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

ЛАБОРАТОРНАЯ РАБОТА №2

по курсу “Объектно-ориентированное программирование”
I семестр, 2021/22 учебный год

Студент: *Абросимов Алексей Дмитриевич, группа М8О-207Б-20*
Преподаватель: *Дорохов Евгений Павлович, каф. 806*

Задание

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру (колонка фигура 1), согласно вариантам задания. Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Классы фигур должны содержать набор следующих методов:
- Перегруженный оператор ввода координат вершин фигуры из потока std::istream (>>). Он должен заменить конструктор, принимающий координаты вершин из стандартного потока.
- Перегруженный оператор вывода в поток std::ostream (<<), заменяющий метод Print из лабораторной работы 1.
- Оператор копирования (=)
- Оператор сравнения с такими же фигурами (==)
- Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке).
- Класс-контейнер должен содержать набор следующих методов:
- size()
- empty()
- operator[]
- resize()
- push_back()
- pop_back()
- erase(size_t pos)

- clear()
- operator<<
- operator>>

Вариант №3:

- Фигура : Прямоугольник
- Контейнер: Вектор (TVector)

Описание программы:

Исходный код разделён на 6 файлов:

- `figure.h` – описание класса фигуры
- `rectangle.h` – описание класса прямоугольника (наследуется от фигуры)
- `rectangle.cpp` – реализация класса прямоугольника
- `tvector.h` – описание класса квадрата (наследуется от прямоугольника)
- `tvector.cpp` – реализация класса квадрата
- `main.cpp` – основная программа

Дневник отладки:

Вывод:

Выполнение лабораторной работы позволило мне ознакомиться с основами ООП. Я создал несколько классов с наследованием и перегруженными операциями ввода и вывода.

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
```

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);
   size_t VertexesNumber();
   double Area();
   bool isit();
   friend std::ostream& operator<<(std::ostream &out, const Rectangle &rec);
   friend std::istream& operator>>(std::istream &in, Rectangle &rec);
   Rectangle& operator= (Rectangle &rec1);

private:
   double x1;
   double y1;
   double x2;
   double y2;
   double x3;
   double y3;
   double x4;
   double y4;
};
```

```
#endif // RECTANGLE_H
```

rectangle.cpp:

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


Rectangle::Rectangle():x1(0),y1(0),x2(0),y2(0),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;
};
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 << ")"<< std::endl;
   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& Rectangle::operator= (Rectangle &rec){
    this->x1=rec.x1;
    this->x2=rec.x2;
    this->x3=rec.x3;
    this->x4=rec.x4;
    this->y1=rec.y1;
    this->y2=rec.y2;
    this->y3=rec.y3;
    this->y4=rec.y4;
}

            }
```

Tvector.h;

```
#ifndef TVECTOR_H
#define TVECTOR_H
#include "rectangle.h"

class TVector
{
private:
    int length;
    int count;
    Rectangle *arr;
public:
    TVector();
    TVector(const TVector& other);
    void InsertLast(Rectangle &newrec);
    const Rectangle Last();
    int Length();
    bool Empty();
    void Resize(int nindex);
    void Remove(int pos);
    void Clear();

    Rectangle& operator[] (int i) ;
    friend std::ostream& operator<<(std::ostream &out, TVector &cont);
};

#endif // TVECTOR_H
```

Tvector.cpp;

```
#include "tvector.h"
#include "rectangle.h"
TVector::TVector():length(0),count(0)
{ }

int TVector::Length(){
    return this->length;
}
bool TVector::Empty(){
    if(this->length>0) return true;
```

```
    else return false;
}

void TVector::Resize(int newlength){
    std::cout<<length<< " " << count<<std::endl;
    if(newlength==length) return;
    if(newlength>length){
        Rectangle *narr=new Rectangle[newlength];
        for(int i=0;i<length;i++)
            narr[i]=arr[i];
        arr=narr;
        length=newlength;
    }
    else {
        Rectangle *narr=new Rectangle[newlength];
        for(int i=0;i<newlength;i++)
            narr[i]=arr[i];
        arr=narr;
        count=newlength;
    }
    std::cout<<length<< " " << count<<std::endl;
}
void TVector::InsertLast(Rectangle &newrec){
    if(count==length){
        length++;
        count++;
        Rectangle *narr=new Rectangle[length];
        for(int i=0;i<length-1;i++) narr[i]=arr[i];
        narr[length-1]=newrec;
        //free(arr);
        arr=narr;
    }
    else if(count<length){
        arr[count]=newrec;
        count++;
    }
}

void TVector::Clear(){
    free(arr);
    length=0;
    count=0;
}
const Rectangle TVector::Last(){
        Rectangle *narr=new Rectangle[length];
        for(int i=0;i<count-1;i++){
                narr[i]=arr[i];
        }
        Rectangle tmp=arr[count-1];
        count--;
        length--;
        arr=narr;
        return tmp;
}
```

```
void TVector::Remove(int pos){
if(count==0)
{
   std::cout<<"Container is empty"<<std::endl;
   return;
}
   Rectangle *narr=new Rectangle[length-1];
   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
Rectangle& TVector::operator[] (int i)
{
   if(i >= 0 && i < this->length)
      return this->arr[i];
}


std::ostream& operator<<(std::ostream &out,  TVector &cont){
   for(int i=0;i<cont.count;i++){
      out<<"Rectangle #"<< i+1<<"coords is " << cont[i];
   }
   return out;
}
```

main.cpp

```
#include <iostream>
#include "rectangle.cpp"
#include "rectangle.h"
#include "figure.h"
#include "tvector.h"
int main()
{
   TVector container;
   Rectangle rec1(1,2,3,4,5,6,7,1);
   Rectangle rec2(1,2,3,4,5,6,7,2);
   Rectangle rec3(1,2,3,4,5,6,7,3);
   Rectangle rec4(1,2,3,4,5,6,7,4);
   container.InsertLast(rec1);
   container.InsertLast(rec2);
   container.InsertLast(rec3);
   container.InsertLast(rec4);
   std::cout<<container;
```

```
    std::cout<< container.Length()<<std::endl;
    container.Remove(2);
    std::cout<<container.Last()<<std::endl;
    std::cout<<container;
    return 0;
}
```