

Московский Авиационный Институт
(Национальный Исследовательский Университет)

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

Лабораторная работа №5.
Тема: «Итераторы»

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 шаблонами классов.
- Построение итераторов для динамических структур данных.

Задание

Используя структуры данных, разработанные для предыдущей лабораторной работы спроектировать и разработать Итератор для динамической структуры данных.

Итератор должен быть разработан в виде шаблона и должен уметь работать со всеми типами фигур, согласно варианту задания.

Итератор должен позволять использовать структуру данных в операторах типа for. Например:
`for(auto i : stack) std::cout << *i << std::endl;`

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

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

Выводы

Листинг

```
queue_item_impl.cpp:
template <class T>
QueueItem<T>::QueueItem(const std::shared_ptr<T>& item)
{
    m_item = item;
}

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

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

template <class T>
std::shared_ptr<T> QueueItem<T>::getItem() const
{
    return m_item;
}

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>

template <class T>
class QueueItem
{
public:
    QueueItem(const std::shared_ptr<T>& item);

    void setNext(std::shared_ptr<QueueItem<T>> next);
    std::shared_ptr<QueueItem<T>> getNext();
    std::shared_ptr<T> getItem() const;

private:
    std::shared_ptr<T> m_item;
    std::shared_ptr<QueueItem<T>> m_next;
};

#include "queue_item_impl.cpp"

#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;
}

```

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_impl.cpp:

```

template <class T>
Queue<T>::Queue()
{
    m_size = 0;
}

```

```

template <class T>
Queue<T>::~~Queue()
{
    while (size() > 0)
        pop();
}

```

```

template <class T>
void Queue<T>::push(const std::shared_ptr<T>& item)

```

```

{
    std::shared_ptr<QueueItem<T>> itemPtr = std::make_shared<QueueItem<T>>(item);

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

    ++m_size;
}

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

    --m_size;
}

template <class T>
unsigned int Queue<T>::size() const
{
    return m_size;
}

template <class T>
std::shared_ptr<T> Queue<T>::front() const
{
    return m_front->getItem();
}

template <class T>
Iterator<QueueItem<T>, T> Queue<T>::begin() const
{
    return Iterator<QueueItem<T>, T>(m_front);
}

template <class T>
Iterator<QueueItem<T>, T> Queue<T>::end() const
{
    return Iterator<QueueItem<T>, T>(nullptr);
}

template <class K>
std::ostream& operator << (std::ostream& os, const Queue<K>& queue)
{
    if (queue.size() == 0)
    {
        os << "=====" << std::endl;
        os << "Queue is empty" << std::endl;
    }
}

```

```

        else
            for (std::shared_ptr<K> item : queue)
                item->print();

        return os;
    }

```

figure.h:

```

#ifndef FIGURE_H
#define FIGURE_H

```

```

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

```

```

#endif

```

iterator_impl.cpp:

```

template <class N, class T>
Iterator<N, T>::Iterator(const std::shared_ptr<N>& item)
{
    m_item = item;
}

```

```

template <class N, class T>
std::shared_ptr<T> Iterator<N, T>::operator * ()
{
    return m_item->getItem();
}

```

```

template <class N, class T>
std::shared_ptr<T> Iterator<N, T>::operator -> ()
{
    return m_item->getItem();
}

```

```

template <class N, class T>
Iterator<N, T> Iterator<N, T>::operator ++ ()
{
    m_item = m_item->getNext();

    return *this;
}

```

```

template <class N, class T>
Iterator<N, T> Iterator<N, T>::operator ++ (int index)
{
    Iterator tmp(m_item);

    m_item = m_item->getNext();

    return tmp;
}

```

```

template <class N, class T>
bool Iterator<N, T>::operator == (const Iterator& other) const
{
    return m_item == other.m_item;
}

```

```

template <class N, class T>
bool Iterator<N, T>::operator != (const Iterator& other) const
{
    return !(*this == other);
}

queue.h:
#ifndef QUEUE_H
#define QUEUE_H

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

template <class T>
class Queue
{
public:
    Queue();
    ~Queue();

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

    Iterator<QueueItem<T>, T> begin() const;
    Iterator<QueueItem<T>, T> end() const;

    template <class K>
    friend std::ostream& operator << (std::ostream& os, const Queue<K>& queue);

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

#include "queue_impl.cpp"

#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 = main.cpp square.cpp rectangle.cpp trapezoid.cpp
PROG = lab5

```

```

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<Figure> 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;
}

```

iterator.h:

```

#ifndef ITERATOR_H
#define ITERATOR_H

```

```

template <class N, class T>
class Iterator
{
public:
    Iterator(const std::shared_ptr<N>& item);

    std::shared_ptr<T> operator * ();
    std::shared_ptr<T> operator -> ();
    Iterator operator ++ ();
    Iterator operator ++ (int index);
    bool operator == (const Iterator& other) const;
    bool operator != (const Iterator& other) const;

```

```

private:
    std::shared_ptr<N> m_item;
};

```

```

#include "iterator_impl.cpp"

```

```

#endif

```