Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и информатики

Отчёт по лабораторным работам
по курсу
«Объектно-ориентированное программирование»

Студент: Савина А.А.
Преподаватель: Поповкин А.В
Группа: М8О-208Б-17
Вариант: 23

Дата:
Оценка:
Подпись:

Москва
2018г.

Оглавление

Лабораторная работа №1

Цель работы:

- Программирование классов на языке C++;
- Управление памятью в языке C++;
- Изучение базовых понятий ООП;
- Знакомство с классами в C++;
- Знакомство с перегрузкой операторов;
- Знакомство с дружественными функциями;
- Знакомство с операциями ввода-вывода из стандартных библиотек.

Задание:

Необходимо спроектировать и запрограммировать на языке C++ классы фигур: 6-угольник, 8-угольник, треугольник.

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 иметь 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:

Для данной лабораторной работы особенно понадобились следующие понятия:

Виртуальная функция - функция, для которой связывание вызова функции с её определением происходит во время выполнения программы.

Для обычной функции это происходит во время компиляции.

Виртуальная функция вызывается только через указатель/ссылку на базовый класс.

Виртуальные функции обеспечивают динамическую типизацию.

Чистая виртуальная функция - функция, член класса, тело которой не определено. Она нужна, чтобы отложить реализацию функции на более поздний срок.

Класс, имеющий хотя бы одну чистую виртуальную функцию называется абстрактным классом.

Override показывает то, что тут виртуальная функция, которая overrides виртуальную функцию из базового класса.

При выполнении работы были созданы следующие файлы:

Figure.h	Базовый класс фигуры
Hexagon/Octagon/Triangle.h	Header файлы для классов фигур
Hexagon/Octagon/Triangle.cpp	Имплементация public funcitons фигур
main.cpp	Использование классов-фигур
Makefile	makefile

Листинг кода:

Figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include <iostream>
class Figure {
public:
	virtual double Square() = 0;
	virtual void Print() = 0;
	virtual ~Figure() {};
};
#endif // FIGURE_H
```

Hexagon.h

```
#ifndef HEXAGON_H
#define HEXAGON_H

#include <cstdlib>
#include <iostream>
#include "Figure.h"

class Hexagon : public Figure {
public:
	Hexagon();
	Hexagon(std::istream &is);
	Hexagon(size_t i);
	double Square() override;
	void Print() override;
	virtual ~Hexagon();
private:
	size_t side_a;
};

#endif //HEXAGON_H
```

Hexagon.cpp

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

Hexagon::Hexagon() : Hexagon(0) {
}

Hexagon::Hexagon(size_t i) : side_a(i) {
	std::cout << "Hexagon created: " << side_a << std::endl;
}

Hexagon::Hexagon(std::istream &is) {
	is >> side_a;
}

double Hexagon::Square() {
```

```
	return 3*sqrt(3)*(side_a)*(side_a)/2;
}

void Hexagon::Print() {
	std::cout << "a=" << side_a << std::endl;
}

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

Octagon.h

```
#ifndef OCTAGON_H
#define OCTAGON_H

#include <cstdlib>
#include <iostream>
#include "Figure.h"

class Octagon : public Figure {
public:
	Octagon();
	Octagon(std::istream &is);
	Octagon(size_t i);

	double Square() override;
	void Print() override;

	virtual ~Octagon();

private:
	size_t side_a;
};
#endif //OCTAGON_H
```

Octagon.cpp

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

Octagon::Octagon() : Octagon(0) {
}

Octagon::Octagon(size_t i) : side_a(i) {
		std::cout << "Octagon created: " << side_a << std::endl;
}

Octagon::Octagon(std::istream &is) {
	is >> side_a;
}

double Octagon::Square() {
	return 2*(side_a)*(side_a)*(1+sqrt(2));
}
```

```
void Octagon::Print() {
	std::cout << "a=" << side_a << std::endl;
}

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

Triangle.h

```
#ifndef TRIANGLE_H
#define TRIANGLE_H

#include <cstdlib>
#include <iostream>
#include "Figure.h"

class Triangle : public Figure {
public :
	Triangle();
	Triangle(std::istream &is);
	Triangle(size_t i, size_t j, size_t k);

	double Square() override;
	void Print() override;

	virtual ~Triangle();

private:
	size_t side_a;
	size_t side_b;
	size_t side_c;
};

#endif  // TRIANGLE_H
```

Triangle.cpp

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

Triangle::Triangle() : Triangle(0, 0, 0) {
}

Triangle::Triangle(size_t i, size_t j, size_t k) : side_a(i), side_b(j), side_c(k) {
	std::cout << "Triangle created: " << side_a << ", " << side_b << ", " << side_c << std::endl;
}

Triangle::Triangle(std::istream &is) {
	is >> side_a;
	is >> side_b;
	is >> side_c;
}


double Triangle::Square() {
```

```
	double p = double(side_a + side_b + side_c) / 2.0;
	return sqrt(p * (p - double(side_a))*(p - double(side_b))*(p - double(side_c)));
}

void Triangle::Print() {
	std::cout << "a=" << side_a << ", b=" << side_b << ", c=" << side_c << std::endl;
}

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

main.cpp

```
#include <cstdlib>
#include "Hexagon.h"
#include "Octagon.h"
#include "Triangle.h"
using namespace std;

void help() {
	cout << "Choose:" << endl;
	cout << "1 - Create regular hexagon;" << endl;
	cout << "2 - Create regular octagon;" << endl;
	cout << "3 - Create triangle;" << endl;
	cout << "0 - Exit." << endl;
	cout << "-------------------------" << endl;
}

int main() {
	int c;
	Figure *ptr = nullptr;
	help();
	while (cin >> c) {
		if (c == 1) {
			cout << "Side of regular hexagon:" << endl;
			ptr = new Hexagon(cin);
			ptr->Print();
			cout << "Area of your hexagon = " << ptr->Square() << endl;
			delete ptr;
		}
		else if (c == 2) {
			cout << "Side of the relugar octagon:" << endl;
			ptr = new Octagon(cin);
			ptr->Print();
			cout << "Area of your octagon = " << ptr->Square() << endl;
			delete ptr;
		}
		else if (c == 3) {
			cout << "Sides of your triangle:" << endl;
			ptr = new Triangle(cin);
			ptr->Print();
			cout << "Area of your amazing triangle = " << ptr->Square() << endl;
			delete ptr;
		}

		else
```

```
                        break;
        }
        return 0;
}
```

Makefile

```
CC = g++
FLAGS = -std=c++11 -Wall -Werror -Wno-sign-compare -Wno-unused-result -g
FILES = *.cpp
PROG = no

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

clean:
        rm -f *.o $(PROG)
```

Результаты:

```
user@lubuntu:~/INF/ooplabs1819/lab1/lab1$ ./no
Choose:
1 - Create regular hexagon;
2 - Create regular octagon;
3 - Create triangle;
0 - Exit.
--------------------------
1
Side of regular hexagon:
3
a=3
Area of your hexagon = 23.3827
Hexagon deleted
2
Side of the relugar octagon:
5
a=5
Area of your octagon = 120.711
Octagon deleted
3
Sides of your triangle:
2 4 5
a=2, b=4, c=5
Area of your amazing triangle = 3.79967
Triangle deleted
0
user@lubuntu:~/INF/ooplabs1819/lab1/lab1$
```

Выводы:

Эта работа позволила мне изучить понятия, необходимые для программирования классов на C++, и познакомила с базовыми понятиями ООП.

Лабораторная работа №2

Цель работы:

- Закрепление навыков работы с классами;
- Создание простых динамических структур данных;
- Работа с объектами, передаваемыми «по значению».

Задание:

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня (N-Tree), содержащий одну фигуру(6-угольник).

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 содержать 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 контейнера;
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Программа должна позволять:

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

Описание:

1) Value containers - композиции, в которых хранятся копии объектов, которые они хранят (следовательно, они несут ответственность за создание и уничтожение этих копий);

2) Reference containers - агрегаты, в которых хранятся указатели ссылок на другие объекты (следовательно, они НЕ несут ответственности за создание и уничтожение этих объектов).

Указатель содержит адрес того, на что он указывает.

Создание reference означает создание нового способа вызова местоположения в памяти, где находится объект.

friend класса - это нечто отдельное от класса. Но у него есть доступ к материалам внутри класса.

this ключевое слово, которое идентифицирует особый тип указателя

this хранит адрес текущего объекта, с которым в данный момент происходит работа.

При выполнении работы были созданы следующие файлы:

Figure.h	Базовый класс фигуры
Hexagon.h	Header файлы для классов фигуры
Hexagon.cpp	Имплементация funcitons фигуры
TNTreeItem.h	Header файл для узлов дерева. Нужен, чтобы связать узлы и фигуры, которые туда будет класть
TNTreeItem.cpp	Имплементация funcitons узла
TNTree.h	Header файл для самого дерева
TNTree.cpp	Имплементация funcitons дерева
main.cpp	Использование контейнера
Makefile	makefile

В листинге кода приведены только те файлы, в которых произошло изменение или новые файлы.

Листинг кода:

TNTreeItem.h

```
#ifndef TNTREEITEM_H
#define TNTREEITEM_H

#include <string>
#include "Hexagon.h"

class TNTreeItem {
public:
   TNTreeItem(); //Default constructor
   TNTreeItem(std::string id, const Hexagon& hexagon); //Overload constructor
   friend std::ostream& operator<<(std::ostream& os, const TNTreeItem& obj);

         //Mutator functions
   void SetSon(TNTreeItem* son); //sets pointer to son
   void SetBrother(TNTreeItem* brother); //sets pointer to bro
   //Accessor functions
         TNTreeItem* GetSon(); //returns pointer to son
   TNTreeItem* GetBrother(); //returns pointer to bro
   Hexagon GetHexagon() const; //returns hexagon
   std::string getId(); //returns id

   virtual ~TNTreeItem();

private:
         //Member variables
   Hexagon hexagon;
   std::string id;
   TNTreeItem *son; //pointers
   TNTreeItem *brother;
};

#endif  /* TNTREEITEM_H */
```

TNTreeItem.cpp

```
#include <iostream>
#include <string>

#include "TNTreeItem.h"
#include "Hexagon.h"

//let's create an item
TNTreeItem::TNTreeItem() {
   this->brother = nullptr;
   this->son = nullptr;
   this->id = "";
}
//let's put our hexagon over here
TNTreeItem::TNTreeItem(std::string id, const Hexagon& hexagon) {
   this->hexagon = hexagon;
   this->brother = nullptr;
   this->son = nullptr;
   this->id = id;
}
```

```
//let's set our son
void TNTreeItem::SetSon(TNTreeItem* son) {
   this->son = son;
}
//let's set pur bro
void TNTreeItem::SetBrother(TNTreeItem* brother) {
   this->brother = brother;
}

Hexagon TNTreeItem::GetHexagon() const {
   return this->hexagon; //hexagonio
}

TNTreeItem* TNTreeItem::GetSon() {
   return this->son; //pointer to son return
}

TNTreeItem* TNTreeItem::GetBrother() {
   return this->brother; //pointer to brobro
}

std::string TNTreeItem::getId() {
   return this->id;
}

std::ostream& operator<<(std::ostream& os, const TNTreeItem& obj) {
   os << obj.hexagon << std::endl;
   return os;
}
//let's deletO our item of a treetO
TNTreeItem::~TNTreeItem() {
   delete brother;
   delete son;
}
```

TNTree.h

```
#ifndef TNTREE_H
#define TNTREE_H

#include <iostream>
#include <string>

#include "Hexagon.h"
#include "TNTreeItem.h"

class TNTree {
public:
   TNTree(); //Default constructor. Our tree is alive
   TNTree(const TNTree& orig); //Overload constructor !!treeobject is the parameter
         //Mutator functions
   void Insert(TNTree* tree, std::string parent_id, std::string id, Hexagon &hexagon); //inserto hexagoniO
   void Remove(std::string id); //remove via id
   void node_remove(TNTreeItem* TNTree, std::string id); //remove an item via id
   void PrintTree(TNTreeItem* treeit, size_t num); //print le tree
   void PrintItem(TNTree* tree, std::string id); //print itemu
   void SetNull(TNTreeItem* t); //set our treeto to nullptr
```

```
        TNTreeItem* getHead(); //pointer to the head of our tree
    bool IsEmpty(); //is it empty?? ooo, mistery

    virtual ~TNTree(); //Destructor

private:
        //Member variables
    TNTreeItem* head;
    TNTreeItem* FindNode(TNTreeItem* node, std::string id);
};
#endif /* TNTREE_H */
```

TNTree.cpp

```
#include <string>
#include "TNTree.h"
#include "TNTreeItem.h"

TNTree::TNTree() { //our tree is alive and well
    head = nullptr;
}

TNTree::TNTree(const TNTree& orig) { //head = head of our trreee
    head = orig.head;
}

void TNTree::Insert(TNTree* tree, std::string parent_id, std::string id, Hexagon &hexagon) {
    if (!tree->head) { //no item in head basically
        tree->head = new TNTreeItem(id, hexagon); //overload constructor in TNTreeItem.h
        return;
    }
    else { //oh my there is some stuff here
        TNTreeItem *parent_node = FindNode(tree->head, parent_id); //so we look
                                                                    //for a parent of where to put it
        if (parent_node) { //if there is an id for a parent
            if (!parent_node->GetSon()) { //if there is no son
                parent_node->SetSon(new TNTreeItem(id, hexagon)); //make a new one
            }
            else { //if there is a son
                TNTreeItem *brother = parent_node->GetSon();
                while (brother->GetBrother()) {
                    brother = brother->GetBrother();
                } //make bro
                brother->SetBrother(new TNTreeItem(id, hexagon));
            }
        }
        else { //if there is no parent_id found
            std::cout << "Error: invaild parent_id" << '\n';
        }
    }
}

TNTreeItem* TNTree::FindNode(TNTreeItem* treeItem, std::string id) { //find a pointer to our item
    if (treeItem->getId() == id) { //if our item has id we our looking for
        return treeItem;
    }
```

```
    if (treeItem->GetSon()) { //if our item has a son
        TNTreeItem* tr = FindNode(treeItem->GetSon(), id); //let's take a look at his son as an item
        if (tr != nullptr) { //success hence tr is what we are looking for
            return tr;
        }
    }
    if (treeItem->GetBrother()) { //same as with son, but going through brothers
        TNTreeItem* tr = FindNode(treeItem->GetBrother(), id);
        if (tr != nullptr) {
            return tr;
        }
    }
    return nullptr;
}

bool TNTree::IsEmpty() { //empty treetio has a head of nullptr
    return head == nullptr;
}

void TNTree::node_remove(TNTreeItem* treeItem, std::string id) { //let's deletio our node
    if (treeItem->GetSon()) { //if there are sons we have to delete them as well
        if (treeItem->GetSon()->getId() == id) { //id of the son is the id we are looking for
            TNTreeItem *tr = treeItem->GetSon();
            treeItem->SetSon(treeItem->GetSon()->GetBrother()); //getting rid of the brothers
            SetNull(tr->GetBrother());// = nullptr;
            delete tr;
            return;
        }
        else {
            delete treeItem;
        }
    }
    if (treeItem->GetBrother()) {
        if (treeItem->GetBrother()->getId() == id) {
            TNTreeItem *tr = treeItem->GetBrother();
            treeItem->SetBrother(treeItem->GetBrother()->GetBrother());
            SetNull(tr->GetBrother());// = nullptr;
            delete tr;
            return;
        }
        else {
            delete treeItem;
        }
    }
}

void TNTree::SetNull(TNTreeItem* t) {//pointer to an item is nullptr
    t = nullptr;
}

TNTreeItem* TNTree::getHead() {//get pointer to a head of a tree
    return this->head;
}

void TNTree::Remove(std::string id) { //remove by id
    if (head->getId() == id) {
```

```
        head = head->GetSon();
    } else {
        TNTree::node_remove(head, id);
    }
}

void TNTree::PrintTree(TNTreeItem* treeItem, size_t num) { //let's print our tree
    if (treeItem) { //if there is a treeo
        for (int i = 0; i < num; ++i) { //beauty
            printf("   ");
        }
        std::cout << treeItem->getId()<< '\n'; //get id, print id
        if (treeItem->GetSon()) { // if there is son - go get the son
            PrintTree(treeItem->GetSon(), num + 1);
        }
        if (treeItem->GetBrother()) { //if there is bro - go get bro
            PrintTree(treeItem->GetBrother(), num);
        }
    }
}

void TNTree::PrintItem(TNTree* tree, std::string id) {//let's print our item
    TNTreeItem* tmp = FindNode(tree->head, id); //getting our pointer
    if(tmp) { //if there is a pointer
        tmp->GetHexagon().Print(); //print our hexagonion
    }
    delete tmp;
}

TNTree::~TNTree() { //delete the tree = delete the head
    delete head;
}
```

main.cpp

```
#include <iostream>
#include <string>

#include "TNTree.h"
#include "Hexagon.h"

int main(int argc, char** argv)
{
    TNTree *tree = new TNTree(); //pointer to our alive tree
    std::string c;

    std::cout << "Type 'h' or 'help' for help" << std::endl;
    while (1) {
        std::cin.clear();
        std::cin.sync();
        std::cout << ">";
        std::cin >> c;

        if (c == "e" || c == "exit") {
                delete tree;
            break;
        }
```

```
        else if (c == "i" || c == "insert") {
        size_t size_a;
        std::string parent_id, id;
        std::cout << "Enter parent id: ";
        std::cin >> parent_id;
        std::cout << "Enter node id: ";
        std::cin >> id;
        std::cout << "Enter Hexagon side:\n";
        if (!(std::cin >> size_a)) {
            std::cout << "Incorrect value" << std::endl;
            continue;
        }
        Hexagon* n = new Hexagon(size_a); //we made our hexagon
        tree->Insert(tree, parent_id, id, *n); //let's insert hexagonnn
        delete n; //delete the pointer
        }
        else if (c == "r" || c == "remove") {
          std::string id;
          std::cout << "Enter item id: ";
          if (!(std::cin >> id)) {
            std::cout << "Incorrect value" << std::endl;
            continue;
          }
          tree->Remove(id);
        }
        else if (c == "d" || c == "destroy") {
         delete tree; //delete tree and make place for next
         tree = new TNTree();
         std::cout << "The tree was deleted." << std::endl;
        }
        else if (c == "p" || c == "print") { //print the entire treeo
            if (!tree->IsEmpty()) {
              tree->PrintTree(tree->getHead(), 0);
            }
            else {
              std::cout << "The tree is empty" << std::endl;
            }
        }
        else if (c == "pi" || c == "printi") { //printing an item
          std::string id;
          std::cout << "Enter item id: ";
          std::cin >> id;
          tree->PrintItem(tree, id);
        }
        else if (c == "help" || c  == "h") {
          std::cout << "'h'  or 'help'       - help;" << std::endl;
          std::cout << "'r'  or 'remove'     - remove hexagon;" << std::endl;
          std::cout << "'d'  or 'destroy'    - delete the tree;" << std::endl;
          std::cout << "'p'  or 'print'      - print the tree;" << std::endl;
          std::cout << "'i'  or 'insert'     - insert hexagon into your tree;" << std::endl;
          std::cout << "'pi' or 'printi'     - print hexagon;" << std::endl;
          std::cout << "'e'  or 'exit'       - exit." << std::endl;
        }
    }
    return 0;
}
```

Результаты:

```
user@lubuntu:~/INF/ooplabs1819/lab2/lab2$ ./no
Type 'h' or 'help' for help
>h
'h'  or 'help'         - help;
'r'  or 'remove'     - remove hexagon;
'd'  or 'destroy'    - delete the tree;
'p'  or 'print'        - print the tree;
'i'  or 'insert'        - insert hexagon into your tree;
'pi' or 'printi'       - print hexagon;
'e'  or 'exit'          - exit.
>i
Enter parent id: 1
Enter node id: 1
Enter Hexagon side:
5
>i
Enter parent id: 1
Enter node id: 5
Enter Hexagon side:
34
>i
Enter parent id: 5
Enter node id: 3
Enter Hexagon side:
4
>i
Enter parent id: 1
Enter node id: 2
Enter Hexagon side:
5
>p
1
  5
    3
  2
>pi
Enter item id: 5
a=34
>r
Enter item id: 2
>p
1
  5
    3
>e
user@lubuntu:~/INF/ooplabs1819/lab2/lab2$ ./no
```

Выводы:

Я сделала N-Tree!! Ужас! Но оно работает. Вот. Вывод)

Лабораторная работа №3

Цель работы:

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

Задание:

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня (N-Tree), содержащий все три фигуры (6-угольник, 8-угольник, треугольник).

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).

Программа должна позволять:

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

Описание:

Зачем нам такие указатели нужны? Утечка памяти, обращение к неинициализированным отраслям памяти, удаление уже удаленных объектов. Это плохо.

Что такое smart objects? Смарт-объекты хранят указатели на динамически аллоцированные участки памяти произвольного типа + автоматически очищают память по выходу из области видимости.

Умный указатель с подсчётом ссылок, т.е. где-то существует некая переменная, которая хранит количество указателей. Если эта переменная становится = 0, то объект уничтожается. Счётчик увеличивается при каждом вызове оператора, либо при наличии оператора присваивания + есть оператор приведения к bool, что даёт привычный синтаксис указателей, не заботясь об освобождении памяти.

Shared pointer сохраняет общее владение объектом через указатель. Несколько объектов shared_ptr могут владеть одним и тем же объектом.Shared_ptr может иметь объект в собственности, сохраняя указатель на объекты-члены, в то же время владея объектом, которому они принадлежат. Хранимый указатель доступен через get (), операторы dereference и сравнения.

Собственность - ответственность за освобождение ресурса.

Smart pointer также может быть пустым.

Реализация:

Shared_ptr содержит 2 указателя:

1) сохраненный указатель get ();
2) указатель на ... контрольный блок.

Блок управления - динамически размещаемые объекты, в которых хранятся:

1) указатель на управляемый объект или сам управляемый объект;
2) удалитель;
3) распределитель;
4) номер shared_ptr, которому принадлежит управляемый объект;
5) количество уязвимых мест, ссылающихся на управляемый объект.

У нас есть предметы из дерева.

Элемент дерева имеет 4 переменных-члена: 1)id и 3 shared_ptrs.

2) figure of Figure, 3) сын TNTReeItem, 4) брат TNTreeItem.

Мы создаём epmty shared_ptr для фига типа Figure. А затем делаем из него figure определенного типа (треугольник, шестиугольник, восьмиугольник).

При выполнении работы были созданы следующие файлы:

Figure.h	Базовый класс фигуры
Hexagon/Triangle/Octagon.h	Header файлы для классов фигур
Hexagon/Triangle/Octagon.cpp	Имплементация funcitons фигур
TNTreeItem.h	Header файл для узлов дерева. Нужен, чтобы связать узлы и фигуры, которые туда будет класть
TNTreeItem.cpp	Имплементация funcitons узла
TNTree.h	Header файл для самого дерева
TNTree.cpp	Имплементация funcitons дерева
main.cpp	Использование контейнера
Makefile	makefile

В листинге кода приведены только те файлы, в которых произошло изменение или новые файлы.

Листинг кода:

TNTreeItem.h

```
#ifndef TNTREEITEM_H
#define TNTREEITEM_H

#include <string>
#include <memory>
#include "Figure.h"
#include "Hexagon.h"
#include "Triangle.h"
#include "Octagon.h"

class TNTreeItem {
public:
   TNTreeItem(); //Default constructor
   TNTreeItem(std::string id, const std::shared_ptr<Figure> &figure); //Overload constructor
   friend std::ostream& operator<<(std::ostream& os, const TNTreeItem& obj);
        //Mutator functions
   void SetSon(std::shared_ptr<TNTreeItem> son);
   void SetBrother(std::shared_ptr<TNTreeItem> brother); //sets pointer to bro
   //Accessor functions
        std::shared_ptr<TNTreeItem> GetSon();
   std::shared_ptr<TNTreeItem> GetBrother(); //returns pointer to bro
   std::shared_ptr<Figure> GetFigure() const; //returns figure
   std::string getId(); //returns id
   virtual ~TNTreeItem();
private:
        //Member variables
   std::string id;
   std::shared_ptr<Figure> figure;
        std::shared_ptr<TNTreeItem> son;
        std::shared_ptr<TNTreeItem> brother;
};
```

```
#endif  /* TNTREEITEM_H */
```

TNTreeItem.cpp

```
#include <iostream>
#include <string>

#include "TNTreeItem.h"
#include "Hexagon.h"
//let's create an item
TNTreeItem::TNTreeItem() {
   this->brother = nullptr;
   this->son = nullptr;
   this->id = "";
}
//let's put our hexagon over here
TNTreeItem::TNTreeItem(std::string id, const std::shared_ptr<Figure> &figure) { //only
                                                    // borrowing the resource smth* ish
   this->figure = figure;
   this->brother = nullptr;
   this->son = nullptr;
   this->id = id;
}
//let's set our son
void TNTreeItem::SetSon(std::shared_ptr<TNTreeItem> son) {
   this->son = son;
}
//increase reference counter, later - decrease
void TNTreeItem::SetBrother(std::shared_ptr<TNTreeItem> brother) {
   this->brother = brother;
}

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

std::shared_ptr<TNTreeItem> TNTreeItem::GetSon() {
   return this->son; //pointer to son return
}

std::shared_ptr<TNTreeItem> TNTreeItem::GetBrother() {
   return this->brother; //pointer to brobro
}

std::string TNTreeItem::getId() {
   return this->id;
}

std::ostream& operator<<(std::ostream& os, const TNTreeItem& obj) {
   os << obj.figure << std::endl;
   return os;
}
//let's deletO our item of a treetO
TNTreeItem::~TNTreeItem() {
  // delete brother;
  // delete son;
}
```

TNTree.h

```
#ifndef TNTREE_H
#define TNTREE_H

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

#include "TNTreeItem.h"
#include "Triangle.h"
#include "Hexagon.h"
#include "Octagon.h"

class TNTree {
public:
   TNTree(); //Default constructor. Our tree is alive
   TNTree(const TNTree& orig); //Overload constructor !!treeobject is the parameter

         friend std::ostream &operator<<(std::ostream os, const TNTree &tree);
         //Mutator functions
   void Insert(std::string parent_id, std::string id, std::shared_ptr<Figure> &figure); //inserto figureiO
   void Remove(std::string id); //remove via id
   //void node_remove(TNTreeItem* TNTree, std::string id); //remove an item via id
   void PrintTree(); //print le tree
   void PrintItem(std::string id); //print itemu
   void SetNull(std::shared_ptr<TNTreeItem> treeItem); //set our itit to nullptrrrtr
   void Des(); //ass

         std::shared_ptr<TNTreeItem> getHead(); //Accessor funciton
         bool IsEmpty(); //is it empty?? ooo, mistery

   virtual ~TNTree(); //Destructor

private:
         //Member variables
   std::shared_ptr<TNTreeItem> head;
   std::shared_ptr<TNTreeItem> FindNode(std::shared_ptr<TNTreeItem> node, std::string id);
         void print_nodes(std::shared_ptr<TNTreeItem> treeItem, size_t num);
         void node_remove(std::shared_ptr<TNTreeItem> treeItem, std::string id);
};
#endif /* TNTREE_H */
```

TNTree.cpp

```
#include <string>
#include "TNTree.h"
#include "TNTreeItem.h"

TNTree::TNTree() { //our tree is alive and well
   head = nullptr;
}

TNTree::TNTree(const TNTree& orig) { //head = head of our trreee
   head = orig.head;
}
```

```
//done
void TNTree::Insert(std::string parent_id, std::string id, std::shared_ptr<Figure> &figure) {
    if (!this->head) { //no item in head basically
        this->head = std::make_shared<TNTreeItem>(id, figure); //in TNTreeItem.h
        return;
    }
    else { //oh my there is some stuff here
        std::shared_ptr<TNTreeItem> parent_node = FindNode(this->head, parent_id); //so
                                                                    //we look for a parent of where to put it
        if (parent_node) { //if there is an id for a parent
            if (!parent_node->GetSon()) { //if there is no son
                parent_node->SetSon(std::make_shared<TNTreeItem>(id, figure)); //make a new one
            }
            else { //if there is a son
                std::shared_ptr<TNTreeItem> brother = parent_node->GetSon();
                while (brother->GetBrother()) {
                    brother = brother->GetBrother();
                } //make bro
                brother->SetBrother(std::make_shared<TNTreeItem>(id, figure));
            }
        }
        else { //if there is no parent_id found
            std::cout << "Error: invalid parent_id" << '\n';
        }
    }
}
//done
std::shared_ptr<TNTreeItem> TNTree::FindNode(std::shared_ptr<TNTreeItem> treeItem, std::string id)
{ //find a pointer to our item
    if (treeItem->getId() == id) { //if our item has id we our looking for
        return treeItem;
    }
    if (treeItem->GetSon()) { //if our item has a son
        std::shared_ptr<TNTreeItem>tr = FindNode(treeItem->GetSon(), id); //let's take a look
                                                                    // at his son as an item
        if (tr != nullptr) { //success hence tr is what we are looking for
            return tr;
        }
    }
    if (treeItem->GetBrother()) { //same as with son, but going through brothers
        std::shared_ptr<TNTreeItem> tr = FindNode(treeItem->GetBrother(), id);
        if (tr != nullptr) {
            return tr;
        }
    }
    return nullptr;
}

bool TNTree::IsEmpty() { //empty treetio has a head of nullptr
    return head == nullptr;
}

void TNTree::Des() { //our tree is back to being a baby
        head = nullptr;
}
```

```
void TNTree::SetNull(std::shared_ptr<TNTreeItem> t) {//pointer to an item is nullptr
   t = nullptr;
}

//doneso
void TNTree::node_remove(std::shared_ptr<TNTreeItem> treeItem, std::string id) { //let's deletio our node
   if (treeItem->GetSon()) { //if there are sons we have to delete them as well
      if (treeItem->GetSon()->getId() == id) { //id of the son is the id we are looking for
         std::shared_ptr<TNTreeItem> tr = treeItem->GetSon();
         treeItem->SetSon(treeItem->GetSon()->GetBrother()); //getting rid of the brothers
         SetNull(tr->GetBrother());// = nullptr;
                        //delete tr ! nope, cause shared_ptr
         return;
      }
      else {
                        node_remove(treeItem->GetSon(), id);
                }

   }
   if (treeItem->GetBrother()) {
      if (treeItem->GetBrother()->getId() == id) {
         std::shared_ptr<TNTreeItem> tr = treeItem->GetBrother();
         treeItem->SetBrother(treeItem->GetBrother()->GetBrother());
         SetNull(tr->GetBrother());// = nullptr;
         //delete tr; hohoho you know what i mean nudgenudge winkwink
                        return;
      }
      else {
                        node_remove(treeItem->GetBrother(), id);
                }
   }
}

std::shared_ptr<TNTreeItem> TNTree::getHead() {//get pointer to a head of a tree
   return this->head;
}

void TNTree::Remove(std::string id) { //remove by id
   if (head->getId() == id){
                Des();
        }
   else {
                node_remove(head, id);
        }
}
//okok we are printing it through nodes this time
void TNTree::PrintTree() {
   if (this->head != nullptr) {
                print_nodes(this->head, 0);
        }
}
//done
void TNTree::print_nodes(std::shared_ptr<TNTreeItem> treeItem, size_t num) {
        if (treeItem) {
```

```
        for (int i = 0; i < num; ++i) {
            printf("   ");
        }
        std::cout << treeItem->getId()<< '\n';
        if (treeItem->GetSon()) {
            print_nodes(treeItem->GetSon(), num + 1);
        }
        if (treeItem->GetBrother()) {
            print_nodes(treeItem->GetBrother(), num);
        }
    }
}
void TNTree::PrintItem(std::string id) {//let's print our item
    std::shared_ptr<TNTreeItem> tmp = FindNode(this->head, id); //getting our pointer
    if(tmp) { //if there is a pointer
        tmp->GetFigure()->Print(); //print our fig
    }
    //delete tmp; ohh boii shared_ptr
}

TNTree::~TNTree() { //delete the tree = delete the head
    //delete head; OH GOD
}
```

main.cpp

```
#include <iostream>
#include <string>

#include "TNTree.h"

int main(int argc, char** argv)
{
    TNTree tree; //make a tree object
    std::string c;

    std::cout << "Type 'help' for help" << std::endl;
    while (1) {
        std::cin.clear();
        std::cin.sync();
        std::cout << ">";
        std::cin >> c;

        if (c == "e" || c == "exit") {
        //        delete tree;
            break;
        }
                    else if (c == "t" || c == "triangle") {
            std::shared_ptr<Figure> fig;
                                std::string parent_id, id;

            std::cout << "Enter parent id: ";
            std::cin >> parent_id;
            std::cout << "Enter node id: ";
            std::cin >> id;

                                fig = std::make_shared<Triangle> (std::cin);
```

```
    tree.Insert(parent_id, id, fig);
}

            else if (c == "h" || c == "hexagon") {
                    std::shared_ptr<Figure> fig;
                    std::string parent_id, id;

                    std::cout << "Enter parent id: ";
                    std::cin >> parent_id;
                    std::cout << "Enter node id: ";
                    std::cin >> id;

                    fig = std::make_shared<Hexagon> (std::cin);
                    tree.Insert(parent_id, id, fig);
            }
            else if (c == "o" || c == "octagon") {
                    std::shared_ptr<Figure> fig;
                    std::string parent_id, id;

                    std::cout << "Enter parent id: ";
                    std::cin >> parent_id;
                    std::cout << "Enter node id: ";
                    std::cin >> id;

                    fig = std::make_shared<Octagon> (std::cin);
                    tree.Insert(parent_id, id, fig);
            }
            else if (c == "r" || c == "remove") {
   std::string id;
   std::cout << "Enter item id: ";
   if (!(std::cin >> id)) {
      std::cout << "Incorrect value" << std::endl;
      continue;
   }
   tree.Remove(id);
}
    else if (c == "d" || c == "destroy") {
  if (!tree.IsEmpty()) {
      tree.Des();
   }
   else {
      std::cout << "The tree is already empty" << std::endl;
   }
}
    else if (c == "p" || c == "print") { //print the entire treeo
   if (!tree.IsEmpty()) {
      tree.PrintTree();
   }
   else {
      std::cout << "The tree is empty" << std::endl;
   }
}
    else if (c == "pi" || c == "printi") { //printing an item
   if (!tree.IsEmpty()) {
                                std::string id;
                                std::cout << "Enter item id: ";
```

```
                                    std::cin >> id;

                                    tree.PrintItem(id);
            }
            else {
                                     std::cout << "The tree is empty" << std::endl;
                                }
        }
                    else if (c == "help") {
            std::cout << "'help'            - help;" << std::endl;
                                std::cout << "'t'  or 'triangle' - insert triangle" << std::endl;
                                std::cout << "'h'  or 'hexagon'  - insert hexagon;" << std::endl;
                                std::cout << "'o'  or 'octagon'  - insert octagon;" << std::endl;
            std::cout << "'r'  or 'remove'   - remove figure;" << std::endl;
            std::cout << "'d'  or 'destroy'  - delete the tree;" << std::endl;
            std::cout << "'p'  or 'print'    - print the tree;" << std::endl;
            std::cout << "'pi' or 'printi'   - print figure;" << std::endl;
            std::cout << "'e'  or 'exit'     - exit." << std::endl;
        }
    }

    return 0;
}
```

Результаты:

```
user@lubuntu:~/INF/ooplabs1819/lab3/lab3$ ./no
Type 'help' for help
>t
Enter parent id: 1
Enter node id: 1
Enter sides of your triangle:
2 5 7
>o
Enter parent id: 1
Enter node id: 3
PLEASE, one side of your octagon:
5
>h
Enter parent id: 3
Enter node id: 5
One side of your hexagon, please:
7
>p
1
  3
    5
>pi
Enter item id: 3
Octagon: a = 5
>r
Enter item id: 5
>p
1
  3
>d
>p
The tree is empty
>e
user@lubuntu:~/INF/ooplabs1819/lab3/lab3$
```

Выводы:

Shared pointer, конечно, хорош. Здорово, когда не нужно думать о том, когда и где освобождать a poiner. Но имплементация всего этого. Уф. Намучалась я тут с деревам и этими умненькими указателями.

Лабораторная работа №4

Цель работы:

- Знакомство с шаблонами классов;
- Построение шаблонов динамических структур данных.

Задание:

Необходимо спроектировать и запрограммировать на языке C++ шаблон класса-контейнера первого уровня (N-Tree), содержащий все три фигуры (6-угольник, 8-угольник, треугольник).

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).

Программа должна позволять:

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

Описание:

Есть шаблоны функций и шаблоны классов. Шаблон - это модель программирования, которая позволяет подключать к коду любой тип данных.

Компиляция шаблонов происходит только для необходимых типов данных. «On demand compilation».

«Undefined reference to» template class constructor error..

Понимание проблемы:

Проблема вызвана тем, что main.cpp и cola.cpp сначала компилируются отдельно. В main.cpp компилятор неявно создает экземпляры классов шаблонов, потому что эти конкретные экземпляры используются в main.cpp.

Плохая новость заключается в том, что реализации этих функций-членов не находятся ни в main.cpp, ни в каком-либо заголовочном файле, включенном в main.cpp, и поэтому компилятор не может включать полные версии этих функций в main.o. При компиляции TNTree.cpp компилятор также не будет компилировать эти экземпляры, потому что нет явных или явных экземпляров.

При компиляции TNTree.cpp компилятор не имеет ни малейшего представления, какие экземпляры понадобятся; и мы не можем ожидать, что он скомпилируется для каждого типа, чтобы эта проблема никогда не возникала!

Два решения:

1) Сказать компилятору в конце TNTree.cpp, какие конкретные классы шаблонов потребуются, заставив его скомпилировать определённые экземпляры;

2) Поместить реализацию функций-членов в заголовочный файл, который будет включаться каждый раз, когда любой другой «translation unit» (например, main.cpp) использует класс шаблона.

Функции-члены шаблонного класса сами являются также шаблонами. Из-за этого они должны быть определены с любыми необходимыми параметрами шаблона.

При выполнении работы были созданы следующие файлы:

Figure.h	Базовый класс фигуры
Hexagon/Triangle/Octagon.h	Header файлы для классов фигур
Hexagon/Triangle/Octagon.cpp	Имплементация funcitons фигур
TNTreeItem.h	Header файл для узлов дерева. Нужен, чтобы связать узлы и фигуры, которые туда будет класть
TNTreeItem.cpp	Имплементация funcitons узла
TNTree.h	Header файл для самого дерева
TNTree.cpp	Имплементация funcitons дерева
main.cpp	Использование контейнера
Makefile	makefile

В листинге кода приведены только те файлы, в которых произошло изменение или новые файлы.

Листинг кода:

TNTreeItem.h

```
#ifndef TNTREEITEM_H
#define TNTREEITEM_H

#include <string>
#include <memory>

#include "Figure.h"
#include "Hexagon.h"
#include "Triangle.h"
#include "Octagon.h"

template <class T>
class TNTreeItem {
public:
   TNTreeItem(); //Default constructor
   TNTreeItem(std::string id, const std::shared_ptr<T> &figure); //Overload constructor

   template <class U>
         friend std::ostream& operator<<(std::ostream& os, const TNTreeItem<U>& obj);

         //Mutator functions
   void SetSon(std::shared_ptr<TNTreeItem<T>> son);
   void SetBrother(std::shared_ptr<TNTreeItem<T>> brother); //sets pointer to bro
   //Accessor functions
         std::shared_ptr<TNTreeItem<T>> GetSon();
   std::shared_ptr<TNTreeItem<T>> GetBrother(); //returns pointer to bro
   std::shared_ptr<T> GetFigure() const; //returns figure
   std::string getId(); //returns id

   virtual ~TNTreeItem() {};

private:
         //Member variables
   std::string id;
   std::shared_ptr<T> figure;

         std::shared_ptr<TNTreeItem<T>> son;
         std::shared_ptr<TNTreeItem<T>> brother;
};

#endif  /* TNTREEITEM_H */
```

TNTreeItem.cpp

```
#include <iostream>
#include <string>

#include "TNTreeItem.h"
#include "Figure.h"
```

```
template class TNTreeItem<Figure>;
template std::ostream& operator<<(std::ostream&, const TNTreeItem<Figure> &obj);

//let's create an item
template <class T>
TNTreeItem<T>::TNTreeItem() {
    this->brother = nullptr;
    this->son = nullptr;
    this->id = "";
}
//let's put our figure over here
template <class T>
TNTreeItem<T>::TNTreeItem(std::string id, const std::shared_ptr<T> &figure) { //only borrowing the
resource smth* ish
    this->figure = figure;
    this->brother = nullptr;
    this->son = nullptr;
    this->id = id;
}
//let's set our son
template <class T>
void TNTreeItem<T>::SetSon(std::shared_ptr<TNTreeItem<T>> son) {
    this->son = son;
}
//increase reference counter, later - decrease
template <class T>
void TNTreeItem<T>::SetBrother(std::shared_ptr<TNTreeItem<T>> brother) {
    this->brother = brother;
}

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

template <class T>
std::shared_ptr<TNTreeItem<T>> TNTreeItem<T>::GetSon() {
    return this->son; //pointer to son return
}

template <class T>
std::shared_ptr<TNTreeItem<T>> TNTreeItem<T>::GetBrother() {
    return this->brother; //pointer to brobro
}

template <class T>
std::string TNTreeItem<T>::getId() {
    return this->id;
}

template <class T>
std::ostream& operator<<(std::ostream& os, const TNTreeItem<T>& obj) {
    os << obj.figure << std::endl;
    return os;
}
```

TNTree.h

```
#ifndef TNTREE_H
#define TNTREE_H

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

#include "TNTreeItem.h"
#include "Triangle.h"
#include "Hexagon.h"
#include "Octagon.h"

template <class T>
class TNTree {
public:
   TNTree(); //Default constructor. Our tree is alive
   TNTree(const TNTree<T>& orig); //Overload constructor !!treeobject is the parameter

         //       friend std::ostream &operator<<(std::ostream os, const TNTree &tree);
         //Mutator functions
   void Insert(std::string parent_id, std::string id, std::shared_ptr<T> &figure); //inserto figureiO
   void Remove(std::string id); //remove via id
   //void node_remove(TNTreeItem* TNTree, std::string id); //remove an item via id
   void PrintTree() const; //print le tree
   void PrintItem(std::string id); //print itemu
   void SetNull(std::shared_ptr<TNTreeItem<T>> treeItem); //set our treeto to nullptr
   void Des(); //ass

         std::shared_ptr<TNTreeItem<T>> getHead(); //Accessor funciton
         bool IsEmpty() const; //is it empty?? ooo, mistery

   virtual ~TNTree() {}; //Destructor

private:
         //Member variables
   std::shared_ptr<TNTreeItem<T>> head;
   std::shared_ptr<TNTreeItem<T>> FindNode(std::shared_ptr<TNTreeItem<T>> node, std::string id);
         void print_nodes(std::shared_ptr<TNTreeItem<T>> treeItem, size_t num) const;
         void node_remove(std::shared_ptr<TNTreeItem<T>> treeItem, std::string id);
};
#endif /* TNTREE_H */
```

TNTree.cpp

```
#include <string>
#include <memory>

#include "TNTree.h"
#include "TNTreeItem.h"

template class TNTree<Figure>;

template <class T>
TNTree<T>::TNTree() { //our tree is alive and well
   head = nullptr;
}
```

```
template <class T>
TNTree<T>::TNTree(const TNTree<T>& orig) { //head = head of our trreee
   head = orig.head;
}
//done
template <class T>
void TNTree<T>::Insert(std::string parent_id, std::string id, std::shared_ptr<T> &figure) {
   if (!this->head) { //no item in head basically
      this->head = std::make_shared<TNTreeItem<T>>(id, figure); //in TNTreeItem.h
      return;
   }
   else { //oh my there is some stuff here
      std::shared_ptr<TNTreeItem<T>> parent_node = FindNode(this->head, parent_id); //so we look for a
parent of where to put it
      if (parent_node) { //if there is an id for a parent
         if (!parent_node->GetSon()) { //if there is no son
            parent_node->SetSon(std::make_shared<TNTreeItem<T>>(id, figure)); //make a new one
         }
         else { //if there is a son
            std::shared_ptr<TNTreeItem<T>> brother = parent_node->GetSon();
            while (brother->GetBrother()) {
               brother = brother->GetBrother();
            } //make bro
            brother->SetBrother(std::make_shared<TNTreeItem<T>>(id, figure));
         }
      }
      else { //if there is no parent_id found
         std::cout << "Error: invaild parent_id" << '\n';
      }
   }
}
//done
template <class T>
std::shared_ptr<TNTreeItem<T>>   TNTree<T>::FindNode(std::shared_ptr<TNTreeItem<T>>   treeItem,
std::string id) { //find a pointer to our item
   if (treeItem->getId() == id) { //if our item has id we our looking for
      return treeItem;
   }
   if (treeItem->GetSon()) { //if our item has a son
      std::shared_ptr<TNTreeItem<T>>tr = FindNode(treeItem->GetSon(), id); //let's take a look at his son
as an item
      if (tr != nullptr) { //success hence tr is what we are looking for
         return tr;
      }
   }
   if (treeItem->GetBrother()) { //same as with son, but going through brothers
      std::shared_ptr<TNTreeItem<T>> tr = FindNode(treeItem->GetBrother(), id);
      if (tr != nullptr) {
         return tr;
      }
   }
   return nullptr;
}
template <class T>
void TNTree<T>::SetNull(std::shared_ptr<TNTreeItem<T>> t) {//pointer to an item is nullptr
   t = nullptr;
```

```
}
template <class T>
bool TNTree<T>::IsEmpty() const { //empty treetio has a head of nullptr
   return head == nullptr;
}

template <class T>
void TNTree<T>::Des() {
          head = nullptr;
}
//doneso
template <class T>
void TNTree<T>::node_remove(std::shared_ptr<TNTreeItem<T>> treeItem, std::string id) { //let's deletio
our node
   if (treeItem->GetSon()) { //if there are sons we have to delete them as well
      if (treeItem->GetSon()->getId() == id) { //id of the son is the id we are looking for
         std::shared_ptr<TNTreeItem<T>> tr = treeItem->GetSon();
         treeItem->SetSon(treeItem->GetSon()->GetBrother()); //getting rid of the brothers
         SetNull(tr->GetBrother());// = nullptr;
                              //delete tr ! nope, cause shared_ptr
         return;
      }
      else {
                              TNTree<T>::node_remove(treeItem->GetSon(), id);
      }
   }
   if (treeItem->GetBrother()) {
      if (treeItem->GetBrother()->getId() == id) {
         std::shared_ptr<TNTreeItem<T>> tr = treeItem->GetBrother();
         treeItem->SetBrother(treeItem->GetBrother()->GetBrother());
         SetNull(tr->GetBrother());// = nullptr;
         //delete tr; hohoho you know what i mean nudgenudge winkwink
         return;
      }
      else {
         TNTree<T>::node_remove(treeItem->GetBrother(), id);
      }
   }
}
template <class T>
std::shared_ptr<TNTreeItem<T>> TNTree<T>::getHead() {//get pointer to a head of a tree
   return this->head;
}
template <class T>
void TNTree<T>::Remove(std::string id) { //remove by id
   if (head->getId() == id) {
      TNTree<T>::Des();
   } else {
      TNTree<T>::node_remove(head, id);
  }
}
//okok we are printing it through nodes this time
template <class T>
void TNTree<T>::PrintTree() const{
   if (this->head != nullptr) {
                  print_nodes(this->head, 0);
```

```
            }
}
//done
template <class T>
void TNTree<T>::print_nodes(std::shared_ptr<TNTreeItem<T>> treeItem, size_t num) const {
            if (treeItem) {
        for (int i = 0; i < num; ++i) {
            printf("   ");
        }
        std::cout << treeItem->getId()<< '\n';
        if (treeItem->GetSon()) {
            print_nodes(treeItem->GetSon(), num + 1);
        }
        if (treeItem->GetBrother()) {
            print_nodes(treeItem->GetBrother(), num);
        }
    }
}
template <class T>
void TNTree<T>::PrintItem(std::string id) {//let's print our item
    std::shared_ptr<TNTreeItem<T>> tmp = FindNode(this->head, id); //getting our pointer
    if(tmp) { //if there is a pointer
        tmp->GetFigure()->Print(); //print our fig
    }
    //delete tmp; ohh boii shared_ptr
}
```

main.cpp

```
#include <iostream>
#include <string>
#include "TNTree.h"
#include "TNTreeItem.h"
#include "Figure.h"

int main(int argc, char** argv)
{
    TNTree<Figure> tree; //make an object
    std::string c;

    std::cout << "Type 'help' for help" << std::endl;
    while (1) {
        std::cin.clear();
        std::cin.sync();
        std::cout << ">";
        std::cin >> c;

        if (c == "e" || c == "exit") {
        //        delete tree;
            break;
        }
       else if (c == "t" || c == "triangle") {
            std::shared_ptr<Figure> fig;
                                std::string parent_id, id;

            std::cout << "Enter parent id: ";
            std::cin >> parent_id;
```

```
    std::cout << "Enter node id: ";
    std::cin >> id;

    fig = std::make_shared<Triangle> (std::cin);
    tree.Insert(parent_id, id, fig);
}
            else if (c == "h" || c == "hexagon") {
                    std::shared_ptr<Figure> fig;
                    std::string parent_id, id;

                    std::cout << "Enter parent id: ";
                    std::cin >> parent_id;
                    std::cout << "Enter node id: ";
                    std::cin >> id;

                    fig = std::make_shared<Hexagon> (std::cin);
                    tree.Insert(parent_id, id, fig);
            }
            else if (c == "o" || c == "octagon") {
                    std::shared_ptr<Figure> fig;
                    std::string parent_id, id;

                    std::cout << "Enter parent id: ";
                    std::cin >> parent_id;
                    std::cout << "Enter node id: ";
                    std::cin >> id;

                    fig = std::make_shared<Octagon> (std::cin);
                    tree.Insert(parent_id, id, fig);
            }
            else if (c == "r" || c == "remove") {
   if (!tree.IsEmpty()) {
                            std::string id;
                            std::cout << "Enter item id: ";
                            if (!(std::cin >> id)) {
                                    std::cout << "Incorrect value" << std::endl;
                                    continue;
                            }
                            tree.Remove(id);
                    }
                    else {
                             std::cout << "The tree is empty" << std::endl;
                    }
}
            else if (c == "d" || c == "destroy") {
  if (!tree.IsEmpty()) {
      tree.Des();
   }
   else {
      std::cout << "The tree is already empty" << std::endl;
   }
  //delete tree; //delete tree and make place for next
  //tree = new TNTree();
  //std::cout << "The tree was deleted." << std::endl;
}
            else if (c == "p" || c == "print") { //print the entire treeo
```

```
            if (!tree.IsEmpty()) {
                tree.PrintTree();
            }
            else {
                std::cout << "The tree is empty" << std::endl;
            }
        }
                else if (c == "pi" || c == "printi") { //printing an item
                        if (!tree.IsEmpty()) {
                                std::cout << "Enter item id: ";
                                std::cin >> id;

                                tree.PrintItem(id);
                        }
                        else {
                                std::cout << "The tree is empty" << std::endl;
                        }
        }
                else if (c == "help") {
            std::cout << "'help'           - help;" << std::endl;
                        std::cout << "'t' or 'triangle' - insert triangle" << std::endl;
                        std::cout << "'h' or 'hexagon'  - insert hexagon;" << std::endl;
                        std::cout << "'o' or 'octagon'  - insert octagon;" << std::endl;
            std::cout << "'r'  or 'remove'  - remove figure;" << std::endl;
            std::cout << "'d'  or 'destroy' - delete the tree;" << std::endl;
            std::cout << "'p'  or 'print'   - print the tree;" << std::endl;
            std::cout << "'pi' or 'printi'  - print figure;" << std::endl;
            std::cout << "'e'  or 'exit'    - exit." << std::endl;
        }
    }

    return 0;
}
```

Результаты:

```
user@lubuntu:~/INF/ooplabs1819/lab4/lab4$ ./no
Type 'help' for help
>t
Enter parent id: 1
Enter node id: 1
Enter sides of your triangle:
2 5 7
>o
Enter parent id: 1
Enter node id: 3
PLEASE, one side of your octagon:
5
>h
Enter parent id: 3
Enter node id: 5
One side of your hexagon, please:
7
>p
1
  3
    5
>pi
Enter item id: 3
Octagon: a = 5
>r
Enter item id: 5
>p
1
  3
>d
>p
The tree is empty
>e
user@lubuntu:~/INF/ooplabs1819/lab4/lab4$
```

Выводы:

Уф. Так много ошибок было. Фу. Но сам концепт of templates is nice.

Лабораторная работа №5

Цель работы:

- Закрепление навыков работы с шаблонами классов;
- Построение итераторов для динамических структур данных.

Задание:

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

Итератор должен быть разработан в виде шаблона и должен уметь работать со всеми типами фигур (6-угольник, 8-угольник, треугольник).

Итератор должен позволять использовать структуру данных в операторах типа for.

Программа должна позволять:

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

Описание:

Итератор - это объект, который позволяет перемещаться (итерироваться) по элементам некоторой последовательности.

В отличие от разнообразных последовательностей элементов (массивы, списки, файлы), итераторы имеют одинаковый интерфейс: получение текущего элемента, перемещение к следующему. Это позволяет писать более общие алгоритмы, которые работают с любыми итераторами, поддерживающими этот минимальный набор функций.

При выполнении работы были созданы следующие файлы:

Figure.h	Базовый класс фигуры
Iterator.h	Итератор
Hexagon/Triangle/Octagon.h	Header файлы для классов фигур
Hexagon/Triangle/Octagon.cpp	Имплементация funcitons фигур
TNTreeItem.h	Header файл для узлов дерева. Нужен, чтобы связать узлы и фигуры, которые туда будет класть
TNTreeItem.cpp	Имплементация funcitons узла
TNTree.h	Header файл для самого дерева
TNTree.cpp	Имплементация funcitons дерева
main.cpp	Использование контейнера
Makefile	makefile

В листинге кода приведены только те файлы, в которых произошло изменение или новые файлы.

Листинг кода:

Iterator.h

```
#ifndef TITERATOR_H
#define TITERATOR_H

#include <iostream>
#include <memory>

template <class N, class T>
class TIterator {
public:
	TIterator(std::shared_ptr<N> n) { //it is a pointer to
		iter = n; //an item officially now
	}

	std::shared_ptr<T> operator*() { //oooh what's over here?
		return iter->GetFigure();
	}

	std::shared_ptr<T> operator->() {
		return iter->GetFigure();
	} // same... ..

	void operator++() {
		iter = iter->Next();
	}

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

	bool operator==(const TIterator &i) {// 2 iterators are equal
		return (iter == i.iter);
	}

	bool operator!=(const TIterator &i) {// 2 iterators are NOT equal
		return (iter != i.iter);
	}

private:
	std::shared_ptr<N> iter;// here is our beauty iter-pointer
};

#endif
```

Изменения в TNTreeItem.h

```
#include "TIterator.h" …
public: …
	std::shared_ptr<TNTreeItem<T>> Begin();
	std::shared_ptr<TNTreeItem<T>> Next();
```

Изменения в TNTreeItem.cpp

```
template <class T>
std::shared_ptr<TNTreeItem<T>> TNTreeItem<T>::Begin() {
        if (!this->son) {
                return this->shared_from_this();
        }
        else {
      auto iter = this->son;
      while (iter->son)
         iter = iter->son;
      return iter;
   }
}

template <class T>
std::shared_ptr<TNTreeItem<T>> TNTreeItem<T>::Next() {
        if (this->brother)
                return this->brother;
        else
                return nullptr;
}
```

Изменения в TNTree.h

```
#include "TIterator.h" …
public: …
        TIterator<TNTreeItem<T>, T> begin();
        TIterator<TNTreeItem<T>, T> end();
```

Изменения в TNTree.cpp

```
template <class T>
TIterator <TNTreeItem<T>, T> TNTree<T>::begin() {
        if (head) { //if there is a head
                return TIterator<TNTreeItem<T>, T>(head);
        }
        else {
                return TIterator<TNTreeItem<T>, T>(nullptr);
        }
}

template <class T>
TIterator <TNTreeItem<T>, T> TNTree<T>::end() {
        return TIterator<TNTreeItem<T>, T>(nullptr);
}
```

Результаты:

```
user@lubuntu:~/INF/ooplabs1819/lab4/lab4$ ./no
Type 'help' for help
>t
Enter parent id: 1
Enter node id: 1
Enter sides of your triangle:
2 5 7
>o
Enter parent id: 1
Enter node id: 3
PLEASE, one side of your octagon:
5
>h
Enter parent id: 3
Enter node id: 5
One side of your hexagon, please:
7
>p
1
  3
    5
>pi
Enter item id: 3
Octagon: a = 5
>r
Enter item id: 5
>p
1
  3
>d
>p
The tree is empty
>e
user@lubuntu:~/INF/ooplabs1819/lab4/lab4$
```

Выводы:

Итератор-то этот работает. Но нужен ли он тут. Не знаю. Не использую я его. Вот.

Лабораторная работа №6

Цель работы:

- Закрепление навыков по работе с памятью в C++;
- Создание аллокаторов памяти для динамических структур данных.

Задание:

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

Аллокатор должен выделять большие блоки памяти для хранения фигур и при создании новых фигур-объектов выделять место под объекты в этой памяти.

Алокатор должен хранить списки использованных/свободных блоков. Для хранения списка свободных блоков нужно применять динамическую структуру данных (Очередь).

Для вызова аллокатора должны быть переопределены оператор new и delete у классов-фигур.

Программа должна позволять:

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

Описание:

Цель построения аллокатора – минимизация вызова операции malloc.

Аллокатор выделяет большие блоки памяти для хранения объектов и при создании новых объектов выделяет место под них.

Аллокатор хранит списки использованных и свободных блоков памяти. Список свободных блоков памяти содержится в очереди.

При выполнении работы были созданы следующие файлы:

Figure.h	Базовый класс фигуры
Iterator.h	Итератор
TAllocationBlock.h	Header файл для аллоктора
TAllocationBlock.cpp	Имплементация аллокатора
TQueueItem.h	Header файл для элементов очереди
TQueueItem.cpp	Имплементация
TQueue.h	Header файл для очереди
TQueue.cpp	Имплементация
Hexagon/Triangle/Octagon.h	Header файлы для классов фигур
Hexagon/Triangle/Octagon.cpp	Имплементация funcitons фигур
TNTreeItem.h	Header файл для узлов дерева. Нужен, чтобы связать узлы и фигуры, которые туда будет класть
TNTreeItem.cpp	Имплементация funcitons узла
TNTree.h	Header файл для самого дерева
TNTree.cpp	Имплементация funcitons дерева
main.cpp	Использование контейнера
Makefile	makefile

В листинге кода приведены только те файлы, в которых произошло изменение или новые файлы.

Листинг кода:

TAllocationBlock.h

```
#ifndef TALLOCATIONBLOCK_H
#define TALLOCATIONBLOCK_H

#include <iostream>
#include <cstdlib>
#include "TQueue.h"
typedef unsigned char Byte;

class TAllocationBlock
{
public:
   TAllocationBlock(int32_t size, int32_t count);
   void *Allocate();
   void Deallocate(void *ptr);
   bool Empty();
   int32_t Size();
   virtual ~TAllocationBlock();
private:
   Byte *_used_blocks;
   TQueue<void *>_free_blocks;
};

#endif /* TALLOCATIONBLOCK_H */
```

TAllocationBlock.cpp

```
#include "TAllocationBlock.h"

TAllocationBlock::TAllocationBlock(int32_t size, int32_t count)
{
   _used_blocks = (Byte *)malloc(size * count);

   for(int32_t i = 0; i < count; ++i) {
      void *ptr = (void *)malloc(sizeof(void *));
      ptr = _used_blocks + i * size;
      _free_blocks.Push(ptr);
   }
}

void *TAllocationBlock::Allocate() {
   if(!_free_blocks.IsEmpty()) {
      void *res = nullptr;
      _free_blocks.Pop();
      return res;
   }
   else {
      throw std::bad_alloc();
   }
}

void TAllocationBlock::Deallocate(void *ptr)
{
   _free_blocks.Push(ptr);
}

bool TAllocationBlock::Empty()
{
   return _free_blocks.IsEmpty();
}
```

TQueueItem.h

```
#ifndef TQUEUEITEM_H
#define TQUEUEITEM_H

#include <iostream>
#include <memory>

template <class T>
class TQueueItem {
public:
   TQueueItem(const T &val, TQueueItem<T> *item);
   virtual ~TQueueItem();
   void SetNext(TQueueItem<T> *item);
   TQueueItem<T> &GetNext() const;
private:
   T *value;
   TQueueItem<T> *next;
};
#endif //TQUEUEITEM_H
```

TQueueItem.cpp

```
#include "TQueueItem.h"
template <class T>
TQueueItem<T>::TQueueItem(const T &val, TQueueItem<T> *item) {
   value = new T(val);
   next = item;
}

template <class T>
void TQueueItem<T>::SetNext(TQueueItem<T> *item)
{
   next = item;
}

template <class T>
TQueueItem<T> &TQueueItem<T>::GetNext() const
{
   return *next;
}

template <class T> TQueueItem<T>::~TQueueItem() {
   delete value;
}

typedef unsigned char Byte;
template class
TQueueItem<void *>;
```

TQueue.h

```
#ifndef TQUEUE_H
#define TQUEUE_H

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

template <class T>
class TQueue {
public:
   TQueue();
   virtual ~TQueue();
void Push(const T &item);
   void Pop();
   T &Front();
   bool IsEmpty() const;
   uint32_t GetSize() const;

private:
        TQueueItem<T> *first;
        TQueueItem<T> *last;
   uint32_t size;

};
#endif //TQUEUE_H
```

TQueue.cpp

```
#include "TQueue.h"

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

template <class T>
bool TQueue<T>::IsEmpty() const {
   return size == 0;
}

template <class T>
uint32_t TQueue<T>::GetSize() const
{
   return size;
}

template <class T>
void TQueue<T>::Push(const T &item) {
        TQueueItem<T> *tmp = new TQueueItem<T>(item, first);
   first = tmp;
   size++;
}

template <class T>
void TQueue<T>::Pop() {
   if(first != nullptr) {
      TQueueItem<T>  *oldFirst = &first->GetNext();
      oldFirst->SetNext(nullptr);
      size--;
      if(size == 0) {
         last = nullptr;
      }
   }
}

template <class T> TQueue<T>::~TQueue() {
        while (!IsEmpty()) {
      TQueueItem<T> *temp = &first->GetNext();
      delete temp;

   }
   first = nullptr;
   last = nullptr;

}

typedef unsigned char Byte;

template class
TQueue<void *>;
```

Изменения в TNTreeItem.h

```
#include "TAllocationBlock.h" …
public: …
        void *operator new(size_t size);
        void operator delete(void *ptr);
private: …
        static TAllocationBlock item_allocator;
```

Изменения в TNTreeItem.cpp

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

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

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

template <class T>
void TNTreeItem<T>::operator delete(void *ptr) {
        item_allocator.Deallocate(ptr);
}
```

Результаты:

```
user@lubuntu:~/INF/ooplabs1819/lab4/lab4$ ./no
Type 'help' for help
>t
Enter parent id: 1
Enter node id: 1
Enter sides of your triangle:
2 5 7
>o
Enter parent id: 1
Enter node id: 3
PLEASE, one side of your octagon:
5
>h
Enter parent id: 3
Enter node id: 5
One side of your hexagon, please:
7
>p
1
  3
    5
>pi
Enter item id: 3
Octagon: a = 5
>r
Enter item id: 5
>p
1
  3
>d
>p
The tree is empty
>e
user@lubuntu:~/INF/ooplabs1819/lab4/lab4$
```

Выводы:

Я, конечно, с аллокатором немного помучилась, но всё вроде бы получилось. Мне не очень понравилось.

Лабораторная работа №7

Цель работы:

- Создание сложных динамических структур данных;
- Закрепление принципа OCP.

Задание:

Необходимо реализовать динамическую структуру данных – «Хранилище объектов» и алгоритм работы с ней. «Хранилище объектов» представляет собой N-Tree.

Каждым элементом контейнера, в свою, является очередью.

Элементом второго контейнера является объекты-фигуры (6-угольник, 8-угольник, треугольник).

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер;
- Распечатывать содержимое контейнера (1-го и 2-го уровня);
- Удалять фигуры из контейнера по критериям:
    - По типу (например, все квадраты);
    - По площади (например, все объекты с площадью меньше чем заданная).

Описание:

Ничего особенного. Просто контейнер в контейнере.

При выполнении работы были созданы следующие файлы:

Figure.h	Базовый класс фигуры
Iterator.h	Итератор
TAllocationBlock.h	Header файл для аллоктора
TAllocationBlock.cpp	Имплементация аллокатора
TQueueItemA.h	Header файл для элементов очереди
TQueueItemA.cpp	Имплементация
TQueueA.h	Header файл для очереди
TQueueA.cpp	Имплементация
Hexagon/Triangle/Octagon.h	Header файлы для классов фигур
Hexagon/Triangle/Octagon.cpp	Имплементация funcitons фигур
TTree.h	Header файл для дерева
TTree.hpp	Имплементация funcitons дерева
TQueue.h	Header файл для очереди
TQueue.hpp	Имплементация funcitons очереди
main.cpp	Использование контейнера
Makefile	makefile

В листинге кода приведены только те файлы, в которых произошло изменение или новые файлы.

Листинг кода:

TTree.h

```
#ifndef TTREE_H
#define TTREE_H

#include <iostream>
#include <memory>

#include "Figure.h"

template <class Q, class O> class TTree {
private:
	class Node {
	public:
		Q data;
		std::shared_ptr<Node> son;
		std::shared_ptr<Node> sibling;
		Node();
		Node(const O&);
		int itemsInNode;
	};

	std::shared_ptr<Node> root;

	void RemByType(std::shared_ptr<Node>&, const int&);
```

```
        void Insert(std::shared_ptr<Node>&, const O&);
        void Inorder(const std::shared_ptr<Node>&);
        void InorderP(const std::shared_ptr<Node>&);
        void RemLesser(std::shared_ptr<Node>&, const double&);
public:
        TTree();

        void insert(const O&);
        void inorder();
        void inorderp();
        void removeByType(const int&);
        void removeLesser(const double&);

        bool IsEmpty() const;
        void Des();
};

#include "TTree.hpp"
#endif
```

TTree.hpp

```
#ifdef TQUEUE_H

template class TTree<TQueue<Figure>, std::shared_ptr<Figure>>;

template <class Q, class O> TTree<Q, O>::TTree() {
        root = std::make_shared<Node>(Node());
        root->son = std::make_shared<Node>(Node());
}

template <class Q, class O> TTree<Q, O>::Node::Node() {
        son = sibling = nullptr;
        itemsInNode = 0;
}

template <class Q, class O> TTree<Q, O>::Node::Node(const O& item) {
        data.Push(item);
        itemsInNode = 1;
}

template <class Q, class O>
bool TTree<Q, O>::IsEmpty() const {
   return root == nullptr;
}

template <class Q, class O>
void TTree<Q, O>::Des() {
        root = nullptr;
        root = std::make_shared<Node>(Node());
        root->son = std::make_shared<Node>(Node());
}

template <class Q, class O>
void TTree<Q, O>::insert(const O& item) {
                Insert(root->son, item);
}
```

```
template <class Q, class O>
void TTree<Q, O>::inorder() {
	Inorder(root->son);
}

template <class Q, class O>
void TTree<Q, O>::inorderp() {
	InorderP(root->son);
}

template <class Q, class O>
void TTree<Q, O>::removeByType(const int& type) {
	RemByType(root->son, type);
}

template <class Q, class O>
void TTree<Q, O>::removeLesser(const double& sqr) {
	RemLesser(root->son, sqr);
}

template <class Q, class O>
void TTree<Q, O>::Insert(std::shared_ptr<Node>& node, const O& item) {
	if (node->itemsInNode < 5) {
		node->data.Push(item);
		node->itemsInNode++;
	}
	else {
		auto sibl = node;

		for (int i = 0; i < 3; i++) {
			if (!sibl->sibling) {
			sibl->sibling = std::make_shared<Node>(Node(item));
			printf("HELPME \n");
			return;
			}

			if (sibl->sibling->itemsInNode < 5) {
				Insert(sibl->sibling, item);
				return;
			}

			sibl = sibl->sibling;
		}
		printf("HELL\n");
		if (node->son) {
			Insert(node->son, item);
		}
		else {
			node->son = std::make_shared<Node>(Node(item));
		}
	}
}

template <class Q, class O>
void TTree<Q, O>::Inorder(const std::shared_ptr<Node>& node) {
	if (node->itemsInNode) {
```

```
                node->data.sort();
                for (const auto& i: node->data) {
                        i->Print();
                }
                printf("\n");
                if (node->sibling) {
                        Inorder(node->sibling);
                }
                if (node->son) {
                        Inorder(node->son);
                }

        }
}

template <class Q, class O>
void TTree<Q, O>::InorderP(const std::shared_ptr<Node>& node) {
        if (node->itemsInNode) {
                node->data.sort_par();
                for (const auto& i: node->data) {
                        i->Print();
                }
                printf("\n");
                if (node->sibling) {
                        InorderP(node->sibling);
                }
                if (node->son) {
                        InorderP(node->son);
                }

        }
}

template <class Q, class O>
void TTree<Q, O>::RemByType(std::shared_ptr<Node>& node, const int& type) {
        if (node->itemsInNode) {
                for (int i = 0; i < 5; i++) {
                        auto iter = node->data.begin();
                        for (int k = 0; k < node->data.GetLength(); k++) {
                                if (iter->type() == type) {
                                        node->data.Pop(k + 1);
                                        node->itemsInNode--;
                                        break;
                                }
                                ++iter;
                        }
                }

                if (node->sibling) {
                        RemByType(node->sibling, type);
                }
                if (node->son) {
                        RemByType(node->son, type);
                }
        }
}
```

```
template <class Q, class O>
void TTree<Q, O>::RemLesser(std::shared_ptr<Node>& node, const double& sqr) {
	if (node->itemsInNode) {
		for (int i = 0; i < 5; i++) {
			auto iter = node->data.begin();

			for (int k = 0; k < node->data.GetLength(); k++) {
				if (iter->Square() < sqr) {
					node->data.Pop(k + 1);
					node->itemsInNode--;
					break;
				}
				++iter;
			}
		}

		if (node->sibling) {
			RemLesser(node->sibling, sqr);
		}
		if (node->son) {
			RemLesser(node->son, sqr);
		}

	}
}
#endif
```

TQueue.h

```
#ifndef TQUEUE_H
#define TQUEUE_H

#include <functional>
#include <iostream>
#include <memory>
#include <future>
#include <thread>

#include "TAllocationBlock.h"

template <class T> class TQueue {
 private:
  class TNode {
  public:
    TNode();
    TNode(const std::shared_ptr<T>&);
    auto GetNext() const;
    auto GetItem() const;
    std::shared_ptr<T> item;
    std::shared_ptr<TNode> next;

    void* operator new(size_t);
    void operator delete(void*);
    static TAllocationBlock nodeAllocator;
  };

  template <class N, class M>
```

```
  class TIterator {
 private:
  N nodePtr;
 public:
  TIterator(const N&);
  std::shared_ptr<M> operator* ();
  std::shared_ptr<M> operator-> ();
  void operator ++ ();
  bool operator == (const TIterator&);
  bool operator != (const TIterator&);
 };

 int length;

 std::shared_ptr<TNode> head;
 std::shared_ptr<TNode> tail;
 auto psort(std::shared_ptr<TNode>&);
 auto parts(std::shared_ptr<TNode>&);
 auto sort_parpar(std::shared_ptr<TNode>&);

public:
 TQueue();
 bool Push(const std::shared_ptr<T>&);
 bool Pop(const int);
 bool IsEmpty() const;
 int GetLength() const;

 void sort();
 void sort_par();

 TIterator<std::shared_ptr<TNode>, T> begin() {return TIterator<std::shared_ptr<TNode>, T>(head->next);};
 TIterator<std::shared_ptr<TNode>, T> end() {return TIterator<std::shared_ptr<TNode>, T>(nullptr);};

};

#include "TQueue.hpp"
#include "TIterator.h"
#endif
```

TQueue.hpp

```
#ifdef TQUEUE_H

#include <unistd.h>
#include <future>
#include <thread>
#include <functional>

template <class T>
TAllocationBlock TQueue<T>::TNode::nodeAllocator(sizeof(TQueue<T>::TNode), 100);

template <class T>
void* TQueue<T>::TNode::operator new(size_t size) {
        return nodeAllocator.Allocate();
}
```

```
template <class T>
void TQueue<T>::TNode::operator delete(void* ptr) {
	nodeAllocator.Deallocate(ptr);
}
//TNode private class for Queue usage
template <class T> TQueue<T>::TNode::TNode() {
  item = std::shared_ptr<T>();
  next = nullptr;
}

template <class T> TQueue<T>::TNode::TNode(const std::shared_ptr<T>& obj) {
	item = obj;
	next = nullptr;
}

template <class T> auto TQueue<T>::TNode::GetNext() const {
  return this->next;
}

template <class T> auto TQueue<T>::TNode::GetItem() const {
  return this->item;
}
//public for TNTree as well
template <class T> TQueue<T>::TQueue() {
	head = std::make_shared<TNode>();
	tail = std::make_shared<TNode>();
	length = 0;
}

template <class T> bool TQueue<T>::IsEmpty() const {
	return this->length == 0;
}

template <class T> int TQueue<T>::GetLength() const {
  return this->length;
}

template <class T> bool TQueue<T>::Push(const std::shared_ptr<T>& obj) {
  auto Nitem = std::make_shared<TNode>(obj);
  std::swap(Nitem->next, head->next);
  std::swap(head->next, Nitem);
  length++;

  return true;
}

template <class T>
bool TQueue<T>::Pop(int pos) {
	if (pos < 1 || pos > length || IsEmpty())
		return false;
	if (pos == 1) {
		if (IsEmpty())
			return false;
		else
		head->next = std::move(head->next->next);
		length--;
```

```
                return true;
        }
        auto iter = head->next;
        int i = 0;

        while (i < pos - 2) {
                iter = iter->next;
                i++;
        }

        iter->next = std::move(iter->next->next);
        length--;

        return true;
}
//sorting
template <class T> void TQueue<T>::sort() {
        head->next = psort(head->next);
}

template <class T> auto TQueue<T>::psort(std::shared_ptr<TNode>& head) {
        if (head == nullptr || head->next == nullptr) {
                return head;
        }

        auto p1 = parts(head);
        auto leftPart = p1->next;
        auto rightPart = head;

        p1->next = nullptr;

        if (leftPart == nullptr) {
                leftPart = head;
                rightPart = head->next;
                head->next = nullptr;
        }

        rightPart = psort(rightPart);
        leftPart = psort(leftPart);
        auto iter = leftPart;
        while (iter->next != nullptr) {
                iter = iter->next;
        }

        iter->next = rightPart;

        return leftPart;
}

template <class T>
auto TQueue<T>::parts(std::shared_ptr<TNode>& head) {
        if (head->next->next == nullptr) {
                if (head->next->GetItem()->Square() > head->GetItem()->Square()) {
                return head->next;
                }
```

```
		else {
		return head;
		}
	}
	else {
		auto i = head->next;
		auto check = head;
		auto lastchange = (check->next->GetItem()->Square() >= check->GetItem()->Square()) ?
check->next : check;

		while ((i != nullptr) && (i->next != nullptr)) {
			if (i->next->GetItem()->Square() >= check->GetItem()->Square()) {
				if (i->next == lastchange->next) {
					lastchange = lastchange->next;
				}
			}
			i = i->next;
		}
		return lastchange;
	}

}

template <class T>
void TQueue<T>::sort_par() {
	head->next = sort_parpar(head->next);
}


template <class T> auto TQueue<T>::sort_parpar(std::shared_ptr<TNode>& head) {
	if (head == nullptr || head->next == nullptr) {
		return head;
	}

	auto p1 = parts(head);
	auto leftPart = p1->next;
	auto rightPart = head;

	p1->next = nullptr;

	if (leftPart == nullptr) {
		leftPart = head;
		rightPart = head->next;
		head->next = nullptr;
	}

	std::packaged_task<std::shared_ptr<TNode>(std::shared_ptr<TNode>&)>
		task1(std::bind(&TQueue<T>::sort_parpar, this, std::placeholders::_1));
	std::packaged_task<std::shared_ptr<TNode>(std::shared_ptr<TNode>&)>
		task2(std::bind(&TQueue<T>::sort_parpar, this, std::placeholders::_1));
	auto rightPartHandle = task1.get_future();
	auto leftPartHandle = task2.get_future();

	std::thread(std::move(task1), std::ref(rightPart)).join();
	rightPart = rightPartHandle.get();
	std::thread(std::move(task2), std::ref(leftPart)).join();
```

```
            leftPart = leftPartHandle.get();

            auto iter = leftPart;
            while (iter->next != nullptr) {
                    iter = iter->next;
            }

            iter->next = rightPart;

            return leftPart;
}



#endif
```

main.cpp

```
#include <iostream>

#include "TQueue.h"
#include "TTree.h"

#include "Octagon.h"
#include "Hexagon.h"
#include "Triangle.h"

int main(void) {
            TTree<TQueue<Figure>, std::shared_ptr<Figure>> tree;
            std::string c;
            int c1;

   std::cout << "'help' for help" << std::endl;
   while (1) {
      std::cin.clear();
      std::cin.sync();
      std::cout << ">";
      std::cin >> c;

      if (c == "e" || c == "exit") {
         break;
      }
      else if (c == "t" || c == "triangle") {
         Triangle fig;

         std::cin >> fig;
         tree.insert(std::make_shared<Triangle>(fig));
      }
      else if (c == "h" || c == "hexagon") {
         Hexagon fig;

         std::cin >> fig;
         tree.insert(std::make_shared<Hexagon>(fig));
      }
      else if (c == "o" || c == "octagon") {
         Octagon fig;
```

```
            std::cin >> fig;
            tree.insert(std::make_shared<Octagon>(fig));
        }
        else if (c == "r" || c == "remove") {
            std::cout << "'1' - type\n'2' - square\n";
            std::cin >> c1;
            if (c1 == 1) {
                std::cout << "'1' triangle\n'2' octagon\n'3' hexagon\n";
                std::cout << "Type:" << std::endl;
                std::cin >> c1;
                tree.removeByType(c1);
            } else if (c1 == 2) {
                double sqr = 0.0;
                std::cout << "Remove all squares less than:" << std::endl;
                std::cin >> sqr;
                tree.removeLesser(sqr);
            }
        }
        else if (c == "ps" || c == "prints") {
                                if (!tree.IsEmpty()) {
                                        tree.inorder();
                                }
                                else {
                                        std::cout << "The tree is empty" << std::endl;
                                }
        }
        else if (c == "pp" || c == "printp") {
                                if (!tree.IsEmpty()) {
                                        tree.inorderp();
                                }
                                else {
                                        std::cout << "The tree is empty" << std::endl;
                                }
        }
        else if (c == "d" || c == "destroy") {
                                if (!tree.IsEmpty()) {
                 tree.Des();
            }
            else {
              std::cout << "The tree is already empty" << std::endl;
            }
        }
        else if (c == "help") {
            std::cout << "'help'              - help;" << std::endl;
                                std::cout << "'t'   or 'triangle'  - insert triangle" << std::endl;
                                std::cout << "'o'   or 'octagon'   - insert octagon;" << std::endl;
                                std::cout << "'h'   or 'hexagon'   - insert hexagon;" << std::endl;
            std::cout << "'r'   or 'remove'    - remove figure;" << std::endl;
            std::cout << "'d'   or 'destroy'   - delete the tree;" << std::endl;
            std::cout << "'ps'  or 'prints'    - print the tree (normal sorting);" << std::endl;
            std::cout << "'pp'  or 'printp'    - print the tree (parallel sorting);" << std::endl;
            std::cout << "'e'   or 'exit'      - exit." << std::endl;
                        }
    }
    return 0;
}
```

Результаты:

```
user@lubuntu:~/INF/ooplabs1819/lab6,7,8/kindofworks/mehvfinal$ ./no
'help' for help
>t
Enter sides of your triangle:
2 4 6
>o
PLEASE, one side of your octagon:
3
>ps
Triangle: a = 2, b = 4, c = 6; Square = 0
Octagon: a = 3; Square = 43.4558

>r
'1' - type
'2' - square
2
Remove all squares less than:
40
>ps
Octagon: a = 3; Square = 43.4558

>r
'1' - type
'2' - square
1
'1' triangle
'2' octagon
'3' hexagon
Type:
2
>ps
>e
user@lubuntu:~/INF/ooplabs1819/lab6,7,8/kindofworks/mehvfinal$
```

Выводы:

Ужас какой-то. Я думала, что вообще не сделаю.

Лабораторная работа №8

Цель работы:

- Знакомство с параллельным программированием в C++.

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

Программа должна позволять:

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

Описание:

Знакомимся с использованием threads на C++. Так как при сортировке происходит разделение очереди на две части и последовательное их сортирование, то можно эти чатси отсортировантъ одновременно при помощи потоков.

При выполнении работы были созданы следующие файлы:

Figure.h	Базовый класс фигуры
Iterator.h	Итератор
TAllocationBlock.h	Header файл для аллоктора
TAllocationBlock.cpp	Имплементация аллокатора
TQueueItemA.h	Header файл для элементов очереди
TQueueItemA.cpp	Имплементация
TQueueA.h	Header файл для очереди
TQueueA.cpp	Имплементация
Hexagon/Triangle/Octagon.h	Header файлы для классов фигур
Hexagon/Triangle/Octagon.cpp	Имплементация funcitons фигур
TTree.h	Header файл для дерева
TTree.hpp	Имплементация funcitons дерева
TQueue.h	Header файл для очереди
TQueue.hpp	Имплементация funcitons очереди
main.cpp	Использование контейнера
Makefile	makefile

В листинге кода приведены только те файлы, в которых произошло изменение или новые файлы.

Листинг кода:

Весь код такой же, что и в 7 лабораторной работе. Вообще я делала сразу 6ую, 7ую и 8ую, чтобы легче было.

Только в Makefile -lpthread надо не забыть и добавить.

Результаты:

```
user@lubuntu:~/INF/ooplabs1819/lab6,7,8/kindofworks/mehvfinal$ ./no
'help' for help
>help
'help'              - help;
't'   or 'triangle'  - insert triangle
'o'   or 'octagon'   - insert octagon;
'h'   or 'hexagon'   - insert hexagon;
'r'   or 'remove'    - remove figure;
'd'   or 'destroy'   - delete the tree;
'ps'  or 'prints'    - print the tree (normal sorting);
'pp'  or 'printp'    - print the tree (parallel sorting);
'e'   or 'exit'      - exit.
>o
PLEASE, one side of your octagon:
7
>t
Enter sides of your triangle:
2 4 5
>pp
Triangle: a = 2, b = 4, c = 5; Square = 3.79967
Octagon: a = 7; Square = 236.593

>e
user@lubuntu:~/INF/ooplabs1819/lab6,7,8/kindofworks/mehvfinal$
```

Выводы:

В курсе Операционных Систем потоки уже были. Поэтому тут легче понимать было. Вот.

Лабораторная работа №9

Цель работы:

- Знакомство с лямбда-выражениями.

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 контейнером 1-го уровня:
  - Генерация фигур со случайным значением параметров;
  - Печать контейнера на экран;
  - Удаление элементов со значением площади меньше определенного числа;
- В контейнер второго уровня поместить цепочку команд.
- Реализовать цикл, который проходит по всем командам в контейнере 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

Описание:

Lambda expression/closure is a syntactic shortcut for a functor. One can use lambda expressions to replace functors.

Functor overloads the function call operator.

При выполнении работы были созданы следующие файлы:

Figure.h	Базовый класс фигуры
Iterator.h	Итератор
TAllocationBlock.h	Header файл для аллоктора
TAllocationBlock.cpp	Имплементация аллокатора
TQueueItem.h	Header файл для элементов очереди
TQueueItem.cpp	Имплементация
TQueue.h	Header файл для очереди
TQueue.cpp	Имплементация
Hexagon/Triangle/Octagon.h	Header файлы для классов фигур
Hexagon/Triangle/Octagon.cpp	Имплементация funcitons фигур
TNTreeItem.h	Header файл для узлов дерева. Нужен, чтобы связать узлы и фигуры, которые туда будет класть
TNTreeItem.cpp	Имплементация funcitons узла
TNTree.h	Header файл для самого дерева
TNTree.cpp	Имплементация funcitons дерева
main.cpp	Использование контейнера
Makefile	makefile

В листинге кода приведены только те файлы, в которых произошло изменение или новые файлы.

Листинг кода:

main.cpp

```
#include <iostream>
#include <string>
#include <cstdint>
#include <future>
#include <mutex>
#include <thread>
#include <functional>
#include <random>
#include <cstdlib>
#include <cstring>

#include "TNTree.h"
#include "TNTreeItem.h"
#include "TQueue.h"
#include "Figure.h"

int main(int argc, char** argv)
{
   TNTree<Figure> tree; //make an object
   std::string c;

        typedef std::function<void(void)> Command;
        TQueue<std::shared_ptr<Command>> comque;//
        std::mutex mtx;


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

                std::cout << ">Commencing Insert" << std::endl;

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

      std::default_random_engine generator(seed);
      std::uniform_int_distribution<int> distFigureParam(1, 10);

      std::shared_ptr<Figure> fig;
                std::string parent_id, id;

      for (int i = 0; i < 4; i++) {
                        switch (i)      {
                                case (0): {
                                        parent_id = "1";
                                        id = "1";

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

                                        while((side_a+side_b < side_c) || (side_a+side_c< side_b) ||
(side_b+side_c < side_a)) {
                                                side_a = distFigureParam(generator);
                                                side_b = distFigureParam(generator);
```

```
                                        side_c = distFigureParam(generator);
                                }

                                fig = std::make_shared<Triangle> (side_a, side_b, side_c);
                                tree.Insert(parent_id, id, fig);

                                std::cout << ">>Triangle Inserted" << std::endl;
                                break;
                        }
                        case (1): {
                                parent_id = "1";
                                id = "2";

                                size_t side_a = distFigureParam(generator);

                                fig = std::make_shared<Octagon> (side_a);
                                tree.Insert(parent_id, id, fig);

                                std::cout << ">>Octagon Inserted" << std::endl;
                                break;
                        }
                        case (2): {
                                parent_id = "1";
                                id = "3";

                                size_t side_a = distFigureParam(generator);

                                fig = std::make_shared<Hexagon> (side_a);
                                tree.Insert(parent_id, id, fig);

                                std::cout << ">>Hexagon Inserted" << std::endl;
                                break;
                        }
                        case (3): {
                                parent_id = "3";
                                id = "4";

                                size_t side_a = distFigureParam(generator);

                                fig = std::make_shared<Octagon> (side_a);
                                tree.Insert(parent_id, id, fig);

                                std::cout << ">>Octagon Inserted" << std::endl;
                                break;
                        }
                }
        }
        std::cout << '\n';
    };

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

std::cout << ">Commencing Print" << std::endl;

        if (!tree.IsEmpty()) {
```

```
        tree.PrintTree();
    }
    else {
        std::cout << "The tree is empty" << std::endl;
    }
    std::cout << ">>Completed Print\n" << std::endl;

        };

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

    std::cout << ">Commencing Print Values" << std::endl;

                if (!tree.IsEmpty()) {
        tree.PrintTreeVal();
    }
    else {
        std::cout << "The tree is empty" << std::endl;
    }

                std::cout << ">>Completed Print Values\n" << std::endl;
        };

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

    std::cout << ">Commencing Remove more than 150" << std::endl;

                double Sq = 150;
                if (!tree.IsEmpty()) {
          tree.RemS(Sq);
        }
        else {
          std::cout << "The tree is empty" << std::endl;
        }

    std::cout << ">>Completed Remove more than 150\n" << std::endl;
        };

  comque.Push(std::shared_ptr<Command>(&PrintoVal, [](Command*){}));
  comque.Push(std::shared_ptr<Command>(&RemSq, [](Command*){}));
  comque.Push(std::shared_ptr<Command>(&PrintoVal, [](Command*){}));
  comque.Push(std::shared_ptr<Command>(&Printo, [](Command*){}));
  comque.Push(std::shared_ptr<Command>(&In, [](Command*){}));


  while (!comque.IsEmpty()) {
                std::shared_ptr<Command> com = comque.Lasto();
                std::future<void> ft = std::async(*com);
    ft.get();
                comque.Pop();
        }

  return 0;
}
```

Результаты:

```
user@lubuntu:~/INF/ooplabs1819/lab9/lab9$ ./no
>Commencing Insert
>>Triangle Inserted
>>Octagon Inserted
>>Hexagon Inserted
>>Octagon Inserted

>Commencing Print
1
  2
  3
    4
>>Completed Print

>Commencing Print Values
Triangle: a = 6, b = 6, c = 3; Square = 8.71421
  Octagon: a = 1; Square = 4.82843
  Hexagon: a = 2; Square = 10.3923
    Octagon: a = 3; Square = 43.4558
>>Completed Print Values

>Commencing Remove more than 150
>>Completed Remove more than 150

>Commencing Print Values
Triangle: a = 6, b = 6, c = 3; Square = 8.71421
  Octagon: a = 1; Square = 4.82843
  Hexagon: a = 2; Square = 10.3923
    Octagon: a = 3; Square = 43.4558
>>Completed Print Values

user@lubuntu:~/INF/ooplabs1819/lab9/lab9$
```

Выводы:

Интересно, но не весело. И потоки тут. Хорошо. Наверное.

Заключение

Курс по Объектно-Ориентированному программированию оказался очень интересным, но местами сложным. Теория, порой, нелегко давалась. А имплементация вещей, которые, по сути, в лабораторной работе не используются, не делала моё существование проще.

Я дерево сделала! И оно рабочее. В прошлом году у меня тоже было NTree в некоторых работах, и оно меня тогда знатно побило. Сейчас вроде бы понятнее стало.

Ещё здорово, что отчёт только в конце сдавать нужно. Это хорошо. Пришёл, лабораторную защитил, всё.

Конечно, остались немного странные для понимания вещи... Но я знаю, что в скором времени всё разъяснится. Взойдёт солнце. И мой апельсин на подоконнике не засохнет из-за батареи под ним и нехватки воды.

Спасибо. Мне понравилось :)