

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

ЛАБОРАТОРНАЯ РАБОТА №6

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студентка: Ивченко Анна Владимировна

Группа: М8О-208Б-20

Преподаватель: Дорохов Евгений Павлович, каф. 806

Москва, 2021

Задание:

Дополнить класс-контейнер из лабораторной работы №5 шаблоном типа данных.

Вариант №28:

- Фигуры: Трапеция
- Контейнер: Очередь

Описание программы:

Программа состоит из 10 файлов:

- `figure.h` – описание класса фигуры
- `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
- `main.cpp` – основная программа
- `trapezoid.h` – описание класса трапеции, наследуется от фигуры
- `trapezoid.cpp` – реализация класса трапеции

Дневник отладки:

В ходе работы ошибок не возникало

Тестирование:

Default queue created

Queue: => =>

Enter n: 2

Enter points: 1 1 2 3 4 5 6 7

Trapezoid: (1, 1) (2, 3) (4, 5) (6, 7)

Queue item: created

Added one trapezoid to tail. Coordinates: Trapezoid: (1, 1) (2, 3) (4, 5) (6, 7)

. Area = 2.40669

Queue: => 2.40669 =>

Length: 1

Enter points: 1 2 3 5 4 3 2 7

Trapezoid: (1, 2) (3, 5) (4, 3) (2, 7)

Queue item: created

Added one trapezoid to tail. Coordinates: Trapezoid: (1, 2) (3, 5) (4, 3) (2, 7)

. Area = 13.2013

Queue: => 13.2013 2.40669 =>

Length: 2

Queue copied

Queue: Queue: => 13.2013 2.40669 =>

Queue2: Queue: => 13.2013 2.40669 =>

Вывод:

В ходе проделанной работы я познакомилась с шаблонами. Шаблоны позволяют определить операции класса или функции и предоставить пользователю указание конкретных типов, с которыми должны работать эти операции. Они играют огромную роль в языке C++. Эта лабораторная работа, на мой взгляд, является очень важной, ведь многое держится на шаблонах.

Исходный код:

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
```

main.cpp:

```
#include <iostream>
#include "tqueue.h"
int main(int argc, char** argv) {
```

```

TQueue<Trapezoid> queue;
std::shared_ptr<Trapezoid> tr(new Trapezoid(1, 2, 3, 4));
std::cout << queue << std::endl;
std::shared_ptr<Trapezoid> t;
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(std::shared_ptr<Trapezoid>(new Trapezoid(*tr)));
    std::cout << queue;
    std::cout << std::endl;
    std::cout << "Length: " << queue.Length() << std::endl;
}

TQueue<Trapezoid> queue2 = queue;
std::cout << "Queue: " << queue << std::endl;
std::cout << "Queue2: " << queue2 << std::endl;
return 0;
}

```

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(std::istream &is);
    Trapezoid(const Trapezoid& other);
    virtual ~Trapezoid();
    size_t VertexesNumber();
    double Area();
    void Print(std::ostream& os);
public:
    double len_ab, len_bc, len_cd, len_da;
};
#endif // TRAPEZOID_H

```

trapezoid.cpp:

```
#include "trapezoid.h"
#include <cmath>
static Point a_o, b_o, c_o, d_o;
Trapezoid::Trapezoid()
: len_ab(0.0),
  len_bc(0.0),
  len_cd(0.0),
  len_da(0.0) {
    std::cout << "Default Trapezoid created" << std::endl;
}
Trapezoid::Trapezoid(double ab, double bc, double cd, double da)
: len_ab(ab),
  len_bc(bc),
  len_cd(cd),
  len_da(da) {
    std::cout << "Trapezoid created" << std::endl;
}
Trapezoid::Trapezoid(std::istream &is) {
    std::cout << "Enter Data:" << std::endl;
    is >> a_o >> b_o >> c_o >> d_o;
    len_ab = a_o.dist(b_o);
    len_bc = b_o.dist(c_o);
    len_cd = c_o.dist(d_o);
    len_da = d_o.dist(a_o);
    std::cout << "Trapezoid created via istream" << std::endl;
}
Trapezoid::Trapezoid(const Trapezoid& other)
: Trapezoid(other.len_ab, other.len_bc, other.len_cd, other.len_da) {
    std::cout << "Made copy of Trapezoid" << std::endl;
}
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);
}
void Trapezoid::Print(std::ostream& os) {
    std::cout << "Trapezoid: ";
    os << a_o; std::cout << " ";
    os << b_o; std::cout << " ";
    os << c_o; std::cout << " ";
    os << d_o; std::cout << std::endl;
}
Trapezoid::~Trapezoid() {
    std::cout << "Trapezoid deleted" << std::endl;
}
```

TQueueItem.h:

```

#ifndef TQUEUE_ITEM_H
#define TQUEUE_ITEM_H
#include <memory>
#include "trapezoid.h"
template<typename T> class TQueueItem {
public:
    TQueueItem(const std::shared_ptr<T>& trapezoid);
    TQueueItem(const TQueueItem& other);
    std::shared_ptr<TQueueItem<T>> SetNext(std::shared_ptr<TQueueItem> &next);
    std::shared_ptr<TQueueItem<T>> GetNext();
    std::shared_ptr<T> GetTrapezoid() const;
    template<typename A> friend std::ostream& operator<<(std::ostream& os, const TQueueItem<A>& obj);
    virtual ~TQueueItem();
public:
    std::shared_ptr<T> item;
    std::shared_ptr<TQueueItem<T>> next;
};
#endif // TQUEUE_ITEM_H

```

TQueueItem.cpp:

```

#include "tqueue_item.h"
#include <iostream>
template <class T>
TQueueItem<T>::TQueueItem(const std::shared_ptr<T>& item) {
    this->item = item;
    this->next = nullptr;
    std::cout << "Queue item: created" << std::endl;
}

template <class T>
TQueueItem<T>::TQueueItem(const TQueueItem& other) { // maybe change to TQueueItem<T>
    this->item = other.item;
    this->next = other.next;
    std::cout << "Queue item: copied" << std::endl;
}

template <class T>
std::shared_ptr<TQueueItem<T>> TQueueItem<T>::SetNext(std::shared_ptr<TQueueItem<T>> &next)
{ /////////////// added &
    std::shared_ptr<TQueueItem<T>> old = this->next;
    this->next = next;
    return old;
}

template <class T>
std::shared_ptr<T> TQueueItem<T>::GetTrapezoid() const {
    return this->item;
}

template <class T>
std::shared_ptr<TQueueItem<T>> TQueueItem<T>::GetNext() {
    return this->next;
}

template <class T>
TQueueItem<T>::~TQueueItem() {
    std::cout << "Queue item: deleted" << std::endl;
}

template <class A>
std::ostream& operator<<(std::ostream& os, const TQueueItem<A>& obj) {

```

```

    os << obj.item->Area();
    return os;
}
template class TQueueItem<Trapezoid>;
template std::ostream& operator<<(std::ostream& os, const TQueueItem<Trapezoid>& obj);

```

tqueue.h:

```

#ifndef TQUEUE_H
#define TQUEUE_H
#include "tqueue_item.h"
#include <memory>
template <typename T> class TQueue {
public:
    TQueue();
    TQueue(const TQueue& other);
    void Push(std::shared_ptr<T> &&trapezoid);
    void Pop();
    std::shared_ptr<T>& Top();
    bool Empty();
    size_t Length();
    template <class A> friend std::ostream& operator<<(std::ostream& os, const TQueue<A>& queue);
    void Clear();
    virtual ~TQueue();
private:
    std::shared_ptr<TQueueItem<T>> head, tail;
};
#endif // TQUEUE_H

```

TQueue.cpp:

```

#include "tqueue.h"
#include <vector>
template <class T>
TQueue<T>::TQueue() : head(nullptr), tail(nullptr) {
    std::cout << "Default queue created" << std::endl;
}
template <class T>
TQueue<T>::TQueue(const TQueue<T>& other) {
    head = other.head;
    tail = other.tail;
    std::cout << "Queue copied" << std::endl;
}
template <class T>
void TQueue<T>::Push(std::shared_ptr<T> &&trapezoid) {
    std::shared_ptr<TQueueItem<T>> other(new TQueueItem<T>(trapezoid));

    if (tail == nullptr) {
        head = tail = other;
        std::cout << "Added one trapezoid to tail. " << "Coordinates: " << *other->item << ". Area = " << other->item->Area() << std::endl;
        return;
    }
    tail->SetNext(other);
    tail = other;
    tail->next = nullptr;
    std::cout << "Added one trapezoid to tail. " << "Coordinates: " << *other->item << ". Area = " << other->item->Area() << std::endl;
}

```



```

}
template <class T>
void TQueue<T>::Pop() {
    if (head == nullptr)
        return;
    std::cout << "Removed one trapezoid from head." << "Coordinates: " << *head->item << ". Area = " << head-
>item->Area() << std::endl;
    head = head->GetNext();
    if (head == nullptr)
        tail = nullptr;
}
template <class T>
std::shared_ptr<T>& TQueue<T>::Top() {
    return head->item;
}
template <class T>
bool TQueue<T>::Empty() {
    return (head == nullptr) && (tail == nullptr);
}
template <class T>
size_t TQueue<T>::Length() {
    if (head == nullptr && tail == nullptr)
        return 0;
    std::shared_ptr<TQueueItem<T>> temp = head;
    int counter = 0;
    while (temp != tail->GetNext()) {
        temp = temp->GetNext();
        counter++;
    }
    return counter;
}
template <class T>
std::ostream& operator<<(std::ostream& os, const TQueue<T>& queue) {
    std::shared_ptr<TQueueItem<T>> temp = queue.head;
    std::vector<std::shared_ptr<TQueueItem<T>>> v;
    os << "Queue: ";
    os << "=> ";
    while (temp != nullptr) {
        v.push_back(temp);
        temp = temp->GetNext();
    }
    for (int i = v.size() - 1; i >= 0; --i)
        os << *v[i] << " ";
    os << "=>";
    return os;
}
template <class T>
void TQueue<T>::Clear() {
    for (int i = 0; i < this->Length(); i++) {
        this->Pop();
    }
    std::cout << "Queue was cleared but still exist" << std::endl;
}
template <class T>
TQueue<T>::~TQueue() {
    std::cout << "Queue was deleted" << std::endl;
}
template class TQueue<Trapezoid>;

```

```
template std::ostream& operator<<(std::ostream& os, const TQueue<Trapezoid>& queue);
```