МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №4

по курсу объектно-ориентированное программирование 3 семестр, 2021/22 уч. Год

Студент Абросимов Алексей Дмитриевич, группа М8О-207Б-20
Преподаватель Дорохов Евгений Павлович

Условие

Задание: Вариант 3: Динамический массив и прямоугольник
Необходимо спроектировать и запрограммировать на языке C++ шаблон класса-контейнера первого уровня, содержащий одну фигуру (колонка фигура 1), согласно вариантам задания.
Классы должны удовлетворять следующим правилам:
Требования к классам фигуры аналогичны требованиям из лабораторной работы №1;
Требования к классу контейнера аналогичны требованиям из лабораторной работы №2;
Шаблон класса-контейнера должен содержать объекты используя std::shared_ptr<…>.

Описание программы

Исходный код лежит в 10 файликах:
1.main.cpp — основная программа, направленная на взамодействие с пользователем.
2.rectangle.h — описание класса прямоугольник, который наследуется от фигуры.
3.rectangle.cpp — описание методов прямоугольника
4.tvector.h — описание класса вектора
5.tvector.cpp — описание методов вектора
6. figure.h — описание абстрактного класса фигуры
7. rhombus.cpp — описание методов ромба
8. rhombus.h — описание класса ромба
9. trapezoid.h — описание класса трапеции
10. trapezoid.cpp — описание методов трапеции

Дневник отладки

Результат работы программы:

```
figure #1coords is Rectangle coords (1,4) (2,4) (2,4) (3,4)
figure #2coords is Rectangle coords (1,4) (2,4) (2,4) (3,4)
figure #3coords is Rectangle coords (1,4) (2,4) (2,4) (3,4)

figure #1coords is Rectangle coords (1,4) (2,4) (2,4) (3,4)
figure #2coords is Rectangle coords (1,4) (2,4) (2,4) (3,4)

figure #1coords is Rhombus coords (1,4) (2,4) (2,4) (3,4)
figure #2coords is Rhombus coords (1,4) (2,4) (2,4) (3,4)

figure #1coords is Rhombus coords (2,4) (2,4) (2,4) (3,4)
figure #2coords is Rhombus coords (2,4) (2,4) (2,4) (3,4)

Trapezoid was deleted
Rhombus was deleted
Rectangle was deleted
Trapezoid was deleted
Trapezoid was deleted
Rhombus was deleted
Rhombus was deleted
Rectangle was deleted
Rectangle was deleted
```

Недочёты

Выводы

Данная лабораторная работа позволила мне ознакомиться с шаблонами. Шаблоны — удивительная вещь, позволяющая экономить невероятное количество строк кода, т. к. один и тот же код может работать с различными типами данных. Работа с шаблонами показалась мне довольно таки простой, но при этом крайне эффективной и удобной.

Ссылка на гитхаб: https://github.com/yungalexxxey/oop_labs/tree/main/lab4

Исходный код

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
#include <iostream>
class figure {
public:
virtual void Print(std::ostream&os)=0;
virtual double Area()=0;
virtual size_t VertexesNumber()=0;
virtual bool isit()=0;
};
#endif // FIGURE_H
```

main.cpp

```

#include <iostream>
#include "rhombus.h"
#include "trapezoid.h"
#include "tvector.h"

int main()
{
TVector<Rectangle> containerrec;
TVector<Rhombus> containerrhom;
TVector<Trapezoid> containertrap;
containerrec.push_back(std::shared_ptr<Rectangle>(new Rectangle(1,2,2,3,4,4,4,4)));
containerrhom.push_back(std::shared_ptr<Rhombus>(new Rhombus(1,2,2,3,4,4,4,4)));
containertrap.push_back(std::shared_ptr<Trapezoid>(new Trapezoid(2,2,2,3,4,4,4,4)));
containerrec.push_back(std::shared_ptr<Rectangle>(new Rectangle(1,2,2,3,4,4,4,4)));
containerrhom.push_back(std::shared_ptr<Rhombus>(new Rhombus(1,2,2,3,4,4,4,4)));
containertrap.push_back(std::shared_ptr<Trapezoid>(new Trapezoid(2,2,2,3,4,4,4,4)));
containerrec.push_back(std::shared_ptr<Rectangle>(new Rectangle(1,2,2,3,4,4,4,4)));
containerrhom.push_back(std::shared_ptr<Rhombus>(new Rhombus(1,2,2,3,4,4,4,4)));
containertrap.push_back(std::shared_ptr<Trapezoid>(new Trapezoid(2,2,2,3,4,4,4,4)));
std::shared_ptr<Rectangle> trec;
std::shared_ptr<Rhombus> trhom;
std::shared_ptr<Trapezoid> ttrap;
std::cout<<containerrec<<std::endl;
trec=containerrec.pop_back();
ttrap=containertrap.pop_back();
trhom=containerrhom.pop_back();
// std::cout<<*trec<<std::endl;
// std::cout<<*trhom<<std::endl;
// std::cout<<*ttrap<<std::endl;
std::cout<<containerrec<<std::endl;
std::cout<< containerrhom<<std::endl;
std::cout<< containertrap<<std::endl;
return 0;
}
```

rectangle.cpp

```
#include "rectangle.h"
#include <math.h>

Rectangle::Rectangle():x1(0),y1(0),x2(1),y2(1),x3(0),y3(0),x4(0),y4(0){
```

```
}
Rectangle::Rectangle(int x1,int x2,int x3,int x4,int y1,int y2,int y3,int y4){
this->x1=x1;
this->x2=x2;
this->x3=x3;
this->x4=x4;
this->y1=y1;
this->y2=y2;
this->y3=y3;
this->y4=y4;
}
Rectangle::~Rectangle(){
std::cout<<"Rectangle was deleted\n";
}

Rectangle::Rectangle(std::istream&is){
std::cout <<"set x1 and y1:";
is >> x1 >> y1;
std::cout <<"set x2 and y2:";
is >> x2 >> y2;
std::cout <<"set x3 and y3:";
is >> x3 >> y3;
std::cout <<"set x4 and y4:";
is >> x4 >> y4;
}
void Rectangle::Print(std::ostream&os){
os << "Rectangle " << "(" <<x1<<" "<<y1<<")"<< "(" <<x2<<" "<<y2<<")"<< "(" <<x3<<" "<<y3<<")"<<
"(" <<x4<<" " <<y4<<")" <<std::endl;
}
size_t Rectangle::VertexesNumber(){
return 4;
}
bool Rectangle::isit(){
double perp;
double perp2;
perp=(x4-x1)*(x2-x1)+(y4-y1)*(y2-y1);
perp2=(x3-x4)*(x3-x2)+(y3-y4)*(y3-y2);
if((perp+perp2)==0) return true;
else return false;

}
double Rectangle::Area(){
double r1 = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
double r2 = sqrt((x2 - x3) * (x2 - x3) + (y2 - y3) * (y2 - y3));
double r3 = sqrt((x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3));
double p=(r1+r2+r3)/2;
double s= 2*sqrt((p * (p - r1) * (p - r2) * (p - r3)));
return s;
}
std::ostream& operator<<(std::ostream &out, const Rectangle &rec){
out << "Rectangle coords " <<"("<< rec.x1 << "," << rec.y1 << ")"<< " " <<"("<< rec.x2 << "," <<
rec.y2 << ")"<< " "<< "("<< rec.x3 << "," << rec.y3 << ")"<< " " << "("<<rec.x4 << "," << rec.y4 << ")\n";
return out;
}
std::istream& operator>>(std::istream &in,Rectangle &rec){
in >> rec.x1;
in >> rec.y1;
in >> rec.x2;
in >> rec.y2;
in >> rec.x3;
in >> rec.y3;
in >> rec.x4;
```

```
in >> rec.y4;
return in;
}
```

rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H
#include "figure.h"
#include <iostream>

class Rectangle:public figure{
public:
Rectangle();
Rectangle(int x1,int x2,int x3,int x4,int y1,int y2,int y3, int y4);
Rectangle(std::istream&is);
bool isit();
void Print(std::ostream&os);
size_t VertexesNumber();
double Area();
~Rectangle();
friend std::ostream &operator<<(std::ostream &out,const Rectangle &rec);
friend std::istream &operator>>(std::istream &in,Rectangle &rec);
private:
double x1;
double y1;
double x2;
double y2;
double x3;
double y3;
double x4;
double y4;
};

#endif // RECTANGLE_H
```

rhombus.cpp

```
#include "rhombus.h"
#include <math.h>
Rhombus::Rhombus():x1(0),y1(0),x2(1),y2(1),x3(0),y3(0),x4(0),y4(0){

}
Rhombus::Rhombus(int x1,int x2,int x3,int x4,int y1,int y2,int y3,int y4){
this->x1=x1;
this->x2=x2;
this->x3=x3;
this->x4=x4;
this->y1=y1;
this->y2=y2;
this->y3=y3;
this->y4=y4;
}
Rhombus::~Rhombus(){
std::cout<<"Rhombus was deleted\n";
}
Rhombus::Rhombus(std::istream&is){
std::cout <<"set x1 and y1:";
is >> x1 >> y1;
std::cout <<"set x2 and y2:";
is >> x2 >> y2;
std::cout <<"set x3 and y3:";
is >> x3 >> y3;
std::cout <<"set x4 and y4:";
is >> x4 >> y4;
}
```

```
void Rhombus::Print(std::ostream&os){
os << "Rhombus " << "(" <<x1<<" "<<y1<<")"<< "(" <<x2<<" "<<y2<<")"<< "(" <<x3<<" "<<y3<<")"<<
"(" <<x4<<" " <<y4<<")" <<std::endl;
}
size_t Rhombus::VertexesNumber(){
return 4;
}
bool Rhombus::isit(){
if((sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2))==sqrt((x2-x3)*(x2-x3)+(y2-y3)*(y2-y3)))&&(sqrt((x3-x4)*(x3-
x4)+(y3-y4)*(y3-y4))==sqrt((x1-x4)*(x1-x4)+(y1-y4)*(y1-y4)))) return true;
else return false;
}
double Rhombus::Area(){
double d1 = sqrt((x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3));
double d2 = sqrt((x2 - x4) * (x2 - x4) + (y2 - y4) * (y2 - y4));
double s=d1*d2/2;
return s;
}

std::ostream& operator<<(std::ostream &out, const Rhombus &rec){
out << "Rhombus coords " <<"("<< rec.x1 << "," << rec.y1 << ")"<< " " <<"("<< rec.x2 << "," << rec.y2
<< ")"<< " "<< "("<< rec.x3 << "," << rec.y3 << ")"<< " " << "("<<rec.x4 << "," << rec.y4 << ")\n";
return out;
}
std::istream& operator>>(std::istream &in,Rhombus &rec){
in >> rec.x1;
in >> rec.y1;
in >> rec.x2;
in >> rec.y2;
in >> rec.x3;
in >> rec.y3;
in >> rec.x4;
in >> rec.y4;
return in;
}
```

rhombus.h

```
#ifndef RHOMBUS_H
#define RHOMBUS_H
#include "figure.h"
#include <iostream>

class Rhombus:public figure{
public:
Rhombus();
Rhombus(int x1,int x2,int x3,int x4,int y1,int y2,int y3, int y4);
Rhombus(std::istream&is);
bool isit();
void Print(std::ostream&os);
size_t VertexesNumber();
double Area();
~Rhombus();
friend std::ostream &operator<<(std::ostream &out,const Rhombus &rec);
friend std::istream &operator>>(std::istream &in,Rhombus &rec);
private:
double x1;
double y1;
double x2;
double y2;
double x3;
double y3;
double x4;
double y4;
};
```

```
#endif // RHOMBUS_H
```

trapezoid.cpp

```
#include "trapezoid.h"
#include <math.h>
Trapezoid::Trapezoid():x1(0),y1(0),x2(1),y2(1),x3(0),y3(0),x4(0),y4(0){

}
Trapezoid::Trapezoid(int x1,int x2,int x3,int x4,int y1,int y2,int y3,int y4){
this->x1=x1;
this->x2=x2;
this->x3=x3;
this->x4=x4;
this->y1=y1;
this->y2=y2;
this->y3=y3;
this->y4=y4;
}
Trapezoid::Trapezoid(std::istream&is){
std::cout <<"set x1 and y1:";
is >> x1 >> y1;
std::cout <<"set x2 and y2:";
is >> x2 >> y2;
std::cout <<"set x3 and y3:";
is >> x3 >> y3;
std::cout <<"set x4 and y4:";
is >> x4 >> y4;
}
void Trapezoid::Print(std::ostream&os){
os << "Trapezoid " << "(" <<x1<<" "<<y1<<")"<< "(" <<x2<<" "<<y2<<")"<< "(" <<x3<<" "<<y3<<")"<<
"(" <<x4<<" " <<y4<<")" <<std::endl;
}
size_t Trapezoid::VertexesNumber(){
return 4;
}
Trapezoid::~Trapezoid(){
std::cout<<"Trapezoid was deleted\n";
}

bool Trapezoid::isit(){
double k=(y1-y4)/(x1-x4);
double k1=(y2-y3)/(x2-x3);
if(k==k1) return true;
else return false;
}
double Trapezoid::Area(){
double h=sqrt((y2-y1)*(y2-y1));
double os1=sqrt((x4-x1)*(x4-x1)+(y1-y4)*(y1-y4));
double os2=sqrt((x3-x2)*(x3-x2)+(y3-y2)*(y3-y2));
double s=(os1+os2)*h/2;
return s;
}
std::ostream& operator<<(std::ostream &out, const Trapezoid &rec){
out << "Trapezoid coords " <<"("<< rec.x1 << "," << rec.y1 << ")"<< " " <<"("<< rec.x2 << "," << rec.y2
<< ")"<< " "<< "("<< rec.x3 << "," << rec.y3 << ")"<< " " << "("<<rec.x4 << "," << rec.y4 << ")"<<
std::endl;
return out;
}
std::istream& operator>>(std::istream &in,Trapezoid &rec){
in >> rec.x1;
in >> rec.y1;
in >> rec.x2;
in >> rec.y2;
```

```
in >> rec.x3;
in >> rec.y3;
in >> rec.x4;
in >> rec.y4;
return in;
}
```

trapezoid.h

```
#ifndef TRAPEZOID_H
#define TRAPEZOID_H
#include "figure.h"
#include <iostream>

class Trapezoid:public figure
{
public:
Trapezoid();
Trapezoid(std::istream&is);
Trapezoid(int x1,int x2,int x3,int x4,int y1,int y2,int y3, int y4);
bool isit();
void Print(std::ostream&os);
size_t VertexesNumber();
double Area();
~Trapezoid();
friend std::ostream &operator<<(std::ostream &out,const Trapezoid &rec);
friend std::istream &operator>>(std::istream &in,Trapezoid &rec);
private:
double x1;
double y1;
double x2;
double y2;
double x3;
double y3;
double x4;
double y4;
};

#endif // TRAPEZOID_H
```

tvector.cpp

```
#include "tvector.h"
#include "figure.h"
#include "rectangle.h"
#include "rhombus.h"
#include "trapezoid.h"

template <class T>
TVector<T>::TVector():length(0),count(0)
{ }

template <class T>
int TVector<T>::size(){
return this->length;
}

template <class T>
bool TVector<T>::empty(){
if(this->length>0) return true;
else return false;
}

template <class T>
void TVector<T>::push_back(std::shared_ptr<T> newfig){
if(count==length){
```

```
length++;
count++;
std::shared_ptr<std::shared_ptr<T>[]> narr(new std::shared_ptr<T>[length]);
for(int i=0;i<length-1;i++) narr[i]=arr[i];

narr[length-1]= newfig;
//free(arr);
arr=narr;
}
else if(count<length){
arr[count]=newfig;
count++;
}
}

template <class T>
TVector<T>::~TVector(){

}

template <class T>
std::shared_ptr<T> TVector<T>::pop_back(){
std::shared_ptr<std::shared_ptr<T>[]> narr(new std::shared_ptr<T>[length]);
for(int i=0;i<count-1;i++){
narr[i]=arr[i];
}
std::shared_ptr<T> tmp=arr[count-1];
count--;
length--;
arr=narr;
return tmp;
}

template <class T>
void TVector<T>::resize(int newlength){
if(newlength==length) return;
if(newlength>length){
std::shared_ptr<std::shared_ptr<T>[]> narr(new std::shared_ptr<T>[length]);
for(int i=0;i<length;i++)
narr[i]=arr[i];
arr=narr;
length=newlength;
}
else {
std::shared_ptr<std::shared_ptr<T>[]> narr(new std::shared_ptr<T>[length]);
for(int i=0;i<newlength;i++)
narr[i]=arr[i];
arr=narr;
count=newlength;
}
}

template <class T>
void TVector<T>::clear(){
resize(1);
pop_back();
length=0;
count=0;
}

template <class T>
void TVector<T>::erase(int pos){
if(count==0)
```

```
{
std::cout<<"Container is empty"<<std::endl;
return;
}
std::shared_ptr<std::shared_ptr<T>[]> narr(new std::shared_ptr<T>[length]);
int current_index=0;
for(int i=0;i<count;i++){
if(i!=pos-1) {
narr[current_index]=arr[i];
current_index++;
}
}
count--;
length--;
arr=narr;

}

//перегрузка операций
template <class T>
std::shared_ptr<T> TVector<T>::operator[] (int i)
{
if(i >= 0 && i < this->length)
return this->arr[i];
}

template <class T>
std::ostream& operator<<(std::ostream &out, TVector<T> &cont){
for(int i=0;i<cont.count;i++){
out<<"figure #"<< i+1<<"coords is " << *cont[i];
}
return out;
}

template class TVector<Rectangle>;
template std::ostream& operator<<(std::ostream& out, TVector<Rectangle>& cont);
template class TVector<Rhombus>;
template std::ostream& operator<<(std::ostream& out, TVector<Rhombus>& cont);
template class TVector<Trapezoid>;
template std::ostream& operator<<(std::ostream& out, TVector<Trapezoid>& cont);
```

tvector.h

```
#ifndef TVECTOR_H
#define TVECTOR_H
#include "rectangle.h"
#include <memory>
#include "figure.h"

template <class T>
class TVector
{
private:
int length;
int count;
std::shared_ptr<std::shared_ptr<T>[]> arr;
public:
TVector();
~TVector();
int size();
bool empty();
void resize(int nindex);
void push_back(std::shared_ptr<T> newrec);
void erase(int pos);
std::shared_ptr<T> pop_back();
```

```
void clear();

std::shared_ptr<T> operator[] (int i) ;
template <class A>
friend std::ostream& operator<<(std::ostream &out, TVector<A> &cont);
};

#endif // TVECTOR_H
```