

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

ЛАБОРАТОРНАЯ РАБОТА №4
по курсу объектно-ориентированное программирование I семестр, 2021/22
уч. год

Студент Абаровский Олег Александрович, группа М8О-207Б-20

Преподаватель Дорохов Евгений Павлович

Условие

Задание: Вариант 1: Динамический массив. 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:

1. Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
2. 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 (`==`)
3. Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке).
4. Класс-контейнер должен содержать набор методов согласно варианту задания.

Описание программы

Исходный код лежит в 8 файлах:

1. `src/main.cpp`: основная программа, взаимодействие с пользователем посредством команд из меню
2. `include/figure.h`: описание абстрактного класса фигур
3. `include/point.h`: описание класса точки
4. `include/triangle.h`: описание класса треугольника, наследующегося от `figures`
5. `include/tvector.h`: описание класса контейнера - динамического массива
6. `include/point.cpp`: реализация класса точки
7. `include/triangle.cpp`: реализация класса треугольника, наследующегося от `figures`
8. `include/tvector.cpp`: реализация класса контейнера - динамического массива

Дневник отладки

18.11 Недочёт: в списке методов нет операции удаления конкретного элемента вектора. Решение: создать метод удаления конкретного элемента вектора, а метод RemoveLast() реализовать на его основе. В интерактивном меню реализовать команду удаления конкретного элемента.

Недочёты

Выводы

Работа была полезной, в ходе её выполнения я получил опыт реализации динамической структуры данных методами объектно-ориентированного программирования. Возможность создавать векторы, очереди, стеки из собственных объектов кажется мне интересной. Этому можно легко найти применение на практике, например, при решении геометрических задач, содержащих большое и(или) изменяющееся количество фигур.

Исходный код

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include <iostream>
#include "point.h"

class Figure {
public:
    virtual size_t VertexesNumber() = 0;
    virtual double Area() = 0;
    virtual void Print(std::ostream &os) = 0;
};

#endif // FIGURE_H
```

point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point {
public:
    Point();
    Point(std::istream &is);
    Point(double x, double y);

    double dist(Point& other);

    friend std::istream& operator>>(std::istream& is, Point& p);
    friend std::ostream& operator<<(std::ostream& os, Point& p);
    Point &operator=(const Point &p);

private:
    double x_;
    double y_;
};

#endif // POINT_H
```

point.cpp

```
#include "point.h"

#include <cmath>

Point::Point() : x_(0.0), y_(0.0) {}

Point::Point(double x, double y) : x_(x), y_(y) {}

Point::Point(std::istream &is) {
    is >> x_ >> y_;
}

double Point::dist(Point& other) {
    double dx = (other.x_ - x_);
    double dy = (other.y_ - y_);
    return std::sqrt(dx*dx + dy*dy);
}

std::istream& operator>>(std::istream& is, Point& p) {
    is >> p.x_ >> p.y_;
    return is;
}

std::ostream& operator<<(std::ostream& os, Point& p) {
    os << "(" << p.x_ << ", " << p.y_ << ")";
    return os;
}

Point &Point::operator=(const Point &p) {
    this->x_ = p.x_;
    this->y_ = p.y_;
    return *this;
}
```

main.cpp

```
#include <iostream>
#include "triangle.h"
#include "tvector.h"

using namespace std;

int main()
{
    cout << "Comands:" << endl;
    cout << "a - add new triangle (a [input])" << endl;
    cout << "d - erase triangle by index (d [idx])" << endl;
    cout << "s - set triangle by index (s [idx] [input])" << endl;
    cout << "p - print all containing triangles (p)" << endl;
    cout << "q - quit (q)" << endl;
    char f = 1;
    TVector *vect = new TVector();
    char cmd;
    while(f)
    {
        cout << "> ";
        cin >> cmd;
        switch(cmd)
        {
            case 'a':
            {
                vect->InsertLast(Triangle(cin));
                break;
            }
            case 'd':
            {
                int di;
                cin >> di;
                vect->Erase(di);
                break;
            }
            case 's':
            {
                int si;
                cin >> si;
                Triangle csq(cin);
                (*vect)[si] = csq;
            }
        }
    }
}
```

```

        break;
    }
    case 'p':
    {
        cout << *vect << endl;
        break;
    }
    case 'q':
    {
        f = 0;
        break;
    }
    default:
        cout << "wrong input" << endl;
    }
}
delete vect;
}

```

triangle.h

```
#ifndef TRIANGLE_H
#define TRIANGLE_H
#include "figure.h"

class Triangle : public Figure {
private:
    Point a_, b_, c_;
public:
    Triangle();
    Triangle(const Triangle &triangle);
    Triangle(std::istream &is);
    size_t VertexesNumber();
    double Area();
    void Print(std::ostream &os);
};

#endif //TRIANGLE_H
```


triangle.cpp

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

Triangle::Triangle() : a_(0, 0), b_(0, 0), c_(0, 0) {}

Triangle::Triangle(const Triangle &triangle) {
    this->a_ = triangle.a_;
    this->b_ = triangle.b_;
    this->c_ = triangle.c_;
}

Triangle::Triangle(std::istream &is) {
    std::cin >> a_ >> b_ >> c_;
}

size_t Triangle::VertexesNumber() {
    return 3;
}

double Triangle::Area() {
    double a = a_.dist(b_);
    double b = b_.dist(c_);
    double c = c_.dist(a_);
    double p = (a + b + c) / 2;
    return sqrt(p * (p - a) * (p - b) * (p - c));
}

void Triangle::Print(std::ostream &os) {
    std::cout << "Triangle " << a_ << b_ << c_ << std::endl;
}
```

tvector.h

```
//TVECTOR.H
#ifndef TVECTOR_H
#define TVECTOR_H
#define et_tvector Triangle

#include <iostream>
#include "triangle.h"

class TVector
{
    public:
        TVector();
        TVector(const TVector& other);
        void Erase(int pos);
        void InsertLast(const et_tvector& elem);
        void RemoveLast();
        const et_tvector& Last();
        et_tvector& operator[] (const size_t idx);
        bool Empty();
        size_t Length();
        friend std::ostream& operator<<(std::ostream& os, TVector& obj);
        void Clear();
        ~TVector();
    private:
        void resize(int newsize);
        et_tvector *vals;
        int len;
        int rLen;
};

#endif
```

tvector.cpp

```
//TVECTOR.CPP
#include "tvector.h"
#include <iostream>
#include <cstring>

TVector::TVector()
{
    vals = NULL;
    len = 0;
    rLen = 0;
}

TVector::TVector(const TVector& other)
{
    len = other.len;
    rLen = other.rLen;
    vals = (et_tvector*)malloc(sizeof(et_tvector)*len);
    memcpy((void*)vals, (void*)other.vals, sizeof(et_tvector)*len);
}

void TVector::Erase(int pos)
{
    if(len == 1)
    {
        Clear();
        return;
    }
    memmove((void*)&(vals[pos]), (void*)&(vals[pos+1]), sizeof(et_tvector)*(len-pos-1));
    len--;
    if(len==rLen>>1)
        resize(len);
}

void TVector::InsertLast(const et_tvector& elem)
{
    if(rLen)
    {
        if(len>=rLen)
        {
            rLen<=1;
            resize(rLen);
        }
    }
}
```

```

        }
    }
    else
    {
        rLen=1;
        resize(rLen);
    }
    vals[len] = elem;
    len++;
}

void TVector::RemoveLast()
{
    Erase(len-1);
}

const et_tvector& TVector::Last()
{
    return vals[len-1];
}

et_tvector& TVector::operator[](const size_t idx)
{
    return vals[idx];
}

bool TVector::Empty()
{
    return len == 0;
}

size_t TVector::Length()
{
    return len;
}

std::ostream& operator<<(std::ostream& os, TVector& obj)
{
    os << '[';
    for(int i = 0; i < obj.len; i++)
    {
        os << ((Triangle)obj.vals[i]).Area();
    }
}

```

```

        if(i != obj.len - 1)
            os << " ";
    }
    os << ']';
    return os;
}

void TVector::Clear()
{
    if(!Empty())
    {
        free(vals);
        vals = NULL;
        len = 0;
        rlen = 0;
    }
}

void TVector::resize(int newsize)
{
    vals = (et_tvector*)realloc((void*)vals, sizeof(et_tvector)*newsize);
}

TVector::~TVector()
{
    Clear();
}

```