end = list.end();

List<qPtr>::Iterator begin = list.begin();

if (isLearning) begin = begin + (list.size() - 1);

\_QUADPTR\_ quad;

string name;

CString msg, msg2;

for (itr = begin; itr != end; ++itr)

{

quad.q = \*itr;

name = quad.q->what();

area = quad.q->area ();

shape = quadMap[name];

switch (shape)

{

case trap:

msg.Format(formatString[2], quad.t->Width(), quad.t->Width2(), quad.t->Height());

break;

case diam:

msg.Format(formatString[1], quad.p->Width(), quad.p->Height());

break;

default:

msg.Format(formatString[0], quad.p->Width(), quad.p->Height());

}

if (isLearning) msg2 = \_T("？");

else msg2.Format (\_T("%.1lf"), area);

msg += msg2;

for (int i = 0; i < 4; ++i)

{

vert[i].x = static\_cast<int>(xc + (ad[shape][i].cx \* ratio));

vert[i].y = static\_cast<int>(yc + (ad[shape][i].cy \* ratio));

}

int tx, ty;

if (isLearning)

{

tx = xc - sw;

ty = yc + sh + 8;

}

else

{

CBrush br(RGB(240, 240, 240));

CBrush \*oldbr = reinterpret\_cast<CBrush \*>(pDC->SelectObject(&br));

CPen \*oldpen = reinterpret\_cast<CPen \*>(pDC->SelectStockObject(NULL\_PEN));

pDC->Rectangle (xc - 80, yc - 50, xc + 600, yc + 50);

pDC->SelectObject(oldbr);

pDC->SelectObject(oldpen);

tx = xc + yOffset - 24;

ty = yc - 8;

yc += yOffset;

}

pDC->Polygon (vert, 4);

pDC->SetBkMode (TRANSPARENT);

pDC->TextOutW (tx, ty, msg);

}

pDC->SelectObject (oldBrush);

}

void Canvas::drawQuad(const Quadrangle \*quad)

{

if (isLearning) pCanvas->Invalidate ();

}

// Canvas 诊断

#ifdef \_DEBUG

void Canvas::AssertValid() const

{

CScrollView::AssertValid();

}

#ifndef \_WIN32\_WCE

void Canvas::Dump(CDumpContext& dc) const

{

CScrollView::Dump(dc);

}

#endif

#endif //\_DEBUG

bool Canvas::toggleMode(void)

{

return isLearning = !isLearning;

}

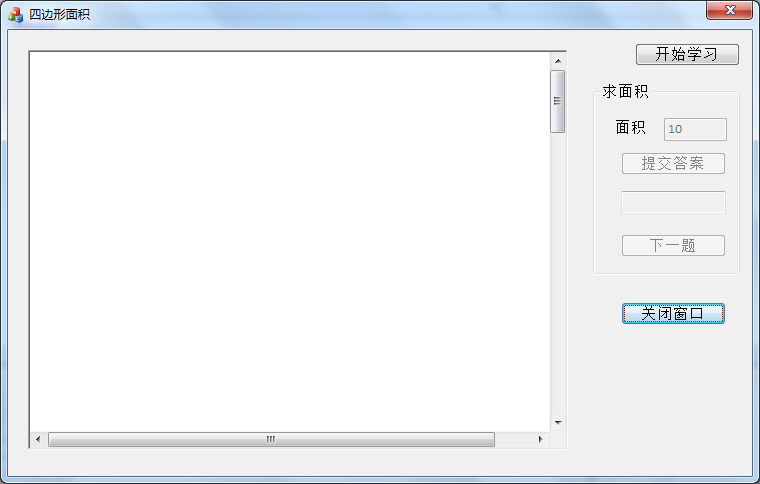
//experimentDlg.cpp

//仅改变全局说明

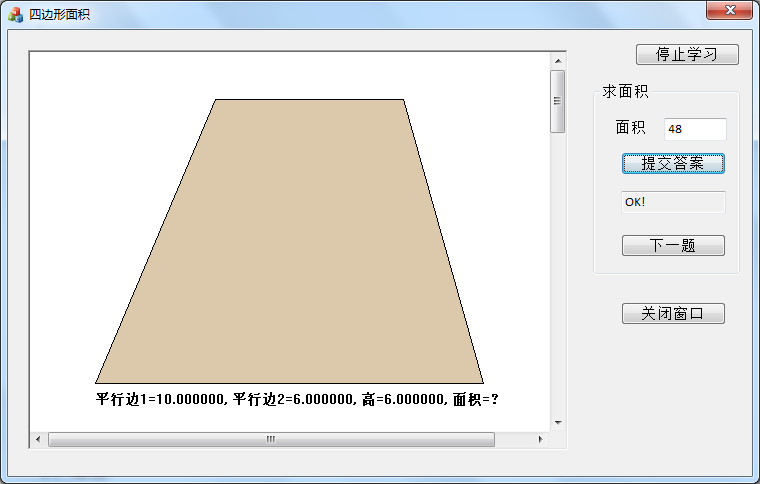
List <qPtr> list;

**过程截图：**

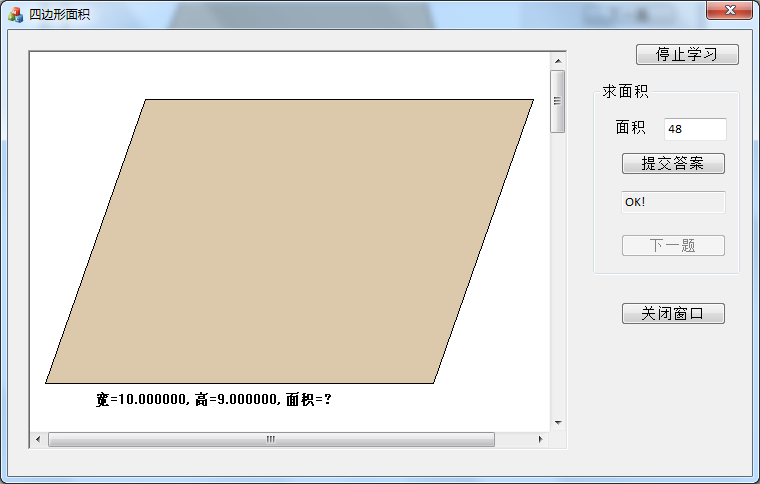
1. 运行程序，开始学习



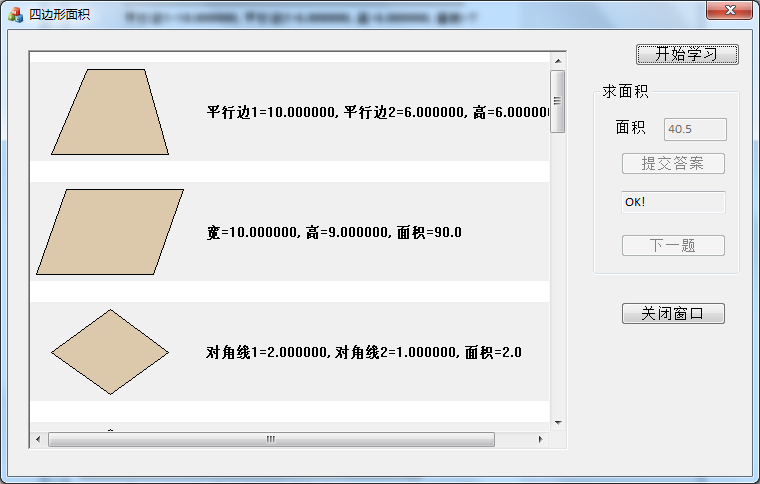
1. 输入答案，判断结果



1. 进入下一题的学习



1. 进入复习模式



1. 点击关闭窗口，退出

**九、实验数据及结果分析：**

实验数据：如过程图展示。

结果分析：经过测试，各项功能正常。能够随机产生形状、大小不同的四边形，能正确处理输入的结果，能够正确进入复习模式，将之前所求面积的图形一一列出。

**十、实验结论：**

实验完成，实现了将list改为模板类并添加迭代器的要求，程序正确运行，得到正确的输出。

**十一、总结及心得体会：**

个人感觉本次实验是四次中比较简单的一个，首先从修改文件上来说，因为修改迭代器并不会影响很多东西，比如各种类型四边形的生成和赋值，所以工作量比较少。但是的确在修改list类的时候感觉到了模板类代表的是一类东西，由于c++的强类型属性，所以在使用时必须具体化。而对于迭代器也有了一些自己的理解，感觉就像是一个抽离出来的类的指针。

并且这次试验感觉资源很多，也因此难度也下降了不少。