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

ЛАБОРАТОРНАЯ РАБОТА №4

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студентка: *Ивченко Анна Владимировна*

Группа: *М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

Москва, 2021

Задание:

Спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру. Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 2.
- Классы фигур должны содержать набор следующих методов:
  - Перегруженный оператор ввода координат вершин фигуры из потока std::istream (>>)
  - Перегруженный оператор вывода в поток std::ostream (<<)
  - Оператор копирования (=)
  - Оператор сравнения с такими же фигурами (==)
- Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке).
- Класс-контейнер должен иметь функции соответствующие варианту.

Нельзя использовать:

- Стандартные контейнеры std.
- Шаблоны (template).
- Различные варианты умных указателей (shared_ptr, weak_ptr).

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

Вариант №28:

- Фигура: Трапеция
- Контейнер: Очередь

Описание программы:

Программа состоит из 10 файлов

- `main.cpp` – основная программа
- `point.h` – описание класса точки
- `point.cpp` – реализация класса точки
- `tqueue_item.h` – описание элемента очереди
- `tqueue_item.cpp` – реализация элемента очереди
- `tqueue.h` – описание очереди
- `tqueue.cpp` – реализация очереди

- trapezoid.h – описание класса трапеции
- trapezoid.cpp – реализация класса трапеции

Дневник отладки:

Тестирование:

```
Default queue created
Enter n: 3
Enter points: 1 1 2 3 4 5 6 7 8
Trapezoid: (1, 1) (2, 3) (4, 5) (6, 7)

Trapezoid copied
Queue item: created
Added one trapezoid to tail. Coordinates: Trapezoid: (1, 1) (2, 3) (4, 5) (6, 0)
. Area = 2.40669
Queue: => 2.40669 =>
Length: 1
Enter points: 2 3 1 1 4 6 4 2
Trapezoid: (8, 2) (3, 1) (1, 4) (6, 4)

Trapezoid copied
Queue item: created
Added one trapezoid to tail. Coordinates: Trapezoid: (8, 2) (3, 1) (1, 4) (6, 0)
. Area = 16.064
Queue: => 16.064 2.40669 =>
Length: 2
Enter points: 2 3 4 5 6 7 1 5
Trapezoid: (2, 2) (3, 4) (5, 6) (7, 1)

Trapezoid copied
Queue item: created
Added one trapezoid to tail. Coordinates: Trapezoid: (2, 2) (3, 4) (5, 6) (7, 0)
. Area = nan
Queue: => nan 16.064 2.40669 =>
Length: 3
Trapezoid copied
Trapezoid: (1, 1) (2, 3) (4, 5) (6, 0)
```

Вывод:

В ходе проделанной работы был реализован контейнер, работа оказалось очень интересной, потому что похожая работа была проделана на 1 курсе на языке Си и сделать лабораторную по принципам ООП и на C++ помогла по другому взглянуть на знакомые уже вещи.

Исходный код:

main.cpp:

```
#include <iostream>
#include "trapezoid.h"
```

```
#include "tqueue.h"
int main(int argc, char** argv) {
	TQueue queue;
	Trapezoid tr;
	std::cout << "Enter n: ";
	int n; std::cin >> n;

	for (int i = 0; i < n; i++) {
		std::cin >> tr;
		std::cout << tr << std::endl;
		queue.Push(tr);
		std::cout << queue;
		std::cout << std::endl;
		std::cout << "Length: " << queue.Length() << std::endl;
	}
	Trapezoid tr1;
	tr1 = queue.Top();
	std::cout << tr1;
	return 0;
}
```

figure.h:

```
#ifndef FIGURE_H
#define FIGURE_H
#include <iostream>
class Figure {
public:
   virtual size_t VertexesNumber() = 0;
   virtual double Area() = 0;
   virtual ~Figure() {};
};
#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);
   void SetX(double x);
   void SetY(double y);
   double GetX();
   double GetY();
   friend std::istream& operator>>(std::istream& is, Point& p);
   friend std::ostream& operator<<(std::ostream& os, Point& p);
   friend std::ostream& operator<<(std::ostream& os, const Point& p);
public:
   double x_;
   double y_;
};
```

```
#endif // POINT_H
```

point.cpp:

```
#include "point.h"
#include <iostream>
#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_;
}
void Point::SetX(double x) {
        this->x_ = x;
}
void Point::SetY(double y) {
        this->y_ = y;
}
double Point::GetX() {
        return this->x_;
}
double Point::GetY() {
        return this->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;
}
std::ostream& operator<<(std::ostream& os, const Point& p) {
        os << "(" << p.x_ << ", " << p.y_ << ")";
        return os;
}
```

trapezoid.h:

```
#ifndef TRAPEZOID_H
#define TRAPEZOID_H
#include "figure.h"
#include <iostream>
#include "point.h"
class Trapezoid : public Figure {
public:
   Trapezoid();
   Trapezoid(double a, double b, double c, double d);
   Trapezoid(const Trapezoid& other);
   friend std::istream& operator>>(std::istream& is, Trapezoid& obj);
   friend std::ostream& operator<<(std::ostream& os, const Trapezoid& obj);
```

```
    Trapezoid& operator=(const Trapezoid& right);
    bool operator==(const Trapezoid& right);
    virtual ~Trapezoid();
    size_t VertexesNumber();
    double Area();
public:
    double len_ab, len_bc, len_cd, len_da;
    Point a_, b_, c_, d_;
};
#endif // TRAPEZOID_H
```

trapezoid.cpp:

```
#include "trapezoid.h"
#include <cmath>
Trapezoid::Trapezoid()
    : len_ab(0.0),
      len_bc(0.0),
      len_cd(0.0),
      len_da(0.0) {
}

Trapezoid::Trapezoid(double ab, double bc, double cd, double da)
    : len_ab(ab),
      len_bc(bc),
      len_cd(cd),
      len_da(da) {
}
Trapezoid::Trapezoid(const Trapezoid& other)
    : Trapezoid(other.len_ab, other.len_bc, other.len_cd, other.len_da) {
}
std::istream& operator>>(std::istream& is, Trapezoid& obj) {
    std::cout << "Enter points: ";
    is >> obj.a_;
    is >> obj.b_;
    is >> obj.c_;
    is >> obj.d_;
    obj.len_ab = obj.a_.dist(obj.b_);
    obj.len_bc = obj.b_.dist(obj.c_);
    obj.len_cd = obj.c_.dist(obj.d_);
    obj.len_da = obj.d_.dist(obj.a_);
    return is;
}
std::ostream& operator<<(std::ostream& os, const Trapezoid& obj) {
    std::cout << "Trapezoid: ";
    os << obj.a_; std::cout << " ";
    os << obj.b_; std::cout << " ";
    os << obj.c_; std::cout << " ";
    os << obj.d_; std::cout << std::endl;
    return os;
}
Trapezoid& Trapezoid::operator=(const Trapezoid& other) {
    if (this == &other)
        return *this;
    len_ab = other.len_ab;
    len_bc = other.len_bc;
    len_cd = other.len_cd;
    len_da = other.len_da;
```

```
    a_.x_ = other.a_.x_;
    a_.y_ = other.a_.y_;
    b_.x_ = other.b_.x_;
    b_.y_ = other.b_.y_;
    c_.x_ = other.c_.x_;
    c_.y_ = other.c_.y_;
    d_.x_ = other.d_.x_;
    d_.x_ = other.d_.x_;
    std::cout << "Trapezoid copied" << std::endl;
        return *this;
}

bool Trapezoid::operator==(const Trapezoid& other) {
    if (this->len_ab == other.len_ab &&
        this->len_bc == other.len_bc &&
        this->len_cd == other.len_cd &&
        this->len_da == other.len_da) {
        std::cout << "Trapezoids are equal" << std::endl;
        return 1;
    } else {
        std::cout << "Trapezoids are not equal" << std::endl;
        return 0;
    }
}
size_t Trapezoid::VertexesNumber() {
    return 4;
}
double Trapezoid::Area() {
    double p = (len_ab + len_bc + len_cd + len_da) / 2;
    return (len_bc + len_da) *
           std::sqrt((p - len_bc) *
                     (p - len_da) *
                     (p - len_da - len_ab) *
                     (p - len_da - len_cd)) /
           std::abs(len_bc - len_da);
}

Trapezoid::~Trapezoid() {

    std::cout << "Trapezoid deleted" << std::endl;
}
```

tqueue_item.h:

```
#ifndef TQUEUE_ITEM_H
#define TQUEUE_ITEM_H
#include "trapezoid.h"
class TQueueItem {
public:
  TQueueItem();
  TQueueItem(const Trapezoid& trapezoid);
  TQueueItem(const TQueueItem& other);
  TQueueItem* SetNext(TQueueItem* next);
  TQueueItem* GetNext();
  Trapezoid GetTrapezoid() const;
  friend std::ostream& operator<<(std::ostream& os, const TQueueItem& obj);

  virtual ~TQueueItem();
```

```
public:
  Trapezoid trapezoid;
  TQueueItem *next;
};

#endif // TQUEUE_ITEM_H
```

tqueue_item.cpp:

```
#include "trapezoid.h"

#include <cmath>

Trapezoid::Trapezoid()

    : len_ab(0.0),

      len_bc(0.0),

      len_cd(0.0),

      len_da(0.0) {

}

Trapezoid::Trapezoid(double ab, double bc, double cd, double da)

    : len_ab(ab),

      len_bc(bc),

      len_cd(cd),

      len_da(da) {

}

Trapezoid::Trapezoid(const Trapezoid& other)

    : Trapezoid(other.len_ab, other.len_bc, other.len_cd, other.len_da) {

}

std::istream& operator>>(std::istream& is, Trapezoid& obj) {

    std::cout << "Enter points: ";

    is >> obj.a_;

    is >> obj.b_;

    is >> obj.c_;

    is >> obj.d_;

    obj.len_ab = obj.a_.dist(obj.b_);

    obj.len_bc = obj.b_.dist(obj.c_);

    obj.len_cd = obj.c_.dist(obj.d_);
```

```
    obj.len_da = obj.d_.dist(obj.a_);
    return is;
}
std::ostream& operator<<(std::ostream& os, const Trapezoid& obj) {
    std::cout << "Trapezoid: ";
    os << obj.a_; std::cout << " ";
    os << obj.b_; std::cout << " ";
    os << obj.c_; std::cout << " ";
    os << obj.d_; std::cout << std::endl;
    return os;
}
Trapezoid& Trapezoid::operator=(const Trapezoid& other) {
    if (this == &other)
        return *this;
    len_ab = other.len_ab;
    len_bc = other.len_bc;
    len_cd = other.len_cd;
    len_da = other.len_da;
    a_.x_ = other.a_.x_;
    a_.y_ = other.a_.y_;
    b_.x_ = other.b_.x_;
    b_.y_ = other.b_.y_;
    c_.x_ = other.c_.x_;
    c_.y_ = other.c_.y_;
    d_.x_ = other.d_.x_;
    d_.x_ = other.d_.x_;
    std::cout << "Trapezoid copied" << std::endl;
    return *this;
}
bool Trapezoid::operator==(const Trapezoid& other) {
    if (this->len_ab == other.len_ab &&
```

```
        this->len_bc == other.len_bc &&
        this->len_cd == other.len_cd &&
        this->len_da == other.len_da) {
        std::cout << "Trapezoids are equal" << std::endl;
        return 1;
    } else {
        std::cout << "Trapezoids are not equal" << std::endl;
        return 0;
    }
}
size_t Trapezoid::VertexesNumber() {
    return 4;
}
double Trapezoid::Area() {
    double p = (len_ab + len_bc + len_cd + len_da) / 2;
    return (len_bc + len_da) *
           std::sqrt((p - len_bc) *
                     (p - len_da) *
                     (p - len_da - len_ab) *
                     (p - len_da - len_cd)) /
           std::abs(len_bc - len_da);
}

Trapezoid::~Trapezoid() {

    std::cout << "Trapezoid deleted" << std::endl;
}
```

tqueue.h:

```
#ifndef TQUEUE_H
#define TQUEUE_H
#include "trapezoid.h"
#include "tqueue_item.h"
class TQueue {
```

```
public:
    TQueue();
    TQueue(const TQueue& other);
    void Push(const Trapezoid& trapezoid);
    void Pop();
    Trapezoid& Top();
    bool Empty();
    size_t Length();
    friend std::ostream& operator<<(std::ostream& os, const TQueue& queue);
    void Clear();
    virtual ~TQueue();
private:
    TQueueItem *head, *tail;
};
#endif // TQUEUE_H
```

tqueue.cpp:

```
#include "tqueue.h"
#include <vector>
TQueue::TQueue() : head(nullptr), tail(nullptr) {
    std::cout << "Default queue created" << std::endl;
}
TQueue::TQueue(const TQueue& other) {
    head = other.head;
    tail = other.tail;
}
void TQueue::Push(const Trapezoid& trapezoid) {
    TQueueItem* other = new TQueueItem(trapezoid);
    if (tail == nullptr) {
        head = tail = other;
        std::cout << "Added one trapezoid to tail. " << "Coordinates: " << other->trapezoid << ". Area = " << other-
>trapezoid.Area() << std::endl;
        return;
    }
    tail->next = other;
    tail = other;
    tail->next = nullptr;
    std::cout << "Added one trapezoid to tail. " << "Coordinates: " << other->trapezoid << ". Area = " << other-
>trapezoid.Area() << std::endl;
}
void TQueue::Pop() {
    if (head == nullptr)
        return;
    std::cout << "Removed one trapezoid " << head->trapezoid << " from head" << std::endl;
    TQueueItem *temp = head;
    head = head->next;
    delete temp;
    if (head == nullptr)
        tail = nullptr;
}
Trapezoid& TQueue::Top() {
    Trapezoid& out = head->trapezoid;
    return out;
}
bool TQueue::Empty() {
    return (head == nullptr) && (tail == nullptr);
}
```

```
size_t TQueue::Length() {
    if (head == nullptr && tail == nullptr)
        return 0;
    TQueueItem *temp = head;
    int counter = 0;
    while (temp != tail->GetNext()) {
        temp = temp->GetNext();
        counter++;
    }
    return counter;
}
std::ostream& operator<<(std::ostream& os, const TQueue& queue) {
    TQueueItem *temp = queue.head;
    std::vector<TQueueItem *> v;
    os << "Queue: ";
    os << "=> ";
    while (temp != nullptr) {
        v.push_back(temp);
        //os << *temp << " ";
        temp = temp->GetNext();
    }
    for (int i = v.size() - 1; i >= 0; --i)
        os << *v[i] << " ";
    os << "=>";
    return os;
}
void TQueue::Clear() {
    for (int i = 0; i < this->Length(); i++) {
        this->Pop();
    }
    std::cout << "Queue was cleared but still exist" << std::endl;
}

TQueue::~TQueue() {
    std::cout << "Queue was deleted" << std::endl;
}
```