Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики ипрограммирования

Отчет по лабораторным работам

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Цысь Г.В
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

_____

Подпись студента

_____

Число
21.10.2018

Москва-2018

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики ипрограммирования

Отчет по лабораторной работе № 1

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Цысь Г.В
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

_____

Подпись студента

_____

Число
21.10.2018

Москва-2018

Лабораторная работа №1

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно вариантов задания. Классы должны удовлетворять следующим правилам:

- Должны иметь общий родительский класс Figure.
- Должны иметь общий виртуальный метод Print, печатающий параметры фигуры и ее тип в стандартный поток вывода cout.
- Должный имеет общий виртуальный метод расчета площади фигуры – Square.
- Должны иметь конструктор, считывающий значения основных параметров фигуры из стандартного потока cin.
- Должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Программа должна позволять вводить фигуру каждого типа с клавиатуры, выводить параметры фигур на экран и их площадь.

ОПИСАНИЕ

Классом называется составной тип данных, элементами которого являются функции и переменные (поля). В основу понятия класс положен тот факт, что «над объектами можно совершать различные операции». Свойства объектов описываются с помощью полей классов, а действия над объектами описываются с помощью функций, которые называются методами класса. Класс имеет имя, состоит из полей, называемых членами класса, и функций — методов класса.

Виртуальные функции — специальный вид функций-членов класса. Виртуальная функция отличается об обычной функции тем, что для обычной функции связывание вызова функции с ее определением осуществляется на этапе компиляции. Для виртуальных функций это происходит во время выполнения программы.

Для объявления виртуальной функции используется ключевое слово virtual. Функция-член класса может быть объявлена как виртуальная, если

- класс, содержащий виртуальную функцию, базовый в иерархии порождения;
- реализация функции зависит от класса и будет различной в каждом порожденном классе.

ЛИСТИНГ ПРОГРАММЫ

Figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
class Figure
{
  public:
    virtual double Square() = 0;
    virtual void Print() = 0;
    virtual ~Figure(){};
};
#endif /* FIGURE_H */
```

Trapeze.h

```
#ifndef TRAPEZE_H
#define TRAPEZE_H
#include <iostream>
#include "figure.h"

class Trapeze : public Figure
{
  public:
    Trapeze();
    Trapeze(std::istream &is);
    double Square() override;
    void Print() override;
    virtual ~Trapeze();

  private:
    int side_1, side_2, height;
};

#endif
```

Trapeze.cpp

```
#include "trapeze.h"
#include <iostream>
#include <cmath>

Trapeze::Trapeze(std::istream &is)
{
    do
    {
        std::cout << "Enter first side: ";
        is >> side_1;
        std::cout << "Enter second side: ";
        is >> side_2;
        std::cout << "Enter the height: ";
        is >> height;
    } while ((side_1 <= 0) || (side_2 <= 0) || (height <= 0));
    std::cout << "Correct value"
              << "\n";
}

double Trapeze::Square()
{
    return ((side_1 + side_2) / 2) * height;
}

void Trapeze::Print()
{
    std::cout << "Trapeze: "
              << "\n";
    std::cout << "side 1:  " << side_1 << " side 2: " << side_2 << " height:
" << height << std::endl;
}

Trapeze::~Trapeze()
{
    std::cout << "Trapeze deleted" << std::endl;
}
```

Rhombus.h

```
#ifndef RHOMBUS_H
#define RHOMBUS_H
#include <iostream>
#include "figure.h"

class Rhombus : public Figure
{
  public:
    Rhombus();
```

```
    Rhombus(std::istream &is);
    double Square() override;
    void Print() override;
    virtual ~Rhombus();

  private:
    double angle;
    int side;
};

#endif
```

Rhombus.cpp

```
#include "rhombus.h"
#include <iostream>
#include <cmath>

Rhombus::Rhombus(std::istream &is)
{
    do
    {
        std::cout << "Enter the angle: ";
        is >> angle;
        std::cout << "Enter the side: ";
        is >> side;
    } while ((angle <= 0) || (angle >= 180) || (side <= 0));
    std::cout << "Correct value"
              << "\n";
}

double Rhombus::Square()
{
    return side * side * sin((angle * M_PI) / 180);
}

void Rhombus::Print()
{
    std::cout << "Rhombus: "
              << "\n";
    std::cout << "angle: " << angle << " side: " << side << std::endl;
}

Rhombus::~Rhombus()
{
    std::cout << "Rhombus deleted" << std::endl;
}
```

Pentagon.h

```
#ifndef PENTAGON_H
#define PENTAGON_H
#include <iostream>
#include "figure.h"

class Pentagon : public Figure
{
  public:
    Pentagon();
    Pentagon(std::istream &is);
    double Square() override;
    void Print() override;
    virtual ~Pentagon();

  private:
    double x[5], y[5];
};

#endif
```

Pentagon.cpp

```
#include "pentagon.h"
#include <iostream>
#include <cmath>

Pentagon::Pentagon(std::istream &is)
{
    for (int i = 0; i < 5; i++)
    {
        std::cout << "Enter coordinates by point в„– " << i + 1 << " ";
        is >> x[i];
        is >> y[i];
    };
}


double Pentagon::Square()
{
    double S = 0;
    for (int i = 0; i < 4; i++)
    {
        S = x[i] * y[i + 1] + S;
    };
    S = S + x[4] * y[0];
    for (int i = 0; i < 4; i++)
        S = S - x[i + 1] * y[i];
    S = S - x[0] * y[4];
    return std::abs(S / 2.0);
}

void Pentagon::Print()
{
    std::cout << "Pentagon: "
              << "\n";
    std::cout << "Coordinates of pentagon: "
              << "\n";
    for (int i = 0; i < 5; i++)
    {
        std::cout << "X: " << x[i] << " Y: " << y[i] << std::endl;
    }
}

Pentagon::~Pentagon()
{
    std::cout << "Pentagon deleted" << std::endl;
}
```

Main.cpp

```
#include <cstdlib>
#include "trapeze.h"
#include "rhombus.h"
#include "pentagon.h"

void Menu()
{
    std::cout << "\n"
              << std::endl;
    std::cout << "1. Trapeze" << std::endl;
    std::cout << "2. Rhombus" << std::endl;
    std::cout << "3. Pentagon" << std::endl;
    std::cout << "0. Exit" << std::endl;
    std::cout << "Enter figure number:";
}

int main(int argc, char **argv)
{
    Menu();
    int action;
```

```
        std::cin >> action;
        while (action != 0)
        {
            if (action == 1)
            {
                /*int* a = (int*)malloc(sizeof(int)*5);
                a[0] = 1;
                free(a);*/
                //Trapeze
                std::cout << "Trapeze created" << std::endl;
                Figure *ptr = new Trapeze(std::cin);
                ptr->Print();
                std::cout << "Square of trapeze:" << ptr->Square() << std::endl;
                delete ptr;
                Menu();
                std::cin >> action;
            }
            else if (action == 2)
            {

                //Rhombus
                std::cout << "Rhombus created: " << std::endl;
                Figure *ptr = new Rhombus(std::cin);
                ptr->Print();
                std::cout << "Square of rhombus:" << ptr->Square() << std::endl;
                delete ptr;
                Menu();
                std::cin >> action;
            }
            else if (action == 3)
            {

                //Pentagon
                std::cout << "Pentagon created: " << std::endl;
                Figure *ptr = new Pentagon(std::cin);
                ptr->Print();
                std::cout << "Square of pentagon:" << ptr->Square() << std::endl;
                delete ptr;
                Menu();
                std::cin >> action;
            }
            else
            {
                std::cout << "Incorrect input!" << std::endl;
                Menu();
                std::cin >> action;
            }
        }
        return 0;
    }
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
    1. Trapeze
    2. Rhombus
    3. Pentagon
    0. Exit
    Enter figure number:1
    Trapeze created
    Enter first side: 2
    Enter second side: 4
    Enter the height: 3
    Correct value
    Trapeze:
    side 1:  2 side 2: 4 height: 3
    Square of trapeze:9
    Trapeze deleted

    1. Trapeze
    2. Rhombus
    3. Pentagon
    0. Exit
    Enter figure number:2
```

```
Rhombus created:
Enter the angle: 30
Enter the side: 5
Correct value
Rhombus:
angle: 30 side: 5
Square of rhombus:12.5
Rhombus deleted

1. Trapeze
2. Rhombus
3. Pentagon
0. Exit
Enter figure number:3
Pentagon created:
Enter coordinates by point в„– 1 2 3
Enter coordinates by point в„– 2 -2 5
Enter coordinates by point в„– 3 3 -4
Enter coordinates by point в„– 4 7 0
Enter coordinates by point в„– 5 4 1
Pentagon:
Coordinates of pentagon:
X: 2 Y: 3
X: -2 Y: 5
X: 3 Y: -4
X: 7 Y: 0
X: 4 Y: 1
Square of pentagon:27
Pentagon deleted
```

ВЫВОД

В первой лабораторной работе в курсе изучения объектно-ориентированного программирования нам предлагалось ознакомиться с базовыми понятиями C++ такими, как классы, наследование, конструкторы и деструкторы, виртуальные функции и тд. В ходе написания программ, я лучше разобрался в этих понятиях и изучил их принципы работы.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики ипрограммирования

Отчет по лабораторной работе № 2

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Цысь Г.В
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

_____

Подпись студента

_____

Число
21.10.2018

Москва-2018

Лабораторная работа №2

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру (колонка фигура 1), согласно вариантов задания и (реализованную в ЛР1). Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream (<<). Оператор должен распечатывать параметры фигуры (тип фигуры, длины сторон, радиус и т.д).
- Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream (>>). Оператор должен вводить основные параметры фигуры (длины сторон, радиус и т.д).
- Классы фигур должны иметь операторы копирования (=).
- Классы фигур должны иметь операторы сравнения с такими же фигурами (==).
- Класс-контейнер должен содежержать объекты фигур “по значению” (не по ссылке).
- 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 контейнераю
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

В C++ разработана новая библиотека ввода-вывода iostream, использующая концепцию объектно-ориентированного программирования:

Библиотека iostream определяет три стандартных потока:

- cin стандартный входной поток (stdin в C)
- cout стандартный выходной поток (stdout в C)
- cerr стандартный поток вывода сообщений об ошибках (stderr в C)

Механизм перегрузки операций позволяет обеспечить более традиционную и удобную запись действий над объектами. Для перегрузки встроенных операторов используется ключевое слова operator.

Тип возвращаемого значения должен быть отличным от void, если необходимо использовать перегруженную операцию внутри другого выражения.

Имеется два способа описания функции, соответствующей переопределяемой операции:

- если функция задается как обычная функция-элемент класса, то первым операндом операции является объект класса, указатель на который передается неявным параметром this;

- если первый операнд переопределяемой операции не является объектом некоторого класса, либо требуется передавать в

качестве операнда не указатель, а сам объект (значение), то соответствующая функция должна быть определена как дружественная классу с полным списком аргументов.

ЛИСТИНГ ПРОГРАММЫ

На протяжении всех лабораторных работ мы работаем с классом фигур, в частности с фигурами, соответствующими определенному варианту. Такие файлы, как Figure.h, Trapeze.h, Trapeze.cpp, Pentagon.h, Pentagon.cpp, Rhombus.h, Rhombus.cpp не изменяются.

TQueue.h

```
#ifndef TQUEUE_H
#define TQUEUE_H

#include "Trapeze.h"
#include "TQueueItem.h"

class TQueue {
public:
	TQueue();
	TQueue(const TQueue& orig);

	void Push(Trapeze &&trapeze);
	bool Empty();
	Trapeze Pop();
	friend std::ostream& operator<<(std::ostream& os, const TQueue& queue);
	virtual ~TQueue();
private:
	TQueueItem *head;
	TQueueItem *last;
};

#endif
```

TQueue.cpp

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

TQueue::TQueue() : head(nullptr), last(nullptr) {
}

TQueue::TQueue(const TQueue& orig) {
	head = orig.head;
	last = orig.last;
}

std::ostream& operator<<(std::ostream& os, const TQueue& queue) {
	TQueueItem *item = queue.head;

	while(item!=nullptr)
	{
		os << *item;
		item = item->GetNext();
	}
	return os;
}

void TQueue::Push(Trapeze &&trapeze) {
	TQueueItem *other = new TQueueItem(trapeze);
	if (Empty())  {
		head = other;
		last = other;
	}
	else {
		last->SetNext(other);
		last = other;
	}
```

```
}

bool TQueue::Empty() {
	return head == nullptr;
}

Trapeze TQueue::Pop() {
	Trapeze result;
	if (head != nullptr) {
		TQueueItem *old_head = head;
		head = head->GetNext();
		result = old_head->GetTrapeze();
		old_head->SetNext(nullptr);
		delete old_head;
	}
	return result;
}

TQueue::~TQueue() {
	delete head;
}
```

TQueueItem.h

```
#ifndef TQUEUEITEM_H
#define TQUEUEITEM_H

#include "Trapeze.h"

class TQueueItem {
public:
	TQueueItem(const Trapeze& trapeze);
	TQueueItem(const TQueueItem& orig);
	friend std::ostream& operator<<(std::ostream& os, const TQueueItem&
obj);

	TQueueItem* SetNext(TQueueItem* next);
	TQueueItem* GetNext();
	Trapeze GetTrapeze() const;

	virtual ~TQueueItem();
private:
	Trapeze trapeze;
	TQueueItem *next;
};

#endif
```

TQueueItem.cpp

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

TQueueItem::TQueueItem(const Trapeze& trapeze) {
	this->trapeze = trapeze;
	this->next = nullptr;
	std::cout << "Queue item: created" << "\n";
}

TQueueItem::TQueueItem(const TQueueItem& orig) {
	this->trapeze = orig.trapeze;
	this->next = orig.next;
	std::cout << "Queue item: copied" << "\n";
}

TQueueItem* TQueueItem::SetNext(TQueueItem* next) {
	TQueueItem* old = this->next;
	this->next = next;
	return old;
}

Trapeze TQueueItem::GetTrapeze() const {
	return this->trapeze;
```

```
}

TQueueItem* TQueueItem::GetNext() {
	return this->next;
}

TQueueItem::~TQueueItem() {
	std::cout << "Queue item: deleted" << "\n";
	delete next;
}

std::ostream& operator<<(std::ostream& os, const TQueueItem& obj) {
	os << "[" << obj.trapeze << "]" << "\n";
	return os;
}
```

Main.cpp

```
#include <iostream>
#include <locale>
#include "TQueue.h"

void Menu()
{
	std::cout << "1. Create queue\n";
	std::cout << "2. Add item\n";
	std::cout << "3. Delete item\n";
	std::cout << "4. Print queue\n";
	std::cout << "5. Delete queue\n";
	std::cout << "6. Menu\n";
	std::cout << "0. Exit\n";
	std::cout << "\n";
}

int main() {
	TQueue *q = nullptr;
	int action;
	Menu();
	std::cin >> action;

	while (action != 0)
	{
		if (action == 1 && q == nullptr){
			q = new TQueue;
			std::cout << "Queue created\n";
		}
		//std::cout << q << " " << action << std::endl;
		if (action == 2 && q != nullptr)
			q->Push(Trapeze(std::cin));
		if (action == 3 && q != nullptr){
			if (!q->Empty()){
				Trapeze r = q->Pop();
				std::cout << r << "\n";
			}
			else
				std::cout << "Queue is empty\n";
		}
		if (action == 4 && q != nullptr)
			std::cout << *q;
		if (action == 5 && q != nullptr){
			delete q;
			std::cout << "Queue deleted\n";
			q = nullptr;
		}
		if (action == 6)
			Menu();
		std::cin >> action;
	}
}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
1. Create queue
2. Add item
```

```
3. Delete item
4. Print queue
5. Delete queue
6. Menu
0. Exit

1
Queue created
2
Enter first side: 1
Enter second side: 1
Enter the height: 1
Correct value
Trapeze created:
Trapeze copied
Queue item: created
Trapeze deleted
2
Enter first side: 2
Enter second side: 2
Enter the height: 2
Correct value
Trapeze created:
Trapeze copied
Queue item: created
Trapeze deleted
2
Enter first side: 3
Enter second side: 3
Enter the height: 3
Correct value
Trapeze created:
Trapeze copied
Queue item: created
Trapeze deleted
4
[a = 1, b = 1, h = 1]
[a = 2, b = 2, h = 2]
[a = 3, b = 3, h = 3]
3
Trapeze created:
Trapeze copy created
Trapeze copied
Trapeze deleted
Queue item: deleted
Trapeze deleted
a = 1, b = 1, h = 1
Trapeze deleted
4
[a = 2, b = 2, h = 2]
[a = 3, b = 3, h = 3]
3
Trapeze created:
Trapeze copy created
Trapeze copied
Trapeze deleted
Queue item: deleted
Trapeze deleted
a = 2, b = 2, h = 2
Trapeze deleted
4
[a = 3, b = 3, h = 3]
5
Queue item: deleted
Trapeze deleted
Queue deleted
0
```

ВЫВОД

В ходе второй лабораторной работы я создал динамическую структуру данных, являющейся очередью. Объекты передаются контейнеру первого уровня по вводимым с консоли значениям.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики ипрограммирования

Отчет по лабораторной работе № 3

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Цысь Г.В
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

_____

Подпись студента

_____

Число
21.10.2018

Москва-2018

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий все три фигуры класса фигуры, согласно вариантов задания (реализованную в ЛР1). 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 раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp)

ЛИСТИНГ ПРОГРАММЫ

TQueue.h

```
#ifndef TQUEUE_H
#define TQUEUE_H

#include "Trapeze.h"
#include "Pentagon.h"
#include "Rhombus.h"
#include "TQueueItem.h"
#include <memory>

class TQueue {
public:
	TQueue();
	TQueue(const TQueue& orig);

	void Push(std::shared_ptr<Figure> &&figure);
	bool Empty();
	std::shared_ptr<Figure> Pop();
	friend std::ostream& operator<<(std::ostream& os, const TQueue& queue);
	virtual ~TQueue();
private:
	std::shared_ptr<TQueueItem> head;
	std::shared_ptr<TQueueItem> last;
};

#endif
```

TQueue.cpp

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

TQueue::TQueue() : head(nullptr), last(nullptr) {
}

TQueue::TQueue(const TQueue& orig) {
	head = orig.head;
	last = orig.last;
}

std::ostream& operator<<(std::ostream& os, const TQueue& queue) {
```

```
	std::shared_ptr<TQueueItem> item = queue.head;

	while(item!=nullptr)
	{
		os << *item;
		item = item->GetNext();
	}
	return os;
}

void TQueue::Push(std::shared_ptr<Figure> &&figure) {
	std::shared_ptr<TQueueItem> other(new TQueueItem(figure));
	if (Empty())  {
		head = other;
		last = other;
	}
	else {
		last->SetNext(other);
		last = other;
	}
}

bool TQueue::Empty() {
	return head == nullptr;
}

std::shared_ptr<Figure> TQueue::Pop() {
	std::shared_ptr<Figure> result;
	if (head != nullptr) {
		result = head->GetFigure();
		head = head->GetNext();
	}
	return result;
}

TQueue::~TQueue() {
}
```

TQueueItem.h

```
#ifndef TQUEUEITEM_H
#define TQUEUEITEM_H

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

class TQueueItem {
public:
	TQueueItem(const std::shared_ptr<Figure>& figure);
	TQueueItem(const TQueueItem& orig);
	friend std::ostream& operator<<(std::ostream& os, const TQueueItem&
obj);

	std::shared_ptr<TQueueItem> SetNext(std::shared_ptr<TQueueItem> &next);
	std::shared_ptr<TQueueItem> GetNext();
	std::shared_ptr<Figure> GetFigure() const;

	virtual ~TQueueItem();
private:
	std::shared_ptr<Figure> figure;
	std::shared_ptr<TQueueItem> next;
};

#endif
```

TQueueItem.cpp

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

TQueueItem::TQueueItem(const std::shared_ptr<Figure> &figure) {
	this->figure = figure;
	this->next = nullptr;
```

```
	std::cout << "Queue item: created" << "\n";
}

TQueueItem::TQueueItem(const TQueueItem& orig) {
	this->figure = orig.figure;
	this->next = orig.next;
	std::cout << "Queue item: copied" << "\n";
}

std::shared_ptr<TQueueItem> TQueueItem::SetNext(std::shared_ptr<TQueueItem>
&next) {
	std::shared_ptr<TQueueItem> old = this->next;
	this->next = next;
	return old;
}

std::shared_ptr<Figure> TQueueItem::GetFigure() const {
	return this->figure;
}

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

TQueueItem::~TQueueItem() {
	std::cout << "Queue item: deleted" << "\n";
}

std::ostream& operator<<(std::ostream& os, const TQueueItem& obj) {
	//os << "[" << *obj.figure << "]" << "\n";
	obj.figure->Print();
	return os;
}
```

Main.cpp

```
#include <iostream>
#include <locale>
#include <memory>
#include "TQueue.h"

void Menu()
{
	std::cout << "1. Create queue\n";
	std::cout << "2. Add item\n";
	std::cout << "    3. Add trapeze\n";
	std::cout << "    4. Add rhombus\n";
	std::cout << "    5. Add pentagon\n";
	std::cout << "6. Delete item\n";
	std::cout << "7. Print queue\n";
	std::cout << "8. Menu\n";
	std::cout << "0. Exit\n";
	std::cout << "\n";
}

int main() {
	std::shared_ptr<TQueue> q = nullptr;
	int action;
	Menu();
	std::cin >> action;

	while (action != 0)
	{
		if (action == 1 && q == nullptr){
			q = std::shared_ptr<TQueue> (new TQueue);
			std::cout << "Queue created\n";
		}
		//std::cout << q << " " << action << std::endl;
		if (action == 2 && q != nullptr) {
			int cnt;
			std::cin >> cnt;
			if (cnt == 3) {
```

```
                        q->Push(std::shared_ptr<Figure> (new
Trapeze(std::cin)));
                }
                else if (cnt == 4) {
                        q->Push(std::shared_ptr<Figure> (new
Rhombus(std::cin)));
                }
                else if (cnt == 5) {
                        q->Push(std::shared_ptr<Figure> (new
Pentagon(std::cin)));
                }
            }
            if (action == 6 && q != nullptr){
                if (!q->Empty()){
                        auto r = q->Pop();
                        //std::cout << r << "\n";
                }
                else
                        std::cout << "Queue is empty\n";
            }
            if (action == 7 && q != nullptr)
                std::cout << *q;
            /*if (action == 5 && q != nullptr){
                delete q;
                std::cout << "Queue deleted\n";
                q = nullptr;
            }*/
            if (action == 8)
                Menu();
            std::cin >> action;
      }
      //int* a;
      //a = new int;
      /*int b = 1;
      std::shared_ptr<int> a(new int);
      int *c = a.get();
      *c = b;
      std::cout << *a << std::endl;*/
      /*std::shared_ptr<Figure> f(new Trapeze(std::cin));
      f->Print();*/
}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
1. Create queue
2. Add item
      3. Add trapeze
      4. Add rhombus
      5. Add pentagon
6. Delete item
7. Print queue
8. Menu
0. Exit

1
Queue created
2
3
Enter first side: 1
Enter second side: 1
Enter the height: 1
Correct value
Queue item: created
2
4
Enter the angle: 30
Enter the side: 2
Correct value
Queue item: created
2
5
Enter the side: 3
Correct value
Queue item: created
7
```

```
Trapeze:
side 1:  1 side 2: 1 height: 1
Rhombus:
angle: 30 side: 2
Pentagon:
Side: 3
6
Queue item: deleted
Trapeze deleted
7
Rhombus:
angle: 30 side: 2
Pentagon:
Side: 3
6
Queue item: deleted
Rhombus deleted
7
Pentagon:
Side: 3
6
7
0
Queue item: deleted
Pentagon deleted
```

ВЫВОД

В лабораторной работе №3 мы добавляем оставшиеся две фигуры и оптимизируем программу с помощью умных указателей. Таким образом, умные указатели помогают нам с очищением памяти, когда это необходимо и избегать какие-либо утечки памяти.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики ипрограммирования

Отчет по лабораторной работе № 4

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Цысь Г.В
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

_____

Подпись студента

_____

Число
21.10.2018

Москва-2018

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ шаблон класса- контейнера первого уровня, содержащий все три фигуры класса фигуры, согласно вариантов задания (реализованную в ЛР1). Классы должны удовлетворять следующим правилам:

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

ОПИСАНИЕ

Шаблоны (templates) - очень мощное средство. Шаблонные функции и классы позволяют очень сильно упростить программисту жизнь и сберечь огромное количество времени, сил и нервов. Если вам покажется, что шаблоны не сильно-то и значимая тема для изучения, знайте - вы заблуждаетесь.

У шаблонных функций должен быть аргумент, чтобы компилятор мог определить какой именно тип использовать. В шаблонах можно использовать несколько параметрических типов, и конечно же можно смешивать параметрические типы со стандартными (только нужно позаботиться о правильном приведении типов).

При создании объекта, после имени класса нужно поставить угловые скобки, в которых указать нужный тип. После этого объекты используются так, как мы привыкли. У шаблонных классов есть одна потрясающая особенность - кроме стандартных типов, они могут работать и с пользовательскими.

ЛИСТИНГ ПРОГРАММЫ

TQueue.h

```
#ifndef TQUEUE_H
#define TQUEUE_H

#include "Trapeze.h"
#include "Pentagon.h"
#include "Rhombus.h"
#include "TQueueItem.h"
#include <memory>

template <class T> class TQueue {
public:
	TQueue();
	TQueue(const TQueue& orig);

	void Push(std::shared_ptr<T> &&item);
	bool Empty();
	std::shared_ptr<T> Pop();
```

```
	template <class A>friend std::ostream& operator<<(std::ostream& os,
const TQueue<A>& queue);
	virtual ~TQueue();
private:
	std::shared_ptr<TQueueItem<T>> head;
	std::shared_ptr<TQueueItem<T>> last;
};

#endif
```

TQueue.cpp

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

template <class T> TQueue<T>::TQueue() : head(nullptr), last(nullptr) {
}

template <class T> TQueue<T>::TQueue(const TQueue<T>& orig) {
	head = orig.head;
	last = orig.last;
}

template <class T> std::ostream& operator<<(std::ostream& os, const
TQueue<T>& queue) {
	std::shared_ptr<TQueueItem<T>> item = queue.head;

	while(item!=nullptr)
	{
		os << *item;
		item = item->GetNext();
	}
	return os;
}

template <class T> void TQueue<T>::Push(std::shared_ptr<T> &&item) {
	std::shared_ptr<TQueueItem<T>> other(new TQueueItem<T>(item));
	if (Empty())  {
		head = other;
		last = other;
	}
	else {
		last->SetNext(other);
		last = other;
	}
}

template <class T> bool TQueue<T>::Empty() {
	return head == nullptr;
}

template <class T> std::shared_ptr<T> TQueue<T>::Pop() {
	std::shared_ptr<T> result;
	if (head != nullptr) {
		result = head->GetFigure();
		head = head->GetNext();
	}
	return result;
}

template <class T> TQueue<T>::~TQueue() {
}

#include "Figure.h"
template class TQueue<Figure>;
template std::ostream& operator<<(std::ostream& os, const TQueue<Figure>&
stack);
```

TQueueItem.h

```
#ifndef TQUEUEITEM_H
#define TQUEUEITEM_H

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

template <class T> class TQueueItem {
public:
	TQueueItem(const std::shared_ptr<T>& figure);
	TQueueItem(const TQueueItem& orig);
	template <class A> friend std::ostream& operator<<(std::ostream& os,
const TQueueItem<A>& obj);

	std::shared_ptr<TQueueItem<T>> SetNext(std::shared_ptr<TQueueItem>
&next);
	std::shared_ptr<TQueueItem<T>> GetNext();
	std::shared_ptr<T> GetFigure() const;

	virtual ~TQueueItem();
private:
	std::shared_ptr<T> figure;
	std::shared_ptr<TQueueItem<T>> next;
};

#endif
```

TQueueItem.cpp

```
#include "TQueueItem.h"
#include <iostream>
template <class T> TQueueItem<T>::TQueueItem(const std::shared_ptr<T>
&figure) {
	this->figure = figure;
	this->next = nullptr;
	std::cout << "Queue item: created" << "\n";
}
template <class T> TQueueItem<T>::TQueueItem(const TQueueItem& orig) {
	this->figure = orig.figure;
	this->next = orig.next;
	std::cout << "Queue item: copied" << "\n";
}
template <class T> std::shared_ptr<TQueueItem<T>>
TQueueItem<T>::SetNext(std::shared_ptr<TQueueItem<T>> &next) {
	std::shared_ptr<TQueueItem<T>> old = this->next;
	this->next = next;
	return old;
}
template <class T> std::shared_ptr<T> TQueueItem<T>::GetFigure() const {
	return this->figure;
}
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" << "\n";
}
template <class A> std::ostream& operator<<(std::ostream& os, const
TQueueItem<A>& obj) {
	//os << "[" << *obj.figure << "]" << "\n";
	obj.figure->Print();
	return os;
}

#include "Figure.h"
template class TQueueItem<Figure>;
template std::ostream& operator<<(std::ostream& os, const
TQueueItem<Figure>&stack);
```

Main.cpp

```
#include <iostream>
```

```
#include <locale>
#include <memory>
#include "TQueue.h"
void Menu()
{
	std::cout << "1. Create queue\n";
	std::cout << "2. Add item\n";
	std::cout << "    3. Add trapeze\n";
	std::cout << "    4. Add rhombus\n";
	std::cout << "    5. Add pentagon\n";
	std::cout << "6. Delete item\n";
	std::cout << "7. Print queue\n";
	std::cout << "8. Menu\n";
	std::cout << "0. Exit\n";
	std::cout << "\n";
}
int main() {
	std::shared_ptr<TQueue<Figure>> q = nullptr;
	int action;
	Menu();
	std::cin >> action;

	while (action != 0)
	{
		if (action == 1 && q == nullptr){
			q = std::shared_ptr<TQueue<Figure>> (new TQueue<Figure>);
			std::cout << "Queue created\n";
		}
		//std::cout << q << " " << action << std::endl;
		if (action == 2 && q != nullptr) {
			int cnt;
			std::cin >> cnt;
			if (cnt == 3) {
				q->Push(std::shared_ptr<Figure> (new
Trapeze(std::cin)));
			}
			else if (cnt == 4) {
				q->Push(std::shared_ptr<Figure> (new
Rhombus(std::cin)));
			}
			else if (cnt == 5) {
				q->Push(std::shared_ptr<Figure> (new
Pentagon(std::cin)));
			}
		}
		if (action == 6 && q != nullptr){
			if (!q->Empty()){
				auto r = q->Pop();
				//std::cout << r << "\n";
			}
			else
				std::cout << "Queue is empty\n";
		}
		if (action == 7 && q != nullptr)
			std::cout << *q;
		/*if (action == 5 && q != nullptr){
			delete q;
			std::cout << "Queue deleted\n";
			q = nullptr;
		}*/
		if (action == 8)
			Menu();
		std::cin >> action;
	}
	//int* a;
	//a = new int;
	/*int b = 1;
	std::shared_ptr<int> a(new int);
	int *c = a.get();
	*c = b;
	std::cout << *a << std::endl;*/
```

```
	/*std::shared_ptr<Figure> f(new Trapeze(std::cin));
	f->Print();*/
	/*TQueue q;
	q.Push(std::shared_ptr<Figure> (new Trapeze(std::cin)));
	q.Push(std::shared_ptr<Figure> (new Rhombus(std::cin)));
	q.Push(std::shared_ptr<Figure> (new Pentagon(std::cin)));
	std::cout << q;*/
}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
1. Create queue
2. Add item
	3. Add trapeze
	4. Add rhombus
	5. Add pentagon
6. Delete item
7. Print queue
8. Menu
0. Exit

1
Queue created
2
5
Enter the side: 1
Correct value
Queue item: created
2
4
Enter the angle: 30
Enter the side: 2
Correct value
Queue item: created
2
3
Enter first side: 3
Enter second side: 3
Enter the height: 3
Correct value
Queue item: created
7
Pentagon:
Side: 1
Rhombus:
angle: 30 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
6
Queue item: deleted
Pentagon deleted
7
Rhombus:
angle: 30 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
6
Queue item: deleted
Rhombus deleted
7
Trapeze:
side 1:  3 side 2: 3 height: 3
6
7
0
Queue item: deleted
Trapeze deleted
```

ВЫВОД

В ходе 4 лабораторной работы я познакомился с шаблонами. Они упрощаю и сокращают код программы, так как создаются для всех типов данных и делают программу(часть программы) универсальной.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики ипрограммирования

Отчет по лабораторной работе № 5

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Цысь Г.В
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

_____

Подпись студента

_____

Число
21.10.2018

Москва-2018

ЗАДАНИЕ

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР№4) 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.

ОПИСАНИЕ

Для доступа к элементам некоторого множества элементов алгоритмы stl используют специальные объекты, называемые итераторами. В контейнерных типах stl они доступны через методы класса (например, begin() в шаблоне класса vector). Функциональные возможности указателей и итераторов близки, так что обычный указатель тоже может использоваться как итератор. категории итераторов:

- итератор ввода (input iterator) - используется потоками ввода;
- итератор вывода (output iterator) - используется потоками вывода;
- однонаправленный итератор (forward iterator) - для прохода по элементам в одном направлении;
- двунаправленный итератор (bidirectional iterator) - способен пройтись по элементам в любом направлении. Такие итераторы реализованы в некоторых контейнерных типах stl (list, set, multiset, map, multimap);
- итераторы произвольного доступа (random access) - через них можно иметь доступ к любому элемента. Такие итераторы реализованы в некоторых контейнерных типах stl (vector, deque, string, array).

ЛИСТИНГ ПРОГРАММЫ

TIterator.h

```
#ifndef TITERATOR_H
#define TITERATOR_H
#include <memory>
#include <iostream>
template <class node, class T>
class TIterator
{
  public:
    TIterator(std::shared_ptr<node> n)
    {
        node_ptr = n;
    }

    std::shared_ptr<T> operator*()
    {
        return node_ptr->GetFigure();
    }

    std::shared_ptr<T> operator->()
    {
        return node_ptr->GetFigure();
    }

    void operator++()
    {
        node_ptr = node_ptr->GetNext();
    }
```

```
    TIterator operator++(int)
    {
        TIterator iter(*this);
        ++(*this);
        return iter;
    }

    bool operator==(TIterator const &i)
    {
        return node_ptr == i.node_ptr;
    }

    bool operator!=(TIterator const &i)
    {
        return !(*this == i);
    }

  private:
    std::shared_ptr<node> node_ptr;
};
#endif
```

TQueue.h

```
#ifndef TQUEUE_H
#define TQUEUE_H

#include "Trapeze.h"
#include "Pentagon.h"
#include "Rhombus.h"
#include "TQueueItem.h"
#include "TIterator.h"
#include <memory>

template <class T> class TQueue {
public:
      TQueue();
      TQueue(const TQueue& orig);

      void Push(std::shared_ptr<T> &&item);
      bool Empty();
      std::shared_ptr<T> Pop();

      TIterator<TQueueItem<T>,T> begin();
      TIterator<TQueueItem<T>,T> end();

      template <class A>friend std::ostream& operator<<(std::ostream& os,
const TQueue<A>& queue);
      virtual ~TQueue();
private:
      std::shared_ptr<TQueueItem<T>> head;
      std::shared_ptr<TQueueItem<T>> last;
};

#endif
```

TQueue.cpp

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

template <class T> TQueue<T>::TQueue() : head(nullptr), last(nullptr) {
}

template <class T> TQueue<T>::TQueue(const TQueue<T>& orig) {
      head = orig.head;
      last = orig.last;
}

template <class T> std::ostream& operator<<(std::ostream& os, const
TQueue<T>& queue) {
      std::shared_ptr<TQueueItem<T>> item = queue.head;
```

```
	while(item!=nullptr)
	{
		os << *item;
		item = item->GetNext();
	}
	return os;
}

template <class T> TIterator<TQueueItem<T>,T> TQueue<T>::begin(){
	return TIterator<TQueueItem<T>,T> (head);
}

template <class T> TIterator<TQueueItem<T>,T> TQueue<T>::end(){
	return TIterator<TQueueItem<T>,T> (nullptr);
}

template <class T> void TQueue<T>::Push(std::shared_ptr<T> &&item) {
	std::shared_ptr<TQueueItem<T>> other(new TQueueItem<T>(item));
	if (Empty())  {
		head = other;
		last = other;
	}
	else {
		last->SetNext(other);
		last = other;
	}
}

template <class T> bool TQueue<T>::Empty() {
	return head == nullptr;
}

template <class T> std::shared_ptr<T> TQueue<T>::Pop() {
	std::shared_ptr<T> result;
	if (head != nullptr) {
		result = head->GetFigure();
		head = head->GetNext();
	}
	return result;
}

template <class T> TQueue<T>::~TQueue() {
}

#include "Figure.h"
template class TQueue<Figure>;
template std::ostream& operator<<(std::ostream& os, const TQueue<Figure>&
stack);
```

TQueueItem.h

```
#ifndef TQUEUEITEM_H
#define TQUEUEITEM_H

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

template <class T> class TQueueItem {
public:
	TQueueItem(const std::shared_ptr<T>& figure);
	TQueueItem(const TQueueItem& orig);
	template <class A> friend std::ostream& operator<<(std::ostream& os,
const TQueueItem<A>& obj);

	std::shared_ptr<TQueueItem<T>> SetNext(std::shared_ptr<TQueueItem>
&next);
	std::shared_ptr<TQueueItem<T>> GetNext();
	std::shared_ptr<T> GetFigure() const;

	virtual ~TQueueItem();
private:
```

```
	std::shared_ptr<T> figure;
	std::shared_ptr<TQueueItem<T>> next;
};

#endif
```

TQueueItem.cpp

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

template <class T> TQueueItem<T>::TQueueItem(const std::shared_ptr<T>
&figure) {
	this->figure = figure;
	this->next = nullptr;
	std::cout << "Queue item: created" << "\n";
}

template <class T> TQueueItem<T>::TQueueItem(const TQueueItem& orig) {
	this->figure = orig.figure;
	this->next = orig.next;
	std::cout << "Queue item: copied" << "\n";
}

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

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

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" << "\n";
}

template <class A> std::ostream& operator<<(std::ostream& os, const
TQueueItem<A>& obj) {
	//os << "[" << *obj.figure << "]" << "\n";
	obj.figure->Print();
	return os;
}

#include "Figure.h"
template class TQueueItem<Figure>;
template std::ostream& operator<<(std::ostream& os, const
TQueueItem<Figure>&stack);
```

Main.cpp

```
#include <iostream>
#include <locale>
#include <memory>
#include "TQueue.h"

void Menu()
{
	std::cout << "1. Create queue\n";
	std::cout << "2. Add item\n";
	std::cout << "    3. Add trapeze\n";
	std::cout << "    4. Add rhombus\n";
	std::cout << "    5. Add pentagon\n";
	std::cout << "6. Delete item\n";
	std::cout << "7. Print queue\n";
	std::cout << "8. Menu\n";
	std::cout << "0. Exit\n";
```

```
	std::cout << "\n";
}

int main() {
	std::shared_ptr<TQueue<Figure>> q = nullptr;
	int action;
	Menu();
	std::cin >> action;

	while (action != 0)
	{
		if (action == 1 && q == nullptr){
			q = std::shared_ptr<TQueue<Figure>> (new TQueue<Figure>);
			std::cout << "Queue created\n";
		}
		//std::cout << q << " " << action << std::endl;
		if (action == 2 && q != nullptr) {
			int cnt;
			std::cin >> cnt;
			if (cnt == 3) {
				q->Push(std::shared_ptr<Figure> (new
Trapeze(std::cin)));
			}
			else if (cnt == 4) {
				q->Push(std::shared_ptr<Figure> (new
Rhombus(std::cin)));
			}
			else if (cnt == 5) {
				q->Push(std::shared_ptr<Figure> (new
Pentagon(std::cin)));
			}
		}
		if (action == 6 && q != nullptr){
			if (!q->Empty()){
				auto r = q->Pop();
				//std::cout << r << "\n";
			}
			else
				std::cout << "Queue is empty\n";
		}
		if (action == 7 && q != nullptr)
			//std::cout << *q;
			for (auto i: *q) {
				i->Print();
			}
		/*if (action == 5 && q != nullptr){
			delete q;
			std::cout << "Queue deleted\n";
			q = nullptr;
		}*/
		if (action == 8)
			Menu();
		std::cin >> action;
	}
}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
1. Create queue
2. Add item
	3. Add trapeze
	4. Add rhombus
	5. Add pentagon
6. Delete item
7. Print queue
8. Menu
0. Exit

1
Queue created
2
4
Enter the angle: 30
```

```
Enter the side: 1
Correct value
Queue item: created
2
3
Enter first side: 2
Enter second side: 2
Enter the height: 2
Correct value
Queue item: created
2
5
Enter the side: 3
Correct value
Queue item: created
7
Rhombus:
angle: 30 side: 1
Trapeze:
side 1:  2 side 2: 2 height: 2
Pentagon:
Side: 3
6
Queue item: deleted
Rhombus deleted
7
Trapeze:
side 1:  2 side 2: 2 height: 2
Pentagon:
Side: 3
6
Queue item: deleted
Trapeze deleted
7
Pentagon:
Side: 3
6
7
0
Queue item: deleted
Pentagon deleted
```

ВЫВОД

В 5 лабораторной работе мы добавляем итератор, что облегчает обращение к объектам и позволяет обращаться к ним по значениям.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики ипрограммирования

Отчет по лабораторной работе № 6

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Цысь Г.В
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

_____

Подпись студента

_____

Число
21.10.2018

Москва-2018

ЗАДАНИЕ

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР№5) спроектировать и разработать аллокатор памяти для динамической структуры данных.

Цель построения аллокатора – минимизация вызова операции malloc. Аллокатор должен выделять большие блоки памяти для хранения фигур и при создании новых фигур-объектов выделять место под объекты в этой памяти. Аллокатор должен хранить списки использованных/свободных блоков. Для хранения списка свободных блоков нужно применять динамическую структуру данных (контейнер 2-го уровня, согласно варианта задания). Для вызова аллокатора должны быть переопределены оператор new и delete у классов-фигур.

ОПИСАНИЕ

Класс шаблона описывает объект, который управляет выделением и освобождением памяти для массивов объектов типа типа. Объект класса распределителя — объект распределителя по умолчанию, заданный в конструкторы несколько классов шаблонов контейнера из стандартной библиотеки C++.

Все контейнеры библиотеки стандартных шаблонов имеют параметр шаблона, который по умолчанию распределителя. Создание контейнера с пользовательским распределителем дает возможность управлять выделением и освобождением памяти для элементов контейнера.

Например, объект распределителя может выделить память в закрытой куче или в общей памяти. Он также может выполнить оптимизацию для крупных или мелких объектов. Он может также указывать, посредством определения типов, которые он предоставляет, что доступ к элементам возможен только через специальные объекты доступа, управляющие общей памятью или выполняющие автоматическую сборку мусора. Таким образом, класс, который выделяет память с использованием объекта распределителя, должен использовать эти типы для объявления указателя и объектов ссылок, как это делают контейнеры в стандартной библиотеке C++.

ЛИСТИНГ ПРОГРАММЫ

TQueue.h

```
#ifndef TQUEUE_H
#define TQUEUE_H

#include "Trapeze.h"
#include "Pentagon.h"
#include "Rhombus.h"
#include "TQueueItem.h"
#include "TIterator.h"
#include <memory>

template <class T> class TQueue {
public:
	TQueue();
	TQueue(const TQueue& orig);

	void Push(std::shared_ptr<T> &&item);
	bool Empty();
	std::shared_ptr<T> Pop();

	TIterator<TQueueItem<T>,T> begin();
	TIterator<TQueueItem<T>,T> end();
```

```
      template <class A>friend std::ostream& operator<<(std::ostream& os,
const TQueue<A>& queue);
      virtual ~TQueue();
private:
      std::shared_ptr<TQueueItem<T>> head;
      std::shared_ptr<TQueueItem<T>> last;
};

#endif
```

TQueue.cpp

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

template <class T> TQueue<T>::TQueue() : head(nullptr), last(nullptr) {
}

template <class T> TQueue<T>::TQueue(const TQueue<T>& orig) {
    head = orig.head;
    last = orig.last;
}

template <class T> std::ostream& operator<<(std::ostream& os, const
TQueue<T>& queue) {
    std::shared_ptr<TQueueItem<T>> item = queue.head;

    while(item!=nullptr)
    {
        os << *item;
        item = item->GetNext();
    }
    return os;
}

template <class T> TIterator<TQueueItem<T>,T> TQueue<T>::begin(){
    return TIterator<TQueueItem<T>,T> (head);
}

template <class T> TIterator<TQueueItem<T>,T> TQueue<T>::end(){
    return TIterator<TQueueItem<T>,T> (nullptr);
}

template <class T> void TQueue<T>::Push(std::shared_ptr<T> &&item) {
    std::shared_ptr<TQueueItem<T> > other(new TQueueItem<T>(item));
    if (Empty())  {
        head = other;
        last = other;
    }
    else {
        last->SetNext(other);
        last = other;
    }
}

template <class T> bool TQueue<T>::Empty() {
    return head == nullptr;
}

template <class T> std::shared_ptr<T> TQueue<T>::Pop() {
    std::shared_ptr<T> result;
    if (head != nullptr) {
        result = head->GetFigure();
        head = head->GetNext();
    }
    return result;
}

template <class T> TQueue<T>::~TQueue() {
}

#include "Figure.h"
```

```
template class TQueue<Figure>;
template std::ostream& operator<<(std::ostream& os, const TQueue<Figure>&
stack);
```

TQueueItem.h

```
#ifndef TQUEUEITEM_H
#define TQUEUEITEM_H

#include "Trapeze.h"
#include <memory>
#include "TAllocationBlock.h"

template <class T> class TQueueItem {
public:
      TQueueItem(const std::shared_ptr<T>& figure);
      TQueueItem(const TQueueItem& orig);
      template <class A> friend std::ostream& operator<<(std::ostream& os,
const TQueueItem<A>& obj);

      std::shared_ptr<TQueueItem<T>> SetNext(std::shared_ptr<TQueueItem>
&next);
      std::shared_ptr<TQueueItem<T>> GetNext();
      std::shared_ptr<T> GetFigure() const;

    void* operator new (size_t size);
    void operator delete(void *p);

      virtual ~TQueueItem();
private:
      std::shared_ptr<T> figure;
      std::shared_ptr<TQueueItem<T> > next;
    static TAllocationBlock item_allocator;
};

#endif
```

TQueueItem.cpp

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

template <class T> TQueueItem<T>::TQueueItem(const std::shared_ptr<T>
&figure) {
    this->figure = figure;
    this->next = nullptr;
    std::cout << "Queue item: created" << "\n";
}

template <class T> TQueueItem<T>::TQueueItem(const TQueueItem& orig) {
    this->figure = orig.figure;
    this->next = orig.next;
    std::cout << "Queue item: copied" << "\n";
}

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

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

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" << "\n";
}
```

```
template <class A> std::ostream& operator<<(std::ostream& os, const
TQueueItem<A>& obj) {
    //os << "[" << *obj.figure << "]" << "\n";
    obj.figure->Print();
    return os;
}



template <class T>
TAllocationBlock TQueueItem<T>::item_allocator(sizeof(TQueueItem<T>), 100);


template <class T> void * TQueueItem<T>::operator new (size_t size) {
    return item_allocator.allocate();
}

template <class T> void TQueueItem<T>::operator delete(void *p) {
    item_allocator.deallocate(p);
}


#include "Figure.h"
template class TQueueItem<Figure>;
template std::ostream& operator<<(std::ostream& os, const
TQueueItem<Figure>&stack);
```

Tree.h

```
#ifndef TREE_H
#define TREE_H

#include <iostream>
#include <memory>
#include <string>

#include "TreeItem.h"

template <class T>
class Tree
{
public:
    Tree(T figure, int n);

    T Delete(std::string& path);

    bool Insert(T figure, std::string& path);

    void Delete();
    void Print();

    ~Tree();
private:
    std::shared_ptr <Node <T> > root;
    int n;
};


#endif
```

Tree.cpp

```
#include "Tree.h"

template <class T> Tree<T>::Tree(T figure, int n) {
    this->n = n;
    root = std::make_shared <Node<T> >(figure);
}

template <class T> T Tree<T>::Delete(std::string& path) {
    std::shared_ptr <Node<T> > curr = this->root;
    std::shared_ptr <Node<T> > prev = nullptr;
```

```
    int index = 0;
    while ( path[index] != '\0' ) {
        switch ( path[index] ) {
            case 'S':
                prev = curr;
                curr = curr->GetSon();
                break;
            case 'B':
                prev = curr;
                curr = curr->GetBro();
                break;
            default:
                std::cout << "Invalid path" << std::endl;
                return nullptr;
        }
        if ( curr == nullptr ) {
            std::cout << "Invalid path" << std::endl;
            return nullptr;
        }
        index++;
    }
    if ( curr->GetSon() ) {
        std::cout << "Current vertex is not a leaf" << std::endl;
        return nullptr;
    }
    if ( !prev || index == 0 ) {
        this->root = nullptr;
        return curr->GetFigure();
    }
    if ( prev->GetSon() == curr ) {
        prev->SetSon(curr->GetBro());
    }
    else {
        prev->SetBro(curr->GetBro());
    }
    return curr->GetFigure();
}

template <class T> bool Tree<T>::Insert(T figure, std::string& path) {
    std::shared_ptr <Node <T> > curr = this->root;
    int index = 0;
    while ( path[index] != '\0' ) {
        std::cout << path[index] << std::endl;
        switch ( path[index] ) {
            case 'S':
                curr = curr->GetSon();
                break;
            case 'B':
                curr = curr->GetBro();
                break;
            default:
                std::cout << "Invalid path" << std::endl;
                return false;
        }
        if ( curr == nullptr ) {
            std::cout << "Invalid path" << std::endl;
            return false;
        }
        index++;
    }
    int sons = curr->CountSons();
    if ( sons == this->n) {
        std::cout << "This node already has " << this->n << " sons" <<
std::endl;
        return false;
    }
    curr->AddSon( std::make_shared <Node <T> > (figure));
    return true;
}

template <class T> void Tree<T>::Print() {
```

```
    PrintNodes(this->root, 0);
}


template <class T> Tree<T>::~Tree() {

}

// template class Tree<int>;
template class Tree<void*>;
// template class Tree<char>;
```

TreeItem.h

```
#ifndef TREEITEM_H
#define TREEITEM_H

#include <iostream>
#include <iomanip>
#include <memory>

template <class T>
class Node {
public:
    Node(T figure);

    std::shared_ptr <Node <T> > GetSon();
    std::shared_ptr <Node <T> > GetBro();

    void SetSon(std::shared_ptr <Node<T> > newSon);
    void SetBro(std::shared_ptr <Node<T> > newBro);

    int CountSons();

    T GetFigure();

    void AddSon(std::shared_ptr <Node<T> > newSon);

    virtual ~Node();
private:
    T figure;
    std::shared_ptr <Node <T> > son;
    std::shared_ptr <Node <T> > brother;
};

template <class T> void PrintNodes(std::shared_ptr<Node<T> > curr, int
depth);

#endif
```

TreeItem.cpp

```
#include "TreeItem.h"

template <class T> Node<T>::Node(T figure) {
    this->figure = figure;
    this->brother = this->son = nullptr;
}

template <class T> std::shared_ptr <Node <T> > Node<T>::GetSon() {
    return this->son;
}

template <class T> std::shared_ptr <Node <T> > Node<T>::GetBro() {
    return this->brother;
}

template <class T> void Node<T>::SetSon(std::shared_ptr <Node<T> > newSon) {
    this->son = newSon;
}

template <class T> void Node<T>::SetBro(std::shared_ptr <Node<T> > newBro) {
    this->brother = newBro;
```

```
}

template <class T> int Node<T>::CountSons() {
    std::shared_ptr<Node<T>> curr = this->son;
    int count = 0;
    while ( curr != nullptr ) {
        curr = curr->GetBro();
        count++;
    }
    return count;
}

template <class T> T Node<T>::GetFigure() {
    return this->figure;
}


template <class T> void Node<T>::AddSon(std::shared_ptr <Node<T> > newSon) {
    if ( this->son == nullptr )
        this->son = newSon;
    else {
        std::shared_ptr<Node<T>> tmp = this->GetSon();
        while ( tmp->brother != nullptr )
            tmp = tmp->GetBro();
        tmp->brother = newSon;
    }
}

template <class T> void PrintNodes(std::shared_ptr <Node <T> > curr, int
depth) {
    std::cout << std::setw(5*depth) << (curr->GetFigure()) << std::endl;
    curr = curr->GetSon();
    depth++;
    while ( curr != nullptr ) {
        PrintNodes(curr, depth);
        curr = curr->GetBro();
    }
}

template <class T> Node<T>::~Node() {
}

// template class Node<int>;
// template void PrintNodes(std::shared_ptr<Node<int> > curr, int depth);

template class Node<void*>;
template void PrintNodes(std::shared_ptr<Node<void*> > curr, int depth);

// template class Node<char>;
// template void PrintNodes(std::shared_ptr<Node<char> > curr, int depth);
```

TAllocationBlock.h

```
#ifndef TALLOCATIONBLOCK_H
#define     TALLOCATIONBLOCK_H

#include <cstdlib>
#include "Tree.h"


class TAllocationBlock {
public:
    TAllocationBlock(size_t size,size_t count);
    void *allocate();
    void deallocate(void *pointer);
    bool has_free_blocks();

    virtual ~TAllocationBlock();
private:
    size_t _size;
    size_t _count;
```

```
    char  *_used_blocks;
    // void  **_free_blocks;
    std::shared_ptr <Tree <void*> > _free;

    size_t _free_count;
};

#endif      /* TALLOCATIONBLOCK_H */
```

TAllocationBlock.cpp

```
#include "TAllocationBlock.h"
#include <iostream>
#include <string>

TAllocationBlock::TAllocationBlock(size_t size,size_t count):
_size(size),_count(count)  {
    _used_blocks = (char*)malloc(_size*_count);
    _free = std::shared_ptr<Tree<void*> >(new Tree<void*> ((nullptr),
_count));
    std::string path = "\0";
    for (size_t i = 0; i < _count; i++) {
        _free->Insert(((void*)(_used_blocks + i * _size)), path);
    }
    _free_count = _count;
    std::cout << "TAllocationBlock: Memory init" << std::endl;
}

void *TAllocationBlock::allocate() {
    void *result = nullptr;

    if(_free_count > 0) {
        std::string path = "S";
        result = _free->Delete(path);
        _free_count--;
        std::cout << "TAllocationBlock: Allocate " << (_count - _free_count)
<< " of " << _count << std::endl;
    } else {
        std::cout << "TAllocationBlock: No memory exception :-)" <<
std::endl;
    }
    return result;
}

void TAllocationBlock::deallocate(void *pointer) {
    std::cout << "TAllocationBlock: Deallocate block "<< std::endl;
    std::string path = "\0";
    _free->Insert(pointer, path);
    _free_count ++;
}

bool TAllocationBlock::has_free_blocks() {
    return _free_count > 0;
}

TAllocationBlock::~TAllocationBlock() {

    if(_free_count<_count) std::cout << "TAllocationBlock: Memory leak?" <<
std::endl;
                    else  std::cout << "TAllocationBlock: Memory freed" <<
std::endl;
    free(_used_blocks);
}
```

TIterator.h

```
#ifndef TITERATOR_H
#define TITERATOR_H
#include <memory>
#include <iostream>
template <class node, class T>
class TIterator
{
  public:
    TIterator(std::shared_ptr<node> n)
    {
        node_ptr = n;
    }

    std::shared_ptr<T> operator*()
    {
        return node_ptr->GetFigure();
    }

    std::shared_ptr<T> operator->()
    {
        return node_ptr->GetFigure();
    }

    void operator++()
    {
        node_ptr = node_ptr->GetNext();
    }

    TIterator operator++(int)
    {
        TIterator iter(*this);
        ++(*this);
        return iter;
    }

    bool operator==(TIterator const &i)
    {
        return node_ptr == i.node_ptr;
    }

    bool operator!=(TIterator const &i)
    {
        return !(*this == i);
    }

  private:
    std::shared_ptr<node> node_ptr;
};
#endif
```

Main.cpp

```
#include <iostream>
#include <locale>
#include <memory>
#include "TQueue.h"

void Menu()
{
      std::cout << "1. Create queue\n";
      std::cout << "2. Add item\n";
      std::cout << "    3. Add trapeze\n";
      std::cout << "    4. Add rhombus\n";
      std::cout << "    5. Add pentagon\n";
      std::cout << "6. Delete item\n";
      std::cout << "7. Print queue\n";
      std::cout << "8. Menu\n";
      std::cout << "0. Exit\n";
      std::cout << "\n";
}
```

```
int main() {
	std::shared_ptr<TQueue<Figure>> q = nullptr;
	int action;
	Menu();
	std::cin >> action;

	while (action != 0)
	{
		if (action == 1 && q == nullptr){
			q = std::shared_ptr<TQueue<Figure>> (new TQueue<Figure>);
			std::cout << "Queue created\n";
		}
		//std::cout << q << " " << action << std::endl;
		if (action == 2 && q != nullptr) {
			int cnt;
			std::cin >> cnt;
			if (cnt == 3) {
				q->Push(std::shared_ptr<Figure> (new
Trapeze(std::cin)));
			}
			else if (cnt == 4) {
				q->Push(std::shared_ptr<Figure> (new
Rhombus(std::cin)));
			}
			else if (cnt == 5) {
				q->Push(std::shared_ptr<Figure> (new
Pentagon(std::cin)));
			}
		}
		if (action == 6 && q != nullptr){
			if (!q->Empty()){
				auto r = q->Pop();
				std::cout << r << "\n";
			}
			else
				std::cout << "Queue is empty\n";
		}
		if (action == 7 && q != nullptr)
			//std::cout << *q;
			for (auto i: *q) {
				i->Print();
			}
		if (action == 8)
			Menu();
		std::cin >> action;
	}

	std::shared_ptr<int> a(new int);
	int *c = a.get();
	*c = b;
	std::cout << *a << std::endl;*/

}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
TAllocationBlock: Memory init
1. Create queue
2. Add item
	3. Add trapeze
	4. Add rhombus
	5. Add pentagon
6. Delete item
7. Print queue
8. Menu
0. Exit

1
Queue created
2
5
Enter the side: 1
Correct value
```

```
TAllocationBlock: Allocate 1 of 100
Queue item: created
2
4
Enter the angle: 30
Enter the side: 2
Correct value
TAllocationBlock: Allocate 2 of 100
Queue item: created
2
3
Enter first side: 3
Enter second side: 3
Enter the height: 3
Correct value
TAllocationBlock: Allocate 3 of 100
Queue item: created
7
Pentagon:
Side: 1
Rhombus:
angle: 30 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
6
Queue item: deleted
TAllocationBlock: Deallocate block
0xbb3410
Pentagon deleted
7
Rhombus:
angle: 30 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
6
Queue item: deleted
TAllocationBlock: Deallocate block
0xbb34c0
Rhombus deleted
7
Trapeze:
side 1:  3 side 2: 3 height: 3
0
Queue item: deleted
Trapeze deleted
TAllocationBlock: Deallocate block
TAllocationBlock: Memory freed
```

ВЫВОД

В ходе 6 работы был я ознакомился и создал аллокатор памяти, который помогает оптимизировать выделение и освобождение памяти. Свободные блоки хранятся в аллокаторе в контейнере второго уровня, который представляет собой N-дерево.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики ипрограммирования

Отчет по лабораторной работе № 7

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Цысь Г.В
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

_____

Подпись студента

_____

Число
21.10.2018

Москва-2018

Лабораторная работа №7

ЗАДАНИЕ

Необходимо реализовать динамическую структуру данных – «Хранилище объектов» и алгоритм работы с ней. «Хранилище объектов» представляет собой контейнер (Контейнер 1-го уровня) – очередь.

Каждым элементом контейнера, в свою очередь, является динамической структурой данных одного из следующих видов (Контейнер 2-го уровня) – N-дерево.

ОПИСАНИЕ

Принцип открытости/закрытости (Open/Closed Principle) можно сформулировать так:

Сущности программы должны быть открыты для расширения, но закрыты для изменения.

Суть этого принципа состоит в том, что система должна быть построена таким образом, что все ее последующие изменения должны быть реализованы с помощью добавления нового кода, а не изменения уже существующего.

Противоречие является лишь кажущимся, поскольку термины соответствуют разным целевым установкам:

- Модуль называют открытым, если он еще доступен для расширения. Например, имеется возможность расширить множество операций в нем или добавить поля к его структурам данных.
- Модуль называют закрытым, если он доступен для использования другими модулями. Это означает, что модуль (его интерфейс — с точки зрения скрытия информации) уже имеет строго определенное окончательное описание. На уровне реализации закрытое состояние модуля означает, что модуль можно компилировать, сохранять в библиотеке и делать его доступным для использования другими модулями (его клиентами). На этапе проектирования или спецификации закрытие модуля означает, что он одобрен руководством, внесен в официальный репозиторий утвержденных программных элементов проекта — базу проекта (project baseline), и его интерфейс опубликован в интересах авторов других модулей.

ЛИСТИНГ ПРОГРАММЫ

TQueue.h

```
#ifndef TQUEUE_H
#define TQUEUE_H

#include "Trapeze.h"
#include "Pentagon.h"
#include "Rhombus.h"
#include "TQueueItem.h"
#include "TIterator.h"
#include <memory>
#include "Tree.h"

template <class T> class TQueue {
public:
	TQueue();
	TQueue(const TQueue& orig);

	void Push(std::shared_ptr<T> &&item);
```

```
      bool Empty();
      std::shared_ptr<T> Pop();

      TIterator<TQueueItem<T>,T> begin();
      TIterator<TQueueItem<T>,T> end();

      template <class A>friend std::ostream& operator<<(std::ostream& os,
const TQueue<A>& queue);
      virtual ~TQueue();
private:
      std::shared_ptr<TQueueItem<T>> head;
      std::shared_ptr<TQueueItem<T>> last;
};

#endif
```

TQueue.cpp

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

template <class T> TQueue<T>::TQueue() : head(nullptr), last(nullptr) {
}

template <class T> TQueue<T>::TQueue(const TQueue<T>& orig) {
    head = orig.head;
    last = orig.last;
}

template <class T> std::ostream& operator<<(std::ostream& os, const
TQueue<T>& queue) {
    std::shared_ptr<TQueueItem<T>> item = queue.head;

    while(item!=nullptr)
    {
        os << *item;
        item = item->GetNext();
    }
    return os;
}

template <class T> TIterator<TQueueItem<T>,T> TQueue<T>::begin(){
    return TIterator<TQueueItem<T>,T> (head);
}

template <class T> TIterator<TQueueItem<T>,T> TQueue<T>::end(){
    return TIterator<TQueueItem<T>,T> (nullptr);
}

template <class T> void TQueue<T>::Push(std::shared_ptr<T> &&item) {
    std::shared_ptr<TQueueItem<T> > other(new TQueueItem<T>(item));
    if (Empty())  {
        head = other;
        last = other;
    }
    else {
        last->SetNext(other);
        last = other;
    }
}

template <class T> bool TQueue<T>::Empty() {
    return head == nullptr;
}

template <class T> std::shared_ptr<T> TQueue<T>::Pop() {
    std::shared_ptr<T> result;
    if (head != nullptr) {
        result = head->GetFigure();
        head = head->GetNext();
    }
    return result;
```

```
}

template <class T> TQueue<T>::~TQueue() {
}

#include "Figure.h"
template class TQueue<Figure>;
template std::ostream& operator<<(std::ostream& os, const TQueue<Figure>&
stack);

template class TQueue<Tree<std::shared_ptr<Figure> > >;
```

TQueueItem.h

```
#ifndef TQUEUEITEM_H
#define TQUEUEITEM_H

#include "Trapeze.h"
#include <memory>
#include "TAllocationBlock.h"

template <class T> class TQueueItem {
public:
      TQueueItem(const std::shared_ptr<T>& figure);
      TQueueItem(const TQueueItem& orig);
      template <class A> friend std::ostream& operator<<(std::ostream& os,
const TQueueItem<A>& obj);

      std::shared_ptr<TQueueItem<T>> SetNext(std::shared_ptr<TQueueItem>
&next);
      std::shared_ptr<TQueueItem<T>> GetNext();
      std::shared_ptr<T> GetFigure() const;

    void* operator new (size_t size);
    void operator delete(void *p);

      virtual ~TQueueItem();
private:
      std::shared_ptr<T> figure;
      std::shared_ptr<TQueueItem<T> > next;
    static TAllocationBlock item_allocator;
};

#endif
```

TQueueItem.cpp

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

template <class T> TQueueItem<T>::TQueueItem(const std::shared_ptr<T>
&figure) {
    this->figure = figure;
    this->next = nullptr;
    std::cout << "Queue item: created" << "\n";
}

template <class T> TQueueItem<T>::TQueueItem(const TQueueItem& orig) {
    this->figure = orig.figure;
    this->next = orig.next;
    std::cout << "Queue item: copied" << "\n";
}

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

template <class T> std::shared_ptr<T> TQueueItem<T>::GetFigure() const {
    return this->figure;
}
```

```
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" << "\n";
}

template <class A> std::ostream& operator<<(std::ostream& os, const
TQueueItem<A>& obj) {
    //os << "[" << *obj.figure << "]" << "\n";
    obj.figure->Print();
    return os;
}



template <class T>
TAllocationBlock TQueueItem<T>::item_allocator(sizeof(TQueueItem<T>), 100);


template <class T> void * TQueueItem<T>::operator new (size_t size) {
    return item_allocator.allocate();
}

template <class T> void TQueueItem<T>::operator delete(void *p) {
    item_allocator.deallocate(p);
}


#include "Figure.h"
template class TQueueItem<Figure>;
template std::ostream& operator<<(std::ostream& os, const
TQueueItem<Figure>&stack);

template class TQueueItem<Tree<std::shared_ptr<Figure> > >;
template std::ostream& operator<<(std::ostream& os, const
TQueueItem<Tree<std::shared_ptr<Figure> > >&stack);
```

TAllocationBlock.h

```
#ifndef TALLOCATIONBLOCK_H
#define     TALLOCATIONBLOCK_H

#include <cstdlib>
#include "Tree.h"


class TAllocationBlock {
public:
    TAllocationBlock(size_t size,size_t count);
    void *allocate();
    void deallocate(void *pointer);
    bool has_free_blocks();

    virtual ~TAllocationBlock();
private:
    size_t _size;
    size_t _count;

    char  *_used_blocks;
    // void  **_free_blocks;
    std::shared_ptr <Tree <void*> > _free;

    size_t _free_count;
};

#endif      /* TALLOCATIONBLOCK_H */
```

TAllocationBlock.cpp

```
#include "TAllocationBlock.h"
#include <iostream>
#include <string>

TAllocationBlock::TAllocationBlock(size_t size,size_t count):
_size(size),_count(count)  {
    _used_blocks = (char*)malloc(_size*_count);
    _free = std::shared_ptr<Tree<void*> >(new Tree<void*> ((nullptr),
_count));
    std::string path = "\0";
    for (size_t i = 0; i < _count; i++) {
        _free->Insert(((void*)(_used_blocks + i * _size)), path);
    }
    _free_count = _count;
    std::cout << "TAllocationBlock: Memory init" << std::endl;
}

void *TAllocationBlock::allocate() {
    void *result = nullptr;

    if(_free_count > 0) {
        std::string path = "S";
        result = _free->Delete(path);
        _free_count--;
        std::cout << "TAllocationBlock: Allocate " << (_count - _free_count)
<< " of " << _count << std::endl;
    } else {
        std::cout << "TAllocationBlock: No memory exception :-)" <<
std::endl;
    }
    return result;
}

void TAllocationBlock::deallocate(void *pointer) {
    std::cout << "TAllocationBlock: Deallocate block "<< std::endl;
    std::string path = "\0";
    _free->Insert(pointer, path);
    _free_count ++;
}

bool TAllocationBlock::has_free_blocks() {
    return _free_count > 0;
}

TAllocationBlock::~TAllocationBlock() {

    if(_free_count<_count) std::cout << "TAllocationBlock: Memory leak?" <<
std::endl;
                    else  std::cout << "TAllocationBlock: Memory freed" <<
std::endl;
    free(_used_blocks);
}
```

TIterator.h

```
#ifndef TITERATOR_H
#define TITERATOR_H
#include <memory>
#include <iostream>
template <class node, class T>
class TIterator
{
  public:
    TIterator(std::shared_ptr<node> n)
    {
        node_ptr = n;
    }
```

```
    std::shared_ptr<T> operator*()
    {
        return node_ptr->GetFigure();
    }

    std::shared_ptr<T> operator->()
    {
        return node_ptr->GetFigure();
    }

    void operator++()
    {
        node_ptr = node_ptr->GetNext();
    }

    TIterator operator++(int)
    {
        TIterator iter(*this);
        ++(*this);
        return iter;
    }

    bool operator==(TIterator const &i)
    {
        return node_ptr == i.node_ptr;
    }

    bool operator!=(TIterator const &i)
    {
        return !(*this == i);
    }

  private:
    std::shared_ptr<node> node_ptr;
};
#endif
```

Tree.h

```
#ifndef TREE_H
#define TREE_H

#include <iostream>
#include <memory>
#include <string>

#include "TreeItem.h"
#include "Figure.h"

template <class T>
class Tree
{
public:
    Tree(T figure, int n);

    T Delete(std::string& path);

    bool Insert(T figure, std::string& path);

    void Delete();
    void Print();

    ~Tree();
private:
    std::shared_ptr <Node <T> > root;
    int n;
};


#endif
```

Tree.cpp

```
#include "Tree.h"

template <class T> Tree<T>::Tree(T figure, int n) {
    this->n = n;
    root = std::make_shared <Node<T> >(figure);
}

template <class T> T Tree<T>::Delete(std::string& path) {
    std::shared_ptr <Node<T> > curr = this->root;
    std::shared_ptr <Node<T> > prev = nullptr;
    int index = 0;
    while ( path[index] != '\0' ) {
        switch ( path[index] ) {
            case 'S':
                prev = curr;
                curr = curr->GetSon();
                break;
            case 'B':
                prev = curr;
                curr = curr->GetBro();
                break;
            default:
                std::cout << "Invalid path" << std::endl;
                return nullptr;
        }
        if ( curr == nullptr ) {
            std::cout << "Invalid path" << std::endl;
            return nullptr;
        }
        index++;
    }
    if ( curr->GetSon() ) {
        std::cout << "Current vertex is not a leaf" << std::endl;
        return nullptr;
    }
    if ( !prev || index == 0 ) {
        this->root = nullptr;
        return curr->GetFigure();
    }
    if ( prev->GetSon() == curr ) {
        prev->SetSon(curr->GetBro());
    }
    else {
        prev->SetBro(curr->GetBro());
    }
    return curr->GetFigure();
}

template <class T> bool Tree<T>::Insert(T figure, std::string& path) {
    std::shared_ptr <Node <T> > curr = this->root;
    int index = 0;
    while ( path[index] != '\0' ) {
        switch ( path[index] ) {
            case 'S':
                curr = curr->GetSon();
                break;
            case 'B':
                curr = curr->GetBro();
                break;
            default:
                std::cout << "Invalid path" << std::endl;
                return false;
        }
        if ( curr == nullptr ) {
            std::cout << "Invalid path" << std::endl;
            return false;
        }
```

```
        index++;
    }
    int sons = curr->CountSons();
    if ( sons == this->n) {
        std::cout << "This node already has " << this->n << " sons" <<
std::endl;
        return false;
    }
    curr->AddSon( std::make_shared <Node <T> > (figure));
    return true;
}

template <class T> void Tree<T>::Print() {
    PrintNodes(this->root, 0);
}


template <class T> Tree<T>::~Tree() {

}

template class Tree<void*>;
template class Tree<std::shared_ptr<Figure> >;
```

TreeItem.h

```
#ifndef TREEITEM_H
#define TREEITEM_H

#include <iostream>
#include <iomanip>
#include <memory>
#include "Figure.h"

template <class T>
class Node {
public:
    Node(T figure);

    std::shared_ptr <Node <T> > GetSon();
    std::shared_ptr <Node <T> > GetBro();

    void SetSon(std::shared_ptr <Node<T> > newSon);
    void SetBro(std::shared_ptr <Node<T> > newBro);

    int CountSons();

    T GetFigure();

    void AddSon(std::shared_ptr <Node<T> > newSon);

    virtual ~Node();
private:
    T figure;
    std::shared_ptr <Node <T> > son;
    std::shared_ptr <Node <T> > brother;
};

template <class T> void PrintNodes(std::shared_ptr<Node<T> > curr, int
depth);
template <class T> void PrintNodes(std::shared_ptr<Node<std::shared_ptr<T> >
> curr, int depth);

#endif
```

TreeItem.cpp

```
#include "TreeItem.h"

template <class T> Node<T>::Node(T figure) {
    this->figure = figure;
```

```
    this->brother = this->son = nullptr;
}

template <class T> std::shared_ptr <Node <T> > Node<T>::GetSon() {
    return this->son;
}

template <class T> std::shared_ptr <Node <T> > Node<T>::GetBro() {
    return this->brother;
}

template <class T> void Node<T>::SetSon(std::shared_ptr <Node<T> > newSon) {
    this->son = newSon;
}

template <class T> void Node<T>::SetBro(std::shared_ptr <Node<T> > newBro) {
    this->brother = newBro;
}

template <class T> int Node<T>::CountSons() {
    std::shared_ptr<Node<T>> curr = this->son;
    int count = 0;
    while ( curr != nullptr ) {
        curr = curr->GetBro();
        count++;
    }
    return count;
}

template <class T> T Node<T>::GetFigure() {
    return this->figure;
}


template <class T> void Node<T>::AddSon(std::shared_ptr <Node<T> > newSon) {
    if ( this->son == nullptr )
        this->son = newSon;
    else {
        std::shared_ptr<Node<T>> tmp = this->GetSon();
        while ( tmp->brother != nullptr )
            tmp = tmp->GetBro();
        tmp->brother = newSon;
    }
}


template <class T> void PrintNodes(std::shared_ptr <Node <std::shared_ptr<T>
> > curr, int depth) {
    std::cout << std::setw(5*depth) << std::endl;
    curr->GetFigure()->Print();
    curr = curr->GetSon();
    depth++;
    while ( curr != nullptr ) {
        PrintNodes(curr, depth);
        curr = curr->GetBro();
    }
}

template <class T> void PrintNodes(std::shared_ptr <Node <T> > curr, int
depth) {
    std::cout << std::setw(5*depth) << (curr->GetFigure()) << std::endl;
    curr = curr->GetSon();
    depth++;
    while ( curr != nullptr ) {
        PrintNodes(curr, depth);
        curr = curr->GetBro();
    }
}

template <class T> Node<T>::~Node() {
}
```

```
// template class Node<int>;
// template void PrintNodes(std::shared_ptr<Node<int> > curr, int depth);

template class Node<void*>;
template void PrintNodes(std::shared_ptr<Node<void*> > curr, int depth);

template class Node<std::shared_ptr<Figure> >;
template void PrintNodes(std::shared_ptr<Node<std::shared_ptr<Figure> > >
curr, int depth);
```

Main.cpp

```
#include <iostream>
#include <locale>
#include <memory>
#include "TQueue.h"
#include "Tree.h"
#include <string>

void Menu()
{
    std::cout << "1. Create queue\n";
    std::cout << "2. Add item\n";
    std::cout << "  3. Add trapeze\n";
    std::cout << "  4. Add rhombus\n";
    std::cout << "  5. Add pentagon\n";
    std::cout << "6. Delete item\n";
    std::cout << "7. Print queue\n";
    std::cout << "8. Menu\n";
    std::cout << "9. Create Tree\n";
    std::cout << "10. Add Tree\n";
    std::cout << "0. Exit\n";
    std::cout << "\n";
}

int main() {
    std::shared_ptr<TQueue<Tree<std::shared_ptr<Figure> > > > queue =
nullptr;
    std::shared_ptr<Tree<std::shared_ptr<Figure> > > t = nullptr;
    int action;
    Menu();

    while (std::cin >> action && action != 0) {
        std::cin.clear();
        std::cin.sync();
        if (action == 1 && queue == nullptr){
            queue = std::shared_ptr<TQueue<Tree<std::shared_ptr<Figure> > > >
(new TQueue<Tree<std::shared_ptr<Figure> > >);
            std::cout << "Queue created" << std::endl;
        }
        else if (action == 2 && t != nullptr) {
            int cnt;
            std::cin >> cnt;
            if (cnt == 3) {
                std::string path = "\0";
                std::cout << "Enter path: " << std::endl;
                std::cin >> path;
                if (path[0] != 'S' && path[0] != 'B') {
                    path = "\0";
                }
                t->Insert(std::shared_ptr<Figure> (new Trapeze(std::cin)),
path);
            }
            else if (cnt == 4) {
                std::string path = "\0";
                std::cout << "Enter path: " << std::endl;
                std::cin >> path;
                if (path[0] != 'S' && path[0] != 'B') {
                    path = "\0";
                }
```

```
                t->Insert(std::shared_ptr<Figure> (new Rhombus(std::cin)),
path);
            }
            else if (cnt == 5) {
                std::string path = "\0";
                std::cout << "Enter path: " << std::endl;
                std::cin >> path;
                if (path[0] != 'S' && path[0] != 'B') {
                    path = "\0";
                }
                t->Insert(std::shared_ptr<Figure> (new Pentagon(std::cin)),
path);
            }
        }
        else if (action == 6 && queue != nullptr){
            if (!queue->Empty()){
                auto r = queue->Pop();
                r->Print();
            }
            else {
                std::cout << "Queue is empty\n";
            }
        }
        else if (action == 7 && queue != nullptr) {
            for (auto i: *queue) {
                i->Print();
            }
        }
        else if (action == 8) {
            Menu();
        }
        else if (action == 9) {
            std::cout << "  1 Trapeze\n  2 Rhombus\n  3 Pentagon" <<
std::endl;
            int cnt;
            std::cin >> cnt;
            std::shared_ptr<Figure> f;
            if (cnt == 1) {
                f = std::shared_ptr<Figure>(new Trapeze(std::cin));
            }
            else if (cnt == 2) {
                f = std::shared_ptr<Figure>(new Rhombus(std::cin));
            }
            else if (cnt == 3) {
                f = std::shared_ptr<Figure>(new Pentagon(std::cin));
            } else {
                continue;
            }

            t = std::shared_ptr<Tree<std::shared_ptr<Figure> > > (new
Tree<std::shared_ptr<Figure> > (f, 4) );
            std::cout << "Tree created" << std::endl;
        }
        else if (action == 10 && queue != nullptr && t != nullptr) {
            queue->Push(std::shared_ptr<Tree<std::shared_ptr<Figure> > >
(t));
        }
    }
}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
TAllocationBlock: Memory init
TAllocationBlock: Memory init
1. Create queue
2. Add item
  3. Add trapeze
  4. Add rhombus
  5. Add pentagon
6. Delete item
7. Print queue
8. Menu
```

```
9. Create Tree
10. Add Tree
0. Exit

9
  1 Trapeze
  2 Rhombus
  3 Pentagon
3
Enter the side: 1
Correct value
Tree created
2
3
Enter path:
L
Enter first side: 2
Enter second side: 2
Enter the height: 2
Correct value
2
3
Enter path:
L
Enter first side: 3
Enter second side: 3
Enter the height: 3
Correct value
2
4
Enter path:
S
Enter the angle: 30
Enter the side: 4
Correct value
2
5
Enter path:
SB
Enter the side: 5
Correct value
1
Queue created
10
TAllocationBlock: Allocate 1 of 100
Queue item: created
9
  1 Trapeze
  2 Rhombus
  3 Pentagon
1
Enter first side: 1
Enter second side: 1
Enter the height: 1
Correct value
Tree created
2
4
Enter path:
L
Enter the angle: 30
Enter the side: 2
Correct value
2
5
Enter path:
L
Enter the side: 3
Correct value
2
4
Enter path:
SB
Enter the angle: 30
Enter the side: 4
Correct value
2
5
Enter path:
```

```
SB
Enter the side: 5
Correct value
10
TAllocationBlock: Allocate 2 of 100
Queue item: created
7

Pentagon:
Side: 1

Trapeze:
side 1:  2 side 2: 2 height: 2

 Rhombus:
angle: 30 side: 4

Trapeze:
side 1:  3 side 2: 3 height: 3

Pentagon:
Side: 5

Trapeze:
side 1:  1 side 2: 1 height: 1

Rhombus:
angle: 30 side: 2

Pentagon:
Side: 3

 Rhombus:
angle: 30 side: 4

Pentagon:
Side: 5
6
Queue item: deleted
TAllocationBlock: Deallocate block

Pentagon:
Side: 1

Trapeze:
side 1:  2 side 2: 2 height: 2

 Rhombus:
angle: 30 side: 4

Trapeze:
side 1:  3 side 2: 3 height: 3

Pentagon:
Side: 5
Pentagon deleted
Trapeze deleted
Rhombus deleted
Trapeze deleted
Pentagon deleted
7
Trapeze:
side 1:  1 side 2: 1 height: 1

Rhombus:
angle: 30 side: 2

Pentagon:
Side: 3

 Rhombus:
angle: 30 side: 4

Pentagon:
Side: 5

0
Queue item: deleted
Pentagon deleted
```

```
Rhombus deleted
Pentagon deleted
Rhombus deleted
Trapeze deleted
TAllocationBlock: Deallocate block
TAllocationBlock: Memory freed
TAllocationBlock: Memory freed
```

ВЫВОД

В седьмой лабораторной работе создается программа, которая позволяет хранить контейнер в контейнере, в котором хранятся фигуры. При этом фигуры должны быть отсортированы по площади. Если контейнер второго уровня освобождается, то он удаляется. Эта лабораторная работа показалась мне наиболее сложной из тех, которые мне приходилось выполнять по ООП, но во время её выполнения я узнал очень много нового.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики ипрограммирования

Отчет по лабораторной работе № 8

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Цысь Г.В
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

_____

Подпись студента

_____

Число
21.10.2018

Москва-2018

ЗАДАНИЕ

Используя структуры данных, разработанные для лабораторной работы №6 (контейнер первого уровня и классы-фигуры) разработать алгоритм быстрой сортировки для класса-контейнера .

Необходимо разработать два вида алгоритма:

- Обычный, без параллельных вызовов.
- С использованием параллельных вызовов. В этом случае, каждый рекурсивный вызов сортировки должен создаваться в отдельном потоке.

Для создания потоков использовать механизмы:

- Future
- packaged_task/async

Для обеспечения потоко-безопасности структур данных использовать:

- mutex
- lock_guard

ОПИСАНИЕ

Параллельное программирование – это техника программирования, которая использует преимущества многоядерных или многопроцессорных компьютеров. Параллельное программирование может быть сложным, которое включает в себя черты более традиционного или последовательного программирования, но в параллельном имеются три дополнительных этапа:

- определение параллелизма: анализ задачи с целью выделить подзадачи, которые могут выполняться одновременно.
- Выявление параллелизма: изменение структуры задачи таким образом, чтобы можно было эффективно выполнять задачи.
- Выражение параллелизма: реализация параллельного алгоритма в исходном коде с помощью системы обозначений параллельного программирования.

ЛИСТИНГ ПРОГРАММЫ

TIterator.h

```
#ifndef TITERATOR_H
#define TITERATOR_H
#include <memory>
#include <iostream>
template <class node, class T>
class TIterator
{
  public:
    TIterator(std::shared_ptr<node> n)
    {
        node_ptr = n;
    }

    std::shared_ptr<T> operator*()
    {
        return node_ptr->GetFigure();
    }

    std::shared_ptr<T> operator->()
    {
        return node_ptr->GetFigure();
    }

    void operator++()
    {
        node_ptr = node_ptr->GetNext();
```

```
    }

    TIterator operator++(int)
    {
        TIterator iter(*this);
        ++(*this);
        return iter;
    }

    bool operator==(TIterator const &i)
    {
        return node_ptr == i.node_ptr;
    }

    bool operator!=(TIterator const &i)
    {
        return !(*this == i);
    }

  private:
    std::shared_ptr<node> node_ptr;
};
#endif
```

TQueue.h

```
#ifndef TQUEUE_H
#define TQUEUE_H

#include <memory>

#include "TIterator.h"
#include "TQueueItem.h"
#include "Figure.h"

template <class T>  class TQueue
{
public:
    TQueue();
    bool Empty();
    int Size();
    void Push(std::shared_ptr <T>&& figure);
    void Sort();
    void ParallelSort();
    std::shared_ptr <T> Pop();
    TIterator<TQueueItem<T>,T> begin();
    TIterator<TQueueItem<T>,T> end();
    template <class A> friend std::ostream& operator << (std::ostream&
os,const TQueue <A>& queue);
    virtual ~TQueue();
protected:
    std::shared_ptr < TQueueItem<T> > first;
    std::shared_ptr < TQueueItem<T> > last;
    int size;
};

#endif // TQUEUE_H
```

TQueue.cpp

```
#include <iostream>
#include <thread>
#include <memory>

#include "TQueue.h"

template <class T> TQueue<T>::TQueue()
{
    this->first = this->last = nullptr;
    this->size = 0;
```

```
}

template <class T> bool TQueue<T>::Empty()
{
    return this->first == nullptr;
}

template <class T> int TQueue<T>::Size() {
    return this->size;
}

template <class T> void TQueue<T>::Push(std::shared_ptr<T> &&figure)
{
    std::shared_ptr < TQueueItem<T> > other(std::make_shared < TQueueItem<T>
> (figure));
    if ( this->first == nullptr )
        this->first = this->last = other;
    else
    {
        this->last->SetNext(other);
        this->last = this->last->GetNext();
    }
    this->size++;
}

template <class T> std::shared_ptr <T> TQueue<T>::Pop()
{
    std::shared_ptr <T> result;
    if ( this->first != nullptr )
    {
        result = this->first->GetFigure();
        this->first = this->first->GetNext();
        if (this->first == nullptr)
            this->last = nullptr;
        this->size--;
    }
    return result;
}

template <class T> std::ostream& operator << (std::ostream& os,const
TQueue<T>& queue)
{
    std::shared_ptr < TQueueItem<T> > item = queue.first;
    while ( item != nullptr )
    {
        os << item << std::endl;
        item = item->GetNext();
    }
    return os;
}

template <class T> TQueue<T>::~TQueue()
{
   /* std::cout << "Queue has been deleted" << std::endl;*/
}

template <class T>  TIterator<TQueueItem<T>,T> TQueue<T>::begin() {
    return TIterator<TQueueItem<T>,T>(first);
}

template <class T> TIterator<TQueueItem<T>,T> TQueue<T>::end() {
      TIterator<TQueueItem<T>,T> a(last);
      ++a;
      return a;
}

template <class T> void TQueue<T>::Sort() {
    if ( Empty() )
        return;
    std::shared_ptr <T> middle = Pop();
    TQueue <T> left;
```

```
    TQueue <T> right;
    while ( !Empty() ) {
        std::shared_ptr <T> curr = Pop();
        if ( middle->Square() > curr->Square() )
            left.Push(std::move(curr));
        else
            right.Push(std::move(curr));
    }

    left.Sort();
    right.Sort();

    while( !left.Empty() )
        Push(std::move(left.Pop()));

    Push(std::move(middle));

    while( !right.Empty() )
        Push(right.Pop());
}

template <class T> void TQueue<T>::ParallelSort() {
    if ( Size() <= 1 )
        return;
    std::shared_ptr <T> middle = Pop();
    TQueue <T> left;
    TQueue <T> right;
    while ( !Empty() ) {
        std::shared_ptr <T> curr = Pop();
        if ( middle->Square() > curr->Square() )
            left.Push(std::move(curr));
        else
            right.Push(std::move(curr));
    }
    //std::cout << std::this_thread::get_id() << std::endl;
    std::thread fst(std::move([&left](){left.ParallelSort();}));
    std::thread snd(std::move([&right](){right.ParallelSort();}));
    //std::this_thread::yield();
    fst.join();
    snd.join();
    while( !left.Empty() )
        Push(left.Pop());

    Push(std::move(middle));


    while( !right.Empty() )
        Push(right.Pop());
}

template class TQueue <Figure>;
//template class TQueue <TQueue <Figure>>;
//template class TQueue <void*>;
template std::ostream& operator<<(std::ostream& os, const  TQueue<Figure>&
queue);
```

TQueueItem.h

```
#ifndef TQUEUEITEM_H
#define TQUEUEITEM_H

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

template <class T> class TQueueItem {
public:
      TQueueItem(const std::shared_ptr<T>& figure);
      TQueueItem(const TQueueItem& orig);
      template <class A> friend std::ostream& operator<<(std::ostream& os,
const TQueueItem<A>& obj);
```

```
      std::shared_ptr<TQueueItem<T>> SetNext(std::shared_ptr<TQueueItem>
&next);
      std::shared_ptr<TQueueItem<T>> GetNext();
      std::shared_ptr<T> GetFigure() const;

    //void* operator new (size_t size);
    //void operator delete(void *p);

      virtual ~TQueueItem();
private:
      std::shared_ptr<T> figure;
      std::shared_ptr<TQueueItem<T> > next;
};

#endif
```

TQueueItem.cpp

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

template <class T> TQueueItem<T>::TQueueItem(const std::shared_ptr<T>
&figure) {
    this->figure = figure;
    this->next = nullptr;
    std::cout << "Queue item: created" << "\n";
}

template <class T> TQueueItem<T>::TQueueItem(const TQueueItem& orig) {
    this->figure = orig.figure;
    this->next = orig.next;
    std::cout << "Queue item: copied" << "\n";
}

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

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

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" << "\n";
}

template <class A> std::ostream& operator<<(std::ostream& os, const
TQueueItem<A>& obj) {
    //os << "[" << *obj.figure << "]" << "\n";
    obj.figure->Print();
    return os;
}



/*template <class T>
TAllocationBlock TQueueItem<T>::item_allocator(sizeof(TQueueItem<T>), 100);


template <class T> void * TQueueItem<T>::operator new (size_t size) {
    return item_allocator.allocate();
}

template <class T> void TQueueItem<T>::operator delete(void *p) {
    item_allocator.deallocate(p);
```

```
}*/


#include "Figure.h"
template class TQueueItem<Figure>;
template std::ostream& operator<<(std::ostream& os, const
TQueueItem<Figure>&stack);
```

Main.cpp

```
#include <iostream>
#include <cstdlib>
#include <vector>
#include <chrono>
#include <random>

#include "TQueue.h"
#include "TQueueItem.h"
#include "TIterator.h"
#include "Figure.h"
#include "Rhombus.h"
#include "Pentagon.h"
#include "Trapeze.h"
/*#include "RemoveCriteriaBySquare.h"
#include "RemoveCriteriaRhombus.h"*/

void menu()
{
    std::cout << "1. Push rhombus in queue"  << std::endl;
    std::cout << "2. Push pentagon in queue"  << std::endl;
    std::cout << "3. Push trapeze in queue"  << std::endl;
    std::cout << "4. Pop element"  << std::endl;
    std::cout << "5. Print queue" << std::endl;
    std::cout << "6. Menu"  << std::endl;
    std::cout << "7. Exit"  << std::endl;
}
/*void TestQueue() {
    Queue <TQueue<Figure>, Figure> q;
    char cmd;
    while ( std::cin >> cmd ) {
        switch (cmd) {
        case '1':
            q.InsertSubitem(std::shared_ptr <Figure> (new
Rhombus(std::cin)));
            break;
        case '2':
            q.InsertSubitem(std::shared_ptr <Figure> (new
Pentagon(std::cin)));
            break;
        case '3':
            q.InsertSubitem(std::shared_ptr <Figure> (new
Hexagon(std::cin)));
            break;
        case '4':
            double square;
            std::cin >> square;
            q.DeleteSubitems(std::shared_ptr <IRemoveCriteria<Figure>> (new
RemoveCriteriaBySquare<Figure>(square)));
            break;
        case '5':
            q.DeleteSubitems(std::shared_ptr <IRemoveCriteria<Figure>> (new
RemoveCriteriaRhombus<Figure>()));
            break;
        case 'p':
            std::cout << q << std::endl;
            break;
        default:
            break;
        }
    }
}*/
```

```
void TestSort() {
    int numTests;
    std::cin >> numTests;
    std::default_random_engine generator;
    std::uniform_int_distribution<int> distribution1(1, 100);
    std::uniform_int_distribution<int> distribution2(1, 100);
    generator.seed(time(nullptr));
    TQueue <Figure> q1;
    TQueue <Figure> q2;
    for (int i = 0; i < numTests; i++) {
        int side = distribution1(generator);
        int angle = distribution2(generator);
        q1.Push(std::shared_ptr <Figure> (std::make_shared <Rhombus>
(side,angle)));
        q2.Push(std::shared_ptr <Figure> (std::make_shared <Rhombus>
(side,angle)));
    }
    auto begin = std::chrono::high_resolution_clock::now();
    q1.Sort();
    for ( TIterator <TQueueItem <Figure>, Figure> i = q1.begin(); i !=
q1.end(); i++){
            (*i)->Print();
            std::cout << (*i)->Square() << std::endl;}
    auto end = std::chrono::high_resolution_clock::now();
    std::cout << "Sort:" <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count() <<
std::endl;
    begin = std::chrono::high_resolution_clock::now();
    q2.ParallelSort();
    for ( TIterator <TQueueItem <Figure>, Figure> i = q2.begin(); i !=
q2.end(); i++){
      (*i)->Print();
      std::cout << (*i)->Square() << std::endl;}
    end = std::chrono::high_resolution_clock::now();
    std::cout << "ParallelSort:" <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count() <<
std::endl;
}

int main(void)
{
    TestSort();
    //TestQueue();
    TQueue<Figure> q;
    char cmd;
    menu();
    while ( std::cin >> cmd )
    {
        switch (cmd) {
        case '1':
            q.Push(std::shared_ptr <Figure> (std::make_shared <Rhombus>
(std::cin)));// new Rhombus(std::cin)));
            break;
        case '2':
            q.Push(std::shared_ptr <Figure> (std::make_shared <Pentagon>
(std::cin)));// new Pentagon(std::cin)));
            break;
        case '3':
            q.Push(std::shared_ptr <Figure> (std::make_shared <Trapeze>
(std::cin)));// new Trapeze(std::cin)));
            break;
        case '4':
            q.Pop()->Print();
            break;
        case '5':
            for ( TIterator <TQueueItem <Figure>, Figure> i = q.begin(); i !=
q.end(); i++)
                (*i)->Print();
            break;
        case '6':
            menu();
```

```
            break;
        case '7':
            return 0;
            break;
        case 's':
            q.ParallelSort();
            break;
        default:
            break;
        }
    }
}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

```
3
Queue item: created
Queue item: created
Queue item: created
Queue item: created
Queue item: created
Queue item: created
Queue item: deleted
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: created
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: created
Rhombus:
angle: 80 side: 9
79.7694
Rhombus:
angle: 58 side: 58
2852.83
Rhombus:
angle: 31 side: 80
3296.24
Sort:0
Queue item: deleted
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: created
Queue item: deleted
Queue item: created
Queue item: deleted
Queue item: created
Queue item: created
Rhombus:
angle: 80 side: 9
79.7694
Rhombus:
angle: 58 side: 58
2852.83
Rhombus:
angle: 31 side: 80
3296.24
ParallelSort:0
Queue item: deleted
Queue item: deleted
Queue item: deleted
Rhombus deleted
```

```
Rhombus deleted
Rhombus deleted
Queue item: deleted
Queue item: deleted
Queue item: deleted
Rhombus deleted
Rhombus deleted
Rhombus deleted
1. Push rhombus in queue
2. Push pentagon in queue
3. Push trapeze in queue
4. Pop element
5. Print queue
6. Menu
7. Exit
2
Enter the side: 1
Correct value
Queue item: created
1
Enter the angle: 20
Enter the side: 2
Correct value
Queue item: created
5
Pentagon:
Side: 1
Rhombus:
angle: 20 side: 2
3
Enter first side: 3
Enter second side: 3
Enter the height: 3
Correct value
Queue item: created
5
Pentagon:
Side: 1
Rhombus:
angle: 20 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
4
Queue item: deleted
Pentagon:
Side: 1
Pentagon deleted
5
Rhombus:
angle: 20 side: 2
Trapeze:
side 1:  3 side 2: 3 height: 3
4
Queue item: deleted
Rhombus:
angle: 20 side: 2
Rhombus deleted
4
Queue item: deleted
Trapeze:
side 1:  3 side 2: 3 height: 3
Trapeze deleted
7
```

ВЫВОД

В ходе лабораторной работы я изучил параллельное программирование, которое позволяет осуществлять сортировку иным способом, который работает быстрее. То есть, все рекурсивные вызовы осуществяются в разных потоках. Для этой лабораторной работы очень пригодились знания полученные в курсе ОС.

Национальный исследовательский институт
«Московский Авиационный Институт»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Кафедра вычислительной математики ипрограммирования

Отчет по лабораторной работе № 9

по курсу «Объектно-ориентированное программирование»

Работу выполнил
студент **2** курса
очного отделения
Цысь Г.В
группы М80-208Б

преподаватель:
Поповкин Александр
Викторович

Оценка:

Подпись преподавателя

_____

Подпись студента

_____

Число
21.10.2018

Москва-2018

Задание

Используя структуры данных, разработанные для лабораторной работы №6 (контейнер первого уровня и классы-фигуры) необходимо разработать:

- Контейнер второго уровня с использованием шаблонов.
- Реализовать с помощью лямбда-выражений набор команд, совершающих операции над контенйром 1-го уровня:
  - Генерация фигур со случайным значением параметров;
  - Печать контейнера на экран;
  - Удаление элементов со значением площади меньше определенного числа;
- В контенер второго уровня поместить цепочку команд.
- Реализовать цикл, который проходит по всем командам в контенере второго уровня и выполняет их, применяя к контейнеру первого уровня.

Для создания потоков использовать механизмы:

- future
- packaged_task/async

Для обеспечения потоко-безопасности структур данных использовать:

- mutex
- lock_guard

```
int main(void)
{
   TList<Figure> list;
   typedef std::function<void(void)> Command;
   TNList<std::shared_ptr<Command>> nlist;
   std::mutex mtx;

   Command cmdInsert = [&]() {
      std::lock_guard<std::mutex> guard(mtx);

      uint32_t seed = std::chrono::system_clock::now().time_since_epoch().count();

      std::default_random_engine generator(seed);
      std::uniform_int_distribution<int> distFigureType(1, 3);
      std::uniform_int_distribution<int> distFigureParam(1, 10);
      for (int i = 0; i < 5; ++ i) {
         std::cout << "Command: Insert" << std::endl;

         switch(distFigureType(generator)) {
            case 1: {
               std::cout << "Inserted Triangle" << std::endl;

               int side_a = distFigureParam(generator);
               int side_b = distFigureParam(generator);
               int side_c = distFigureParam(generator);

               std::shared_ptr<Figure> ptr = std::make_shared<Triangle>(Triangle(side_a, side_b, side_c));
               list.PushFirst(ptr);
               break;
            }

            case 2: {
               std::cout << "Inserted Octagon" << std::endl;

               int side = distFigureParam(generator);
               std::shared_ptr<Figure> ptr = std::make_shared<Octagon>(Octagon(side));
               list.PushFirst(ptr);

               break;
            }

            case 3: {
```

```
                std::cout << "Inserted Foursquare" << std::endl;
                int side = distFigureParam(generator);
                std::shared_ptr<Figure> ptr = std::make_shared<Foursquare>(Foursquare(side));

                list.PushFirst(ptr);

                break;
            }
        }
    }
};


Command cmdRemove = [&]() {
    std::lock_guard<std::mutex> guard(mtx);

    std::cout << "Command: Remove" << std::endl;

    if (list.IsEmpty()) {
        std::cout << "List is empty" << std::endl;
    } else {
        uint32_t seed = std::chrono::system_clock::now().time_since_epoch().count();

        std::default_random_engine generator(seed);
        std::uniform_int_distribution<int> distSquare(1, 150);
        double sqr = distSquare(generator);
        std::cout << "Lesser than " << sqr << std::endl;

        for (int32_t i = 0; i < 5; ++i) {
            auto iter = list.begin();
            for (int32_t k = 0; k < list.GetLength(); ++k) {
                if (iter->Square() < sqr) {
                    list.Pop(k);
                    break;
                }
                ++iter;
            }
        }
    }
};

Command cmdPrint = [&]() {
    std::lock_guard<std::mutex> guard(mtx);

    std::cout << "Command: Print" << std::endl;
    if(!list.IsEmpty()) {
        std::cout << list << std::endl;
    } else {
        std::cout << "List is empty." << std::endl;
    }
};
nlist.Push(std::shared_ptr<Command>(&cmdInsert, [](Command*){}));
nlist.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));
nlist.Push(std::shared_ptr<Command>(&cmdRemove, [](Command*){}));
nlist.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));


while (!nlist.IsEmpty()) {
    std::shared_ptr<Command> cmd = nlist.Top();
    std::future<void> ft = std::async(*cmd);
    ft.get();
    nlist.Pop();
}

return 0;
}
```

Вывод

Данная лабораторная работа была отличным завершением всех лабораторных работ. В ней я познакомился с новыми для себя лямбда функциями, которые, я думаю, пригодятся для дальнейшего изучения программирования.