

Московский Авиационный Институт  
(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»  
Факультет: «Прикладная математика и физика»  
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №3.  
Тема: «Умные указатели»

Группа: 8О-408Б  
Студент: Забарин Никита Игоревич  
Преподаватель: Поповкин Александр Викторович  
Вариант: №26

Москва  
2017

## Цель работы

Целью лабораторной работы является:

- Закрепление навыков работы с классами.
- Знакомство с умными указателями.

## Задание

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий **все три** фигуры класса фигуры, согласно вариантов задания.

Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Класс-контейнер должен содержать объекты используя `std::shared_ptr<...>`.
- Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.
- Класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера).
- Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток `std::ostream (<<)`.
- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположены в отдельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

- Стандартные контейнеры `std`.
- Шаблоны (template).
- Объекты «по-значению»

Программа должна позволять:

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

## Выводы

## Листинг

```
trapezoid.h:
#ifndef TRAPEZOID_H
#define TRAPEZOID_H

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

class Trapezoid : public Figure
{
public:
    Trapezoid();
    Trapezoid(std::istream& is);

    void print() const override;
    double area() const override;

    Trapezoid& operator = (const Trapezoid& other);
    bool operator == (const Trapezoid& other) const;
```

```

        friend std::ostream& operator << (std::ostream& os, const Trapezoid& trapezoid);
        friend std::istream& operator >> (std::istream& is, Trapezoid& trapezoid);

private:
    double m_sideA;
    double m_sideB;
    double m_height;
};

#endif

queue_item.h:
#ifndef QUEUE_ITEM_H
#define QUEUE_ITEM_H

#include <memory>
#include "figure.h"

class QueueItem
{
public:
    QueueItem(const std::shared_ptr<Figure>& figure);

    void setNext(std::shared_ptr<QueueItem> next);
    std::shared_ptr<QueueItem> getNext();
    std::shared_ptr<Figure> getFigure() const;

private:
    std::shared_ptr<Figure> m_figure;
    std::shared_ptr<QueueItem> m_next;
};

#endif

square.h:
#ifndef SQUARE_H
#define SQUARE_H

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

class Square : public Figure
{
public:
    Square();
    Square(std::istream& is);

    void print() const override;
    double area() const override;

    Square& operator = (const Square& other);
    bool operator == (const Square& other) const;

    friend std::ostream& operator << (std::ostream& os, const Square& square);
    friend std::istream& operator >> (std::istream& is, Square& square);

private:
    double m_side;
};

#endif

```

```

trapezoid.cpp:
#include "trapezoid.h"

Trapezoid::Trapezoid()
{
    m_sideA = 0.0;
    m_sideB = 0.0;
    m_height = 0.0;
}

Trapezoid::Trapezoid(std::istream& is)
{
    is >> *this;
}

void Trapezoid::print() const
{
    std::cout << *this;
}

double Trapezoid::area() const
{
    return m_height * (m_sideA + m_sideB) / 2.0;
}

Trapezoid& Trapezoid::operator = (const Trapezoid& other)
{
    if (&other == this)
        return *this;

    m_sideA = other.m_sideA;
    m_sideB = other.m_sideB;
    m_height = other.m_height;

    return *this;
}

bool Trapezoid::operator == (const Trapezoid& other) const
{
    return m_sideA == other.m_sideA && m_sideB == other.m_sideB && m_height == other.m_height;
}

std::ostream& operator << (std::ostream& os, const Trapezoid& trapezoid)
{
    os << "=====" << std::endl;
    os << "Figure type: trapezoid" << std::endl;
    os << "Side A size: " << trapezoid.m_sideA << std::endl;
    os << "Side B size: " << trapezoid.m_sideB << std::endl;
    os << "Height: " << trapezoid.m_height << std::endl;

    return os;
}

std::istream& operator >> (std::istream& is, Trapezoid& trapezoid)
{
    std::cout << "=====" << std::endl;
    std::cout << "Enter side A: ";
    is >> trapezoid.m_sideA;
    std::cout << "Enter side B: ";
    is >> trapezoid.m_sideB;
    std::cout << "Enter height: ";
    is >> trapezoid.m_height;
}

```

```

        return is;
    }

queue.cpp:
#include "queue.h"

Queue::Queue()
{
    m_size = 0;
}

Queue::~Queue()
{
    while (size() > 0)
        pop();
}

void Queue::push(const std::shared_ptr<Figure>& figure)
{
    std::shared_ptr<QueueItem> itemPtr = std::make_shared<QueueItem>(figure);

    if (m_size == 0)
    {
        m_front = itemPtr;
        m_end = m_front;
    }
    else
    {
        m_end->setNext(itemPtr);
        m_end = itemPtr;
    }

    ++m_size;
}

void Queue::pop()
{
    if (m_size == 1)
    {
        m_front = nullptr;
        m_end = nullptr;
    }
    else
        m_front = m_front->getNext();

    --m_size;
}

unsigned int Queue::size() const
{
    return m_size;
}

std::shared_ptr<Figure> Queue::front() const
{
    return m_front->getFigure();
}

std::ostream& operator << (std::ostream& os, const Queue& queue)
{
    if (queue.size() == 0)
    {
        os << "=====" << std::endl;
    }
}

```

```

        os << "Queue is empty" << std::endl;
    }
    else
    {
        std::shared_ptr<QueueItem> item = queue.m_front;

        while (item != nullptr)
        {
            item->getFigure()->print();
            item = item->getNext();
        }
    }

    return os;
}

rectangle.h:
#ifndef RECTANGLE_H
#define RECTANGLE_H

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

class Rectangle : public Figure
{
public:
    Rectangle();
    Rectangle(std::istream& is);

    void print() const override;
    double area() const override;

    Rectangle& operator = (const Rectangle& other);
    bool operator == (const Rectangle& other) const;

    friend std::ostream& operator << (std::ostream& os, const Rectangle& rectangle);
    friend std::istream& operator >> (std::istream& is, Rectangle& rectangle);

private:
    double m_sideA;
    double m_sideB;
};

#endif

queue_item.cpp:
#include "queue_item.h"

QueueItem::QueueItem(const std::shared_ptr<Figure>& figure)
{
    m_next = nullptr;
    m_figure = figure;
}

void QueueItem::setNext(std::shared_ptr<QueueItem> next)
{
    m_next = next;
}

std::shared_ptr<QueueItem> QueueItem::getNext()
{
    return m_next;
}

```

```

std::shared_ptr<Figure> QueueItem::getFigure() const
{
    return m_figure;
}

```

figure.h:

```

#ifndef FIGURE_H
#define FIGURE_H

```

class Figure

```

{
public:
    virtual ~Figure() {}
    virtual void print() const = 0;
    virtual double area() const = 0;
};

```

```

#endif

```

queue.h:

```

#ifndef QUEUE_H
#define QUEUE_H

```

```

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

```

class Queue

```

{
public:
    Queue();
    ~Queue();

    void push(const std::shared_ptr<Figure>& figure);
    void pop();
    unsigned int size() const;
    std::shared_ptr<Figure> front() const;

    friend std::ostream& operator << (std::ostream& os, const Queue& queue);

```

private:

```

    std::shared_ptr<QueueItem> m_front;
    std::shared_ptr<QueueItem> m_end;
    unsigned int m_size;
};

```

```

#endif

```

square.cpp:

```

#include "square.h"

```

Square::Square()

```

{
    m_side = 0.0;
}

```

Square::Square(std::istream& is)

```

{
    is >> *this;
}

```

void Square::print() const

```

{

```

```

        std::cout << *this;
    }

    double Square::area() const
    {
        return m_side * m_side;
    }

    Square& Square::operator = (const Square& other)
    {
        if (&other == this)
            return *this;

        m_side = other.m_side;

        return *this;
    }

    bool Square::operator == (const Square& other) const
    {
        return m_side == other.m_side;
    }

    std::ostream& operator << (std::ostream& os, const Square& square)
    {
        os << "=====" << std::endl;
        os << "Figure type: square" << std::endl;
        os << "Side size: " << square.m_side << std::endl;

        return os;
    }

    std::istream& operator >> (std::istream& is, Square& square)
    {
        std::cout << "=====" << std::endl;
        std::cout << "Enter side: ";
        is >> square.m_side;

        return is;
    }

makefile:
CC = g++
CFLAGS = -std=c++11 -Wall -Werror -Wno-sign-compare -Wno-unused-result
FILES = *.cpp
PROG = lab3

all:
    $(CC) $(CFLAGS) -o $(PROG) $(FILES)

clean:
    rm $(PROG)

main.cpp:
#include "queue.h"
#include "square.h"
#include "rectangle.h"
#include "trapezoid.h"

int main()
{
    unsigned int action;
    Queue q;

```



```

while (true)
{
    std::cout << "=====" << std::endl;
    std::cout << "Menu:" << std::endl;
    std::cout << "1) Add figure" << std::endl;
    std::cout << "2) Delete figure" << std::endl;
    std::cout << "3) Print" << std::endl;
    std::cout << "0) Quit" << std::endl;
    std::cin >> action;

    if (action == 0)
        break;

    if (action > 3)
    {
        std::cout << "Error: invalid action" << std::endl;

        continue;
    }

    switch (action)
    {
        case 1:
        {
            unsigned int figureType;

            std::cout << "=====" << std::endl;
            std::cout << "1) Square" << std::endl;
            std::cout << "2) Rectangle" << std::endl;
            std::cout << "3) Trapezoid" << std::endl;
            std::cout << "0) Quit" << std::endl;
            std::cin >> figureType;

            if (figureType > 0)
            {
                if (figureType > 3)
                {
                    std::cout << "Error: invalid figure type" << std::endl;

                    continue;
                }

                switch (figureType)
                {
                    case 1:
                    {
                        q.push(std::make_shared<Square>(std::cin));

                        break;
                    }

                    case 2:
                    {
                        q.push(std::make_shared<Rectangle>(std::cin));

                        break;
                    }

                    case 3:
                    {
                        q.push(std::make_shared<Trapezoid>(std::cin));

```

```

        break;
    }
}

break;
}

case 2:
{
    q.pop();

    break;
}

case 3:
{
    std::cout << q;

    break;
}
}

return 0;
}

```

rectangle.cpp:

```
#include "rectangle.h"
```

```
Rectangle::Rectangle()
{
    m_sideA = 0.0;
    m_sideB = 0.0;
}

```

```
Rectangle::Rectangle(std::istream& is)
{
    is >> *this;
}

```

```
void Rectangle::print() const
{
    std::cout << *this;
}

```

```
double Rectangle::area() const
{
    return m_sideA * m_sideB;
}

```

```
Rectangle& Rectangle::operator = (const Rectangle& other)
{
    if (&other == this)
        return *this;

    m_sideA = other.m_sideA;
    m_sideB = other.m_sideB;

    return *this;
}

```

```
bool Rectangle::operator == (const Rectangle& other) const

```

```
{  
    return m_sideA == other.m_sideA && m_sideB == other.m_sideB;  
}
```

```
std::ostream& operator << (std::ostream& os, const Rectangle& rectangle)  
{  
    os << "=====" << std::endl;  
    os << "Figure type: rectangle" << std::endl;  
    os << "Side A size: " << rectangle.m_sideA << std::endl;  
    os << "Side B size: " << rectangle.m_sideB << std::endl;  
  
    return os;  
}
```

```
std::istream& operator >> (std::istream& is, Rectangle& rectangle)  
{  
    std::cout << "=====" << std::endl;  
    std::cout << "Enter side A: ";  
    is >> rectangle.m_sideA;  
    std::cout << "Enter side B: ";  
    is >> rectangle.m_sideB;  
  
    return is;  
}
```