Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Отчет по лабораторным работам
по курсу
"Объектно-ориентированное программирование"

Студент: Соболев А. Ю.
Преподаватель: Поповкин А.В.
Группа: М8О-207Б
Дата:
Оценка:

Москва, 2017

Оглавление

1. ЛАБОРАТОРНАЯ РАБОТА №1

1.1. Цель работы

Целью лабораторной работы является:

- 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.

1.2. Задание

Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно вариантов задания.

Классы должны удовлетворять следующим правилам:

- Должны иметь общий родительский класс Figure.
- Должны иметь общий виртуальный метод Print, печатающий параметры фигуры и ее тип в стандартный поток вывода cout.
- Должный 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.

Фигуры: Трапеция, Ромб, Пятиугольник

1.3. Описание

Классы в C++ — это абстракция описывающая методы, свойства, ещё не существующих объектов.

Объекты — конкретное представление абстракции, имеющее свои свойства и методы.

Созданные объекты на основе одного класса называются экземплярами этого класса. Эти объекты могут иметь различное поведение, свойства, но все равно будут являться объектами одного класса. В ООП существует три основных принципа построения классов:

1. Инкапсуляция — это свойство, позволяющее объединить в классе и данные, и методы, работающие с ними и скрыть детали реализации от пользователя.

2. Наследование — это свойство, позволяющее создать новый класс-потомок на основе уже существующего, при этом все характеристики класса родителя
присваиваются классу-потомку.

3. Полиморфизм — свойство классов, позволяющее использовать объекты классов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Перегрузка операторов — один из способов реализации полиморфизма, заключающийся в возможности одновременного существования в одной области видимости нескольких различных вариантов применения оператора, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются.

Дружественная функция — это функция, которая не является членом класса, но имеет доступ к членам класса, объявленным в полях private или protected.

1.4. Исходный код

main.cpp

```
#include <cstring>

#include "Trapezium.h"
#include "Pentagon.h"
#include "Rhombus.h"

int main(int argc, char** argv)
{
    const int size = 16;
    char s[size];
    Trapezium *ptr_tr = nullptr;
    Rhombus   *ptr_rh = nullptr;
    Pentagon  *ptr_pn = nullptr;

    std::cout << "Use 'help' or 'h' to get help." << std::endl;

    for(;;) {
        std::cout << "> ";
        std::cin.getline(s, size);
        std::cin.clear();
```

```
        std::cin.sync();

        if (!strcmp(s, "create_trapezium") || !strcmp(s, "cr_tr")) {
            if (ptr_tr != nullptr) delete ptr_tr;

            std::cout << "Enter smaller base, larger base, left side
and right side:" << std::endl;
            ptr_tr = new Trapezium(std::cin);
        } else if (!strcmp(s, "create_rhombus") || !strcmp(s,
"cr_rh")) {
            if (ptr_rh != nullptr) delete ptr_rh;

            std::cout << "Enter side and smaller angle:" << std::endl;
            ptr_rh = new Rhombus(std::cin);
        } else if (!strcmp(s, "create_pentagon") || !strcmp(s,
"cr_pn")) {
            if (ptr_pn != nullptr) delete ptr_pn;

            std::cout << "Enter side:" << std::endl;
            ptr_pn = new Pentagon(std::cin);
        } else if (!strcmp(s, "print_trapezium") || !strcmp(s,
"pr_tr")) {
            if(ptr_tr == nullptr) {
                std::cout << "Trapezium doesn't exist." << std::endl;
            } else {
                ptr_tr->Print();
            }
        } else if (!strcmp(s, "print_rhombus") || !strcmp(s, "pr_rh"))
{
            if(ptr_rh == nullptr) {
                std::cout << "Rhombus doesn't exist." << std::endl;
            } else {
                ptr_rh->Print();
            }
        } else if (!strcmp(s, "print_pentagon") || !strcmp(s,
"pr_pn")) {
            if(ptr_pn == nullptr) {
                std::cout << "Pentagon doesn't exist." << std::endl;
            } else {
                ptr_pn->Print();
            }
        } else if (!strcmp(s, "square_trapezium") || !strcmp(s,
"sq_tr")) {
            if(ptr_tr == nullptr) {
                std::cout << "Trapezium doesn't exist." << std::endl;
```

```
            } else {
                std::cout << "Square: " << ptr_tr->Square() <<
std::endl;
            }
        } else if (!strcmp(s, "square_rhombus") || !strcmp(s,
"sq_rh")) {
            if(ptr_rh == nullptr) {
                std::cout << "Rhombus doesn't exist." << std::endl;
            } else {
                std::cout << "Square: " << ptr_rh->Square() <<
std::endl;
            }
        } else if (!strcmp(s, "square_pentagon") || !strcmp(s,
"sq_pn")) {
            if(ptr_pn == nullptr) {
                std::cout << "Pentagon doesn't exist." << std::endl;
            } else {
                std::cout << "Square: " << ptr_pn->Square() <<
std::endl;
            }
        } else if (!strcmp(s, "quit") || !strcmp(s, "exit") ||
!strcmp(s, "q")) {
            if (ptr_tr != nullptr) {
                delete ptr_tr;
            }
            if (ptr_rh != nullptr) {
                delete ptr_rh;
            }
            if (ptr_pn != nullptr) {
                delete ptr_pn;
            }
            break;
        } else if (!strcmp(s, "help") || !strcmp(s, "h")) {
            std::cout << " 'create_trapezium' and 'cr_tr' create new
trapezium with your parameters." << std::endl;
            std::cout << " 'create_rhombus' and 'cr_rh'   create new
rhombus with your parameters." << std::endl;
            std::cout << " 'create_pentagon' and 'cr_pn'  create new
pentagon with your parameters." << std::endl;
            std::cout << " 'print_trapezium' and 'pr_tr'  output
parameters of trapezium." << std::endl;
            std::cout << " 'print_rhomb' and 'pr_rh'      output
parameters of rhombus."  << std::endl;
            std::cout << " 'print_pentagon' and 'pr_pn'   output
parameters of pentagon."  << std::endl;
```

```
            std::cout << " 'square_trapezium' and 'sq_tr' output
square of trapezium." << std::endl;
            std::cout << " 'square_rhombus' and 'sq_rh'   output
square of rhombus." << std::endl;
            std::cout << " 'square_pentagon' and 'sq_pn'  output
square of pentagon." << std::endl;
            std::cout << " 'quit', 'exit' and 'q'         exit the
program." << std::endl;
        }
    }
    return 0;
}
```

Figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

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

Pentagon.h

```
#ifndef PENTAGON_H
#define PENTAGON_H

#include <iostream>

#include "Figure.h"

class Pentagon : public Figure{
public:
    Pentagon();
    Pentagon(std::istream &is);
    Pentagon(int side);
    Pentagon(const Pentagon& orig);

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

    virtual ~Pentagon();
private:
    int side;
```

```
};

#endif /*PENTAGON_H */
```

Pentagon.cpp

```
#include <iostream>
#include <cmath>

#include "Pentagon.h"

#define PI 3.14159265

Pentagon::Pentagon() : Pentagon(0) {
    std::cout << "Pentagon created: default." << std::endl;
}

Pentagon::Pentagon(int a) : side(a) {
    std::cout << "Pentagon created: " << side << std::endl;
}

Pentagon::Pentagon(std::istream &is) {
    is >> side;
    if(side < 0) {
        std::cerr << "Error: sides should be greater than 0." <<
std::endl;
    }
}

Pentagon::Pentagon(const Pentagon& orig) {
    std::cout << "Pentagon copy created." << std::endl;
    side = orig.side;
}

double Pentagon::Square() {
    return (double)(5 * side * side / (4 * (double)tan(36 * (PI /
180))));
}

void Pentagon::Print() {
    std::cout << "Sides = " << side << std::endl;
}

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

Rhombus.h

```
#ifndef RHOMBUS_H
#define RHOMBUS_H

#include <iostream>

#include "Figure.h"

class Rhombus : public Figure{
public:
    Rhombus();
    Rhombus(std::istream &is);
    Rhombus(int side, int angle);
    Rhombus(const Rhombus& orig);

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

    virtual ~Rhombus();
private:
    int side;
    int angle;
};

#endif /*RHOMBUS_H */
```

Rhombus.cpp

```
#include <iostream>
#include <cmath>

#include "Rhombus.h"

#define PI 3.14159265

Rhombus::Rhombus() : Rhombus(0, 0) {
    std::cout << "Rhombus created: default." << std::endl;
}

Rhombus::Rhombus(int a, int ang) : side(a), angle(ang) {
    std::cout << "Rhombus created: " << side << ", " << angle <<
std::endl;
}

Rhombus::Rhombus(std::istream &is) {
    is >> side;
```

```
    is >> angle;
    if(side < 0) {
        std::cerr << "Error: side should be greater than 0." <<
std::endl;
    }
    if(angle < 0 || angle > 180) {
        std::cerr << "Error: angle should be greater than 0 and
smaller than 180." << std::endl;
    }
}

Rhombus::Rhombus(const Rhombus& orig) {
    std::cout << "Rhombus copy created." << std::endl;
    side = orig.side;
    angle = orig.angle;
}

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

void Rhombus::Print() {
    std::cout << "Side = " << side << ", angle = " << angle <<
std::endl;
}

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

Trapezium.h

```
#ifndef TRAPEZIUM_H
#define TRAPEZIUM_H

#include <iostream>

#include "Figure.h"

class Trapezium : public Figure {
public:
    Trapezium();
    Trapezium(std::istream &is);
    Trapezium(int sm_base, int lg_base, int lf_side, int rt_side);
    Trapezium(const Trapezium& orig);

    double Square() override;
```

```
    void Print() override;

    virtual ~Trapezium();
private:
    int sm_base;
    int lg_base;
    int lf_side;
    int rt_side;
};

#endif /*TRAPEZIUM_H */
```

Trapezium.cpp

```
#include <iostream>
#include <cmath>

#include "Trapezium.h"

Trapezium::Trapezium() : Trapezium(0, 0, 0, 0) {
    std::cout << "Trapezium created: default." << std::endl;
}

Trapezium::Trapezium(int sb, int lb, int ls, int rs) : sm_base(sb),
lg_base(lb), lf_side(ls), rt_side(rs) {
    std::cout << "Trapezium created: " << sm_base << ", " << lg_base
<< ", " << lf_side << ", " << rt_side << std::endl;
}

Trapezium::Trapezium(std::istream &is) {
    is >> sm_base;
    is >> lg_base;
    is >> lf_side;
    is >> rt_side;

    if(sm_base < 0 || lg_base < 0 || lf_side < 0 || rt_side < 0) {
        std::cerr << "Error: sides should be > 0." << std::endl;
    }
}

Trapezium::Trapezium(const Trapezium& orig) {
    std::cout << "Trapezium copy created" << std::endl;

    sm_base = orig.sm_base;
    lg_base = orig.lg_base;
    lf_side = orig.lf_side;
```

```
    rt_side = orig.rt_side;
}

double Trapezium::Square() {
    if (lg_base > sm_base) {
        return (double)((sm_base + lg_base) / 2 * sqrt(lf_side *
lf_side - (1 / 4) * pow(((lg_base - sm_base) * (lg_base - sm_base) +
lf_side * lf_side - rt_side * rt_side) / (lg_base - sm_base), 2)));
    } else {
        std::cerr << "Error: bigger base is smaller than smaller base"
<< std::endl;
    }
    return 0.0;
}

void Trapezium::Print() {
    std::cout << "Smaller base = " << sm_base << ", larger base = " <<
lg_base << ", left side = " <<  lf_side << ", right side = " <<
rt_side << std::endl;
}

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

1.5. Тестирование

```
➜  lab_1 ./run
Use 'help' or 'h' to get help.
> h
 'create_trapezium' and 'cr_tr' create new trapezium with your
parameters.
 'create_rhombus' and 'cr_rh'   create new rhombus with your
parameters.
 'create_pentagon' and 'cr_pn'  create new pentagon with your
parameters.
 'print_trapezium' and 'pr_tr'  output parameters of trapezium.
 'print_rhomb' and 'pr_rh'      output parameters of rhombus.
 'print_pentagon' and 'pr_pn'   output parameters of pentagon.
 'square_trapezium' and 'sq_tr' output square of trapezium.
 'square_rhombus' and 'sq_rh'   output square of rhombus.
 'square_pentagon' and 'sq_pn'  output square of pentagon.
 'quit', 'exit' and 'q'         exit the program.
> cr_tr
Enter smaller base, larger base, left side and right side:
```

```
12 24 34 56
> > pr_tr
Smaller base = 12, larger base = 24, left side = 34, right side = 56
> sq_tr
Square: 612
> cr_rh
Enter side and smaller angle:
12 30
> > pr_tr
Smaller base = 12, larger base = 24, left side = 34, right side = 56
> pr_rh
Side = 12, angle = 30
> sq_rh
Square: 72
> cr_pn
Enter side:
12
> > pr_pn
Sides = 12
> sq_pn
Square: 247.749
> q
Trapezium deleted.
Rhombus deleted.
Pentagon deleted.
```

1.6. Выводы

Данная работа позволяет ознакомиться с общими принципами объектно-ориентированного программирования и языка C++ в частности. Изучены такие важнейшие фундаментальные понятия, как Классы, Объекты, Наследование, Полиморфизм. Применение их на практике позволяет глубже понять как использовать их при написании реальных программ.

2. ЛАБОРАТОРНАЯ РАБОТА №2

2.1. Цель работы

Целью лабораторной работы является:

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

2.2. Задание

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

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream (<<). Оператор должен распечатывать параметры фигуры (тип фигуры, длины сторон, радиус и т.д).
- Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream (>>).

Оператор должен вводить основные параметры фигуры (длины сторон, радиус и т.д).

- Классы фигур должны иметь операторы копирования (=).
- Классы фигур должны иметь операторы сравнения с такими же фигурами (==).
- Класс-контейнер должен 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 (опередяется структурой контейнера).
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера (опередяется 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 расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

- Стандартные контейнеры std.
- Шаблоны (template).
- Различные варианты умных указателей (shared_ptr, weak_ptr).

Программа должна позволять:

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

2.3. Описание

Дерево общего вида является удобным способом хранения различных данных. В отличие от бинарного дерева у дерева общего вида на у каждой вершины может быть более двух потомков. Реализация данного дерева представляет класс, имеющий конструктор, деструктор, методы работы с деревом в целом и приватное поле, являющееся указателем на вершину дерева. Каждый элемент дерева является объектом TNTreeItem, описанным в отдельном файле, содержащим поля trapezium, сохраняющим трапецию, id, хранящем уникальный идентификатор узла в дереве и указатели на первого потомка и брата, если они есть.

2.4. Исходный код

main.cpp

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

#include "TNTree.h"
#include "Trapezium.h"

int main(int argc, char** argv)
{
    TNTree *tree = new TNTree();
    std::string action;

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

        if (action == "q" || action == "quit") {
            if (tree != nullptr) {
```

```
                delete tree;
            }
            break;
        } else if (action == "i" || action == "insert") {
            size_t size_a, size_b, size_c, size_d;
            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 Trapezium's sides:\n";
            if (!(std::cin >> size_a >> size_b >> size_c >> size_d)) {
                std::cout << "Incorrect value." << std::endl;
                continue;
            }
            Trapezium* n = new Trapezium(size_a, size_b, size_c,
size_d);
            tree->Insert(tree, parent_id, id, *n);
            delete n;
        } else if (action == "r" || action == "Remove") {
            std::string id;
            if (!(std::cin >> id)) {
                std::cout << "Incorrect value." << std::endl;
                continue;
            }
            tree->Remove(id);
        } else if (action == "d" || action == "destroy") {
            delete tree;
            tree = new TNTree();
            std::cout << "The tree was deleted." << std::endl;
        } else if (action == "p" || action == "print") {
            if (!tree->IsEmpty()) {
                tree->PrintTree(tree->getHead(), 0);
            }
            else {
                std::cout << "The tree is empty." << std::endl;
            }
        } else if (action == "pi" || action == "printitem") {
            std::string id;
            std::cout << "Enter item id: ";
            std::cin >> id;

            tree->PrintItem(tree, id);
        } else if (action == "help" || action == "h") {
```

```
            std::cout << "'h'  or 'help'      - display the help." <<
std::endl;
            std::cout << "'r'  or 'Remove'    - Remove the trapezium."
<< std::endl;
            std::cout << "'d'  or 'destroy'   - delete the tree." <<
std::endl;
            std::cout << "'p'  or 'print'     - output the tree." <<
std::endl;
            std::cout << "'i'  or 'insert'    - insert a trapezium
into the tree." << std::endl;
            std::cout << "'pi' or 'printitem' - print item by id." <<
std::endl;
            std::cout << "'q'  or 'quit'      - exit the program." <<
std::endl;
        }
    }

    return 0;
}
```

TNTree.h

```
#ifndef TNTREE_H
#define TNTREE_H

#include <iostream>
#include <string>

#include "Trapezium.h"
#include "TNTreeItem.h"

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

    void Insert(TNTree* tree, std::string parent_id, std::string id,
Trapezium &trapezium);
    void Remove(std::string id);
    void node_remove(TNTreeItem* TNTree, std::string id);
    void PrintTree(TNTreeItem* treeit, size_t num);
    void PrintItem(TNTree* tree, std::string id);
    void SetNull(TNTreeItem* t);
    TNTreeItem* getHead();
    bool IsEmpty();

    virtual ~TNTree();
```

```
private:
    TNTreeItem* head;

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

TNTree.cpp

```
#include <string>

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

TNTree::TNTree() {
    head = nullptr;
}

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

void TNTree::Insert(TNTree* tree, std::string parent_id, std::string
id, Trapezium &trapezium) {
    if (!tree->head) {
        tree->head = new TNTreeItem(id, trapezium);
        return;
    }
    else {
        TNTreeItem *parent_node = FindNode(tree->head, parent_id);
        if (parent_node) {
            if (!parent_node->GetSon()) {
                parent_node->SetSon(new TNTreeItem(id, trapezium));
            }
            else {
                TNTreeItem *brother = parent_node->GetSon();
                while (brother->GetBrother()) {
                    brother = brother->GetBrother();
                }
                brother->SetBrother(new TNTreeItem(id, trapezium));
            }
        }
        else {
            std::cout << "Error: no such parent_id." << '\n';
        }
    }
}
```

```
TNTreeItem* TNTree::FindNode(TNTreeItem* treeItem, std::string id) {
    if (treeItem->getId() == id) {
        return treeItem;
    }
    if (treeItem->GetSon()) {
        TNTreeItem* tr = FindNode(treeItem->GetSon(), id);
        if (tr != nullptr) {
            return tr;
        }
    }
    if (treeItem->GetBrother()) {
        TNTreeItem* tr = FindNode(treeItem->GetBrother(), id);
        if (tr != nullptr) {
            return tr;
        }
    }
    return nullptr;
}

bool TNTree::IsEmpty() {
    return head == nullptr;
}

void TNTree::node_remove(TNTreeItem* treeItem, std::string id) {
    if (treeItem->GetSon()) {
        if (treeItem->GetSon()->getId() == id) {
            TNTreeItem *tr = treeItem->GetSon();
            treeItem->SetSon(treeItem->GetSon()->GetBrother());
            SetNull(tr->GetBrother());// = nullptr;
            delete tr;
            return;
        }
        else {
            node_remove(treeItem->GetSon(), id);
        }
    }
    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 {
            node_remove(treeItem->GetBrother(), id);
        }
    }
}

void TNTree::SetNull(TNTreeItem* t)
{
    t = nullptr;
}

TNTreeItem* TNTree::getHead()
{
    return this->head;
}

void TNTree::Remove(std::string id)
{
    if (head->getId() == id) {
        head = head->GetSon();
    } else {
        TNTree::node_remove(head, id);
   }
}

void TNTree::PrintTree(TNTreeItem* treeItem, size_t num) {
    if (treeItem) {
        for (int i = 0; i < num; ++i) {
            printf("\t");
        }
        std::cout << treeItem->getId()<< '\n';
        if (treeItem->GetSon()) {
            PrintTree(treeItem->GetSon(), num + 1);
        }
        if (treeItem->GetBrother()) {
            PrintTree(treeItem->GetBrother(), num);
        }
    }
}

void TNTree::PrintItem(TNTree* tree, std::string id)
{
    TNTreeItem* tmp = FindNode(tree->head, id);
    if(tmp) {
```

```
        std::cout << tmp->getId();
        tmp->GetTrapezium().Print();
    }
    delete tmp;
}

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

TNTreeItem.h

```
#ifndef TNTREEITEM_H
#define TNTREEITEM_H

#include <string>

#include "Trapezium.h"

class TNTreeItem {
public:
    TNTreeItem();
    TNTreeItem(std::string id, const Trapezium& trapezium);
    friend std::ostream& operator<<(std::ostream& os, const
TNTreeItem& obj);

    void SetSon(TNTreeItem* son);
    void SetBrother(TNTreeItem* brother);
    TNTreeItem* GetSon();
    TNTreeItem* GetBrother();
    Trapezium GetTrapezium() const;
    std::string getId();

    virtual ~TNTreeItem();
private:
    Trapezium trapezium;
    std::string id;
    TNTreeItem *son;
    TNTreeItem *brother;
};

#endif  /* TNTREEITEM_H */
```

TNTreeItem.cpp

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

```
#include "TNTreeItem.h"
#include "Trapezium.h"

TNTreeItem::TNTreeItem() {
    this->brother = nullptr;
    this->son = nullptr;
    this->id = "";
}

TNTreeItem::TNTreeItem(std::string id, const Trapezium& trapezium) {
    this->trapezium = trapezium;
    this->brother = nullptr;
    this->son = nullptr;
    this->id = id;
}

void TNTreeItem::SetSon(TNTreeItem* son) {
    this->son = son;
}

void TNTreeItem::SetBrother(TNTreeItem* brother) {
    this->brother = brother;
}

Trapezium TNTreeItem::GetTrapezium() const {
    return this->trapezium;
}

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

TNTreeItem* TNTreeItem::GetBrother() {
    return this->brother;
}

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

std::ostream& operator<<(std::ostream& os, const TNTreeItem& obj) {
    os << obj.trapezium << std::endl;
    return os;
}
```

```
TNTreeItem::~TNTreeItem() {
    delete brother;
    delete son;
}
```

2.5. Тестирование

```
➜  lab_2 ./run

Type 'h' or 'help' to get help.
> h
'h'  or 'help'      - display the help.
'r'  or 'Remove'    - Remove the trapezium with area s.
'd'  or 'destroy'   - delete the tree.
'p'  or 'print'     - output the tree.
'i'  or 'insert'    - insert a trapezium into the tree.
'pi' or 'printitem' - print item by id.
'q'  or 'quit'      - exit the program.
> i
Enter parent id: 0
Enter node id: 0
Enter Trapezium's sides:
1 2 3 4
> i
Enter parent id: 0
Enter node id: 1
Enter Trapezium's sides:
2 3 4 5
> p
0
     1
> pi 0
Enter item id: 0
Smaller base = 1, larger base = 2, left side = 3, right side = 4
> q
```

2.6.Выводы

Рассмотрена работа с классами и объектами на примере реализации конкретной полезной структуры данных - дерева общего вида.

3. ЛАБОРАТОРНАЯ РАБОТА №3

3.1. Цель работы

Целью лабораторной работы является:

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

3.2. Задание

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

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Класс-контейнер должен содержать объекты используя std:shared_ptr<...>.
- Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.
- Класс-контейнер должен иметь методы по получению фигуры из контейнера (опередеяется структурой контейнера).
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера (опередеяется 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 расположенны в раздельных файлах: отдельно заголовки (.h), отдельно
описание методов (.cpp).

Нельзя использовать:

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

Программа должна позволять:

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

3.3.Описание

Smart pointer — это объект, работать с которым можно как с обычным указателем, но при этом, в отличии от последнего, он предоставляет некоторый дополнительный функционал (например, автоматическое освобождение закрепленной за указателем области памяти).
Умные указатели призваны для борьбы с утечками памяти, которые сложно избежать в больших проектах. Они особенно удобны в местах, где возникают исключения, так как при последних происходит процесс раскрутки стека и уничтожаются локальные объекты.
Рассмотрим следующие умные указатели: unique_ptr, shared_ptr и weak_ptr. Все они объявлены в заголовочном файле <memory>.

unique_ptr - этот указатель пришел на смену старому и проблематичному auto_ptr. Основная проблема последнего заключается в правах владения. Объект этого класса теряет права владения ресурсом при копировании (присваивании, использовании в конструкторе копий, передаче в функцию по значению).

shared_ptr - это самый популярный и самый широкоиспользуемый умный указатель. Он начал своё развитие как часть библиотеки boost. В отличии от auto_ptr и unique_ptr, shared_ptr реализует подсчет ссылок на ресурс. Ресурс освободится тогда, когда счетчик ссылок на него будет равен 0.

weak_ptr - этот указатель также, как и shared_ptr начал свое рождение в проекте boost, затем был включен в новый стандарт. Данный класс позволяет разрушить циклическую зависимость, которая, несомненно, может образоваться при использовании shared_ptr.

3.4. Исходный код

main.cpp

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

#include "TNTree.h"

int main(int argc, char** argv)
{
    TNTree tree;
    std::string action;

    std::cout << "Type 'h' or 'help' to get help." << std::endl;
    while (1) {
        std::cin.clear();
        std::cin.sync();
```

```
std::cout << "> ";
std::cin >> action;

if (action == "q" || action == "quit") {
    break;
} else if (action == "it" || action == "insert trapezium") {
    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<Trapezium>(std::cin);
    tree.Insert(parent_id, id, fig);
} else if (action == "ir" || action == "insert rhombus") {
    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<Rhombus>(std::cin);
    tree.Insert(parent_id, id, fig);
} else if (action == "ip" || action == "insert pentagon") {
    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<Pentagon>(std::cin);
    tree.Insert(parent_id, id, fig);
} else if (action == "r" || action == "Remove") {
    std::string id;
    if (!(std::cin >> id)) {
        std::cout << "Incorrect value." << std::endl;
        continue;
    }
    tree.Remove(id);
```

```
        } else if (action == "d" || action == "destroy") {
            //delete tree;
            //tree = new TNTree();
            //std::cout << "The tree was deleted." << std::endl;
        } else if (action == "p" || action == "print") {
            if (!tree.IsEmpty()) {
            tree.PrintTree();
            }
            else {
                std::cout << "The tree is empty." << std::endl;
            }
        } else if (action == "pi" || action == "printitem") {
            std::string id;
            std::cout << "Enter item id: ";
            std::cin >> id;

            tree.PrintItem(id);
        } else if (action == "help" || action == "h") {
            std::cout << "'h'  or 'help'      - display the help." <<
std::endl;
            std::cout << "'r'  or 'Remove'    - Remove the trapezium."
<< std::endl;
            std::cout << "'d'  or 'destroy'   - delete the tree." <<
std::endl;
            std::cout << "'p'  or 'print'     - output the tree." <<
std::endl;
            std::cout << "'it'                - insert a trapezium
into the tree." << std::endl;
            std::cout << "'ir'                - insert a rhombus into
the tree." << std::endl;
            std::cout << "'ip'                - insert a pentagon into
the tree." << std::endl;
            std::cout << "'pi' or 'printitem' - print item by id." <<
std::endl;
            std::cout << "'q'  or 'quit'      - exit the program." <<
std::endl;
        }
    }

    return 0;
}
```

TNTree.h

```
#ifndef TNTREE_H
#define TNTREE_H
```

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

#include "TNTreeItem.h"
#include "Trapezium.h"
#include "Rhombus.h"
#include "Pentagon.h"

class TNTree {
public:
    TNTree();
    TNTree(const TNTree& orig);
    friend std::ostream& operator<<(std::ostream& os, const TNTree&
tree);

    void Insert(std::string parent_id, std::string id,
std::shared_ptr<Figure> &figure);
    void Remove(std::string id);
    void PrintTree();
    void PrintItem(std::string id);
    void SetNull(std::shared_ptr<TNTreeItem> treeItem);
    std::shared_ptr<TNTreeItem> getHead();
    bool IsEmpty() const;

    virtual ~TNTree();
private:
    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 <iostream>

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

TNTree::TNTree()
{
```

```
    head = nullptr;
}

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

void TNTree::Insert(std::string parent_id, std::string id,
std::shared_ptr<Figure> &figure)
{
    if (!this->head) {
        this->head = std::make_shared<TNTreeItem>(id, figure);
        return;
    }
    else {
        std::shared_ptr<TNTreeItem> parent_node = FindNode(this->head,
parent_id);
        if (parent_node) {
            if (!parent_node->GetSon()) {
                parent_node->SetSon(std::make_shared<TNTreeItem>(id,
figure));
            }
            else {
                std::shared_ptr<TNTreeItem> brother = parent_node-
>GetSon();
                while (brother->GetBrother()) {
                    brother = brother->GetBrother();
                }
                brother->SetBrother(std::make_shared<TNTreeItem>(id,
figure));
            }
        }
        else {
            std::cout << "Error: no such parent_id." << std::endl;
        }
    }
}

std::shared_ptr<TNTreeItem>
TNTree::FindNode(std::shared_ptr<TNTreeItem> treeItem, std::string id)
{
    if (treeItem->getId() == id) {
        return treeItem;
    }
```

```
    if (treeItem->GetSon()) {
        std::shared_ptr<TNTreeItem> tr = FindNode(treeItem->GetSon(),
id);
        if (tr != nullptr) {
            return tr;
        }
    }
    if (treeItem->GetBrother()) {
        std::shared_ptr<TNTreeItem> tr = FindNode(treeItem-
>GetBrother(), id);
        if (tr != nullptr) {
            return tr;
        }
    }
    return nullptr;
}

bool TNTree::IsEmpty() const
{
    return head == nullptr;
}

void TNTree::node_remove(std::shared_ptr<TNTreeItem> treeItem,
std::string id)
{
    if (treeItem->GetSon()) {
        if (treeItem->GetSon()->getId() == id) {
            std::shared_ptr<TNTreeItem> tr = treeItem->GetSon();
            treeItem->SetSon(treeItem->GetSon()->GetBrother());
            SetNull(tr->GetBrother());
            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());
            return;
        }
        else {
```

```
            node_remove(treeItem->GetBrother(), id);
        }
    }
}

void TNTree::SetNull(std::shared_ptr<TNTreeItem> treeItem)
{
    treeItem = nullptr;
}

std::shared_ptr<TNTreeItem> TNTree::getHead()
{
    return this->head;
}

void TNTree::Remove(std::string id)
{
    if (head->getId() == id) {
        head = head->GetSon();
    } else {
        TNTree::node_remove(head, id);
   }
}

void TNTree::PrintTree()
{
    if (this->head != nullptr) {
        print_nodes(this->head, 0);
    }
}

void TNTree::print_nodes(std::shared_ptr<TNTreeItem> treeItem, size_t
num) {
    if (treeItem) {
        for (int i = 0; i < num; ++i) {
            printf("\t");
        }
        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)
{
    std::shared_ptr<TNTreeItem> tmp = FindNode(this->head, id);
    if(tmp) {
        std::cout << tmp->getId();
        tmp->GetFigure()->Print();
    }
}

TNTree::~TNTree() {
}
```

TNTreeItem.h

```
#ifndef TNTREEITEM_H
#define TNTREEITEM_H

#include <string>
#include <memory>

#include "Trapezium.h"
#include "Rhombus.h"
#include "Pentagon.h"

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

    void SetSon(std::shared_ptr<TNTreeItem> son);
    void SetBrother(std::shared_ptr<TNTreeItem> brother);
    std::shared_ptr<TNTreeItem> GetSon();
    std::shared_ptr<TNTreeItem> GetBrother();
    std::shared_ptr<Figure> GetFigure() const;
    std::string getId();

    virtual ~TNTreeItem();
private:
    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"

TNTreeItem::TNTreeItem() {
    this->brother = nullptr;
    this->son = nullptr;
    this->id = "";
}

TNTreeItem::TNTreeItem(std::string id, const std::shared_ptr<Figure>
&figure) {
    this->figure = figure;
    this->brother = nullptr;
    this->son = nullptr;
    this->id = id;
}

void TNTreeItem::SetSon(std::shared_ptr<TNTreeItem> son) {
    this->son = son;
}

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;
}

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

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

```
}

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

TNTreeItem::~TNTreeItem() {
    //delete brother;
    //delete son;
}
```

Trapezium.h

```
#ifndef TRAPEZIUM_H
#define TRAPEZIUM_H

#include <iostream>

#include "Figure.h"

class Trapezium : public Figure {
public:
    Trapezium();
    Trapezium(std::istream &is);
    Trapezium(const Trapezium& orig);
    Trapezium(size_t sm_base, size_t lg_base, size_t lf_side, size_t
rt_side);

    friend bool operator==(const Trapezium& fst, const Trapezium&
snd);
    friend std::ostream& operator<<(std::ostream& os, const Trapezium&
obj);
    friend std::istream& operator>>(std::istream &is, Trapezium &obj);
    Trapezium& operator=(const Trapezium &obj);

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

    virtual ~Trapezium() {};
private:
    size_t sm_base;
    size_t lg_base;
    size_t lf_side;
    size_t rt_side;
};
```

```
#endif /*TRAPEZIUM_H */
```

Trapezium.cpp

```
#include <iostream>
#include <cmath>

#include "Trapezium.h"

Trapezium::Trapezium() : Trapezium(0, 0, 0, 0) {
}

Trapezium::Trapezium(size_t sb, size_t lb, size_t ls, size_t rs) :
sm_base(sb), lg_base(lb),

                  lf_side(ls), rt_side(rs) {
}

Trapezium::Trapezium(const Trapezium& orig) {
    sm_base = orig.sm_base;
    lg_base = orig.lg_base;
    lf_side = orig.lf_side;
    rt_side = orig.rt_side;
}

Trapezium::Trapezium(std::istream &is) {
    std::cout << "Enter the sides of the Trapezium:\n";
    is >> sm_base >> lg_base >> lf_side >> rt_side;

    if(sm_base < 0 || lg_base < 0 || lf_side < 0 || rt_side < 0) {
        std::cerr << "Error: sides should be > 0." << std::endl;
    }
}

bool operator==(const Trapezium& fst, const Trapezium& snd) {
    if (fst.lf_side == snd.lf_side &&
        fst.lg_base == snd.lg_base &&
        fst.rt_side == snd.rt_side &&
        fst.sm_base == snd.sm_base) {
            return true;
    } else {
        return false;
    }
}

std::ostream& operator<<(std::ostream& os, const Trapezium& obj) {
```

```
    os << "a=" << obj.sm_base << ", b=" << obj.lg_base << ", c=" <<
obj.rt_side << ", d=" << obj.lf_side << std::endl;
    return os;
}

double Trapezium::Square() {
    if (lg_base > sm_base) {
    return (double)(((sm_base + lg_base) / 2) * sqrt(lf_side * lf_side
- 1 / 4 *
            pow((lf_side * lf_side - rt_side * rt_side) / (lg_base -
sm_base) + lg_base - sm_base, 2)));
    } else {
        std::cerr << "Error: bigger base < smaller base" << std::endl;
    }
    return 0.0;
}
void Trapezium::Print() {
    std::cout << "\nSmaller base = " << sm_base << ", larger base = "
<< lg_base << ", left side = " <<  lf_side << ", right side = " <<
rt_side << std::endl;
}
```

Rhombus.h

```
#ifndef RHOMBUS_H
#define RHOMBUS_H

#include <iostream>

#include "Figure.h"

class Rhombus : public Figure{
public:
    Rhombus();
    Rhombus(std::istream &is);
    Rhombus(int side, int angle);
    Rhombus(const Rhombus& orig);

    friend bool operator==(const Rhombus& fst, const Rhombus& snd);
    friend std::ostream& operator<<(std::ostream& os, const Rhombus&
obj);
    friend std::istream& operator>>(std::istream& is, Rhombus& obj);
    Rhombus& operator=(const Rhombus& obj);

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

```
    virtual ~Rhombus();
private:
    int side;
    int angle;
};

#endif /*RHOMBUS_H */
```

Rhombus.cpp

```
#include <iostream>
#include <cmath>

#include "Rhombus.h"

#define PI 3.14159265

Rhombus::Rhombus() : Rhombus(0, 0) {}

Rhombus::Rhombus(int a, int ang) : side(a), angle(ang) {}

Rhombus::Rhombus(std::istream &is) {
    std::cout << "Enter a side and the smaller angle of the
rhombus:\n";
    is >> side >> angle;
    if(side < 0) {
        std::cerr << "Error: side should be greater than 0." <<
std::endl;
    }
    if(angle < 0 || angle > 180) {
        std::cerr << "Error: angle should be greater than 0 and
smaller than 180." << std::endl;
    }
}

Rhombus::Rhombus(const Rhombus& orig) {
    side = orig.side;
    angle = orig.angle;
}

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

void Rhombus::Print() {
```

```
    std::cout << "Side = " << side << ", angle = " << angle <<
std::endl;
}

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

Pentagon.h

```
#ifndef PENTAGON_H
#define PENTAGON_H

#include <iostream>

#include "Figure.h"

class Pentagon : public Figure{
public:
    Pentagon();
    Pentagon(std::istream &is);
    Pentagon(int side);
    Pentagon(const Pentagon& orig);

    friend bool operator==(const Pentagon& fst, const Pentagon& snd);
    friend std::ostream& operator<<(std::ostream& os, const Pentagon&
obj);
    friend std::istream& operator>>(std::istream& is, Pentagon& obj);
    Pentagon& operator=(const Pentagon& obj);

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

    virtual ~Pentagon();
private:
    int side;
};

#endif /*PENTAGON_H */
```

Pentagon.cpp

```
#include <iostream>
#include <cmath>

#include "Pentagon.h"

#define PI 3.14159265
```

```
Pentagon::Pentagon() : Pentagon(0) {}

Pentagon::Pentagon(int a) : side(a) {}

Pentagon::Pentagon(std::istream &is) {
    std::cout << "Enter a side of the Pentagon:\n";
    is >> side;
    if(side < 0) {
        std::cerr << "Error: sides should be greater than 0." <<
std::endl;
    }
}

Pentagon::Pentagon(const Pentagon& orig) {
    side = orig.side;
}

double Pentagon::Square() {
    return (double)(5 * side * side / (4 * (double)tan(36 * (PI /
180))));
}

void Pentagon::Print() {
    std::cout << "Sides = " << side << std::endl;
}

Pentagon::~Pentagon() {}
```

3.5. Тестирование

```
➜  lab_3 ./run
Type 'h' or 'help' to get help.
> h
'h'  or 'help'      - display the help.
'r'  or 'Remove'    - Remove the trapezium with area s.
'd'  or 'destroy'   - delete the tree.
'p'  or 'print'     - output the tree.
'it'                - insert a trapezium into the tree.
'ir'                - insert a rhombus into the tree.
'ip'                - insert a pentagon into the tree.
'pi' or 'printitem' - print item by id.
'q'  or 'quit'      - exit the program.
> it
Enter parent id: 0
Enter node id: 1
```

```
Enter the sides of the Trapezium:
1 2 3 4
> ir
Enter parent id: 1
Enter node id: 2
Enter a side and the smaller angle of the rhombus:
12 30
> ip
Enter parent id: 1
Enter node id: 3
Enter a side of the Pentagon:
2
> p
1
     2
     3
> pi 1
Enter item id: 1
Smaller base = 1, larger base = 2, left side = 3, right side = 4
> pi 2
Enter item id: 2Side = 12, angle = 30
> q
```

3.6. Выводы

Появившиеся в стандарте C++11 умные указатели, являются весьма важным инструментом разработчика. Такой указатель помогает избежать множества проблем: «висячие» указатели, «утечки» памяти и отказы в выделении памяти, что весьма упрощает разработку и отладку программ.

4. ЛАБОРАТОРНАЯ РАБОТА №4

4.1. Цель работы

Целью лабораторной работы является:

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

4.2. Задание

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

- Требования к классам фигуры аналогичны требованиям из лабораторной работы 1.
- Шаблон класса-контейнера должен соджержать 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 (опередеяется структурой контейнера).
- Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера
(опередеяется 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 расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

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

4.3. Описание

Шаблоны— средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).
В C++ возможно создание шаблонов функций и классов.
Шаблоны позволяют создавать параметризованные классы и функции.
Параметром может быть любой тип или значение одного из допустимых типов (целое число, enum, указатель на любой объект с глобально доступным именем, ссылка).
Хотя шаблоны предоставляют краткую форму записи участка кода, на самом деле их использование не сокращает исполняемый код, так как для каждого набора параметров компилятор создаёт отдельный экземпляр функции или класса. Как следствие, исчезает возможность совместного использования скомпилированного кода в рамках разделяемых библиотек.

4.4. Исходный код

TNTree.h

```
#ifndef TNTREE_H
#define TNTREE_H

#include <string>

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

template <class T>
class TNTree {
public:
    TNTree();
    TNTree(const TNTree<T>& orig);
    template <class A> friend std::ostream& operator<<(std::ostream&
os, const TNTree<A>& tree);

    void Insert(std::string parent_id, std::string id, const
std::shared_ptr<T> &figure);
    void Remove(std::string id);
    void PrintTree() const;
    void PrintItem(std::string id);
    void SetNull(std::shared_ptr<TNTreeItem<T>> treeItem);
    std::shared_ptr<TNTreeItem<T>> getHead();
    bool IsEmpty() const;
```

```
    virtual ~TNTree() {};
private:
    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 <iostream>

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

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

template <class T>
TNTree<T>::TNTree()
{
    head = nullptr;
}

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

template <class T>
std::ostream& operator<<(std::ostream& os, const TNTree<T>& tree) {
    tree.PrintTree();
    os << "";
    return os;
}

template <class T>
```

```
void TNTree<T>::Insert(std::string parent_id, std::string id, const
std::shared_ptr<T> &figure)
{
    if (!this->head) {
        this->head = std::make_shared<TNTreeItem<T>>(id, figure);
        return;
    }
    else {
        std::shared_ptr<TNTreeItem<T>> parent_node = FindNode(this-
>head, parent_id);
        if (parent_node) {
            if (!parent_node->GetSon()) {
                parent_node-
>SetSon(std::make_shared<TNTreeItem<T>>(id, figure));
            }
            else {
                std::shared_ptr<TNTreeItem<T>> brother = parent_node-
>GetSon();
                while (brother->GetBrother()) {
                    brother = brother->GetBrother();
                }
                brother-
>SetBrother(std::make_shared<TNTreeItem<T>>(id, figure));
            }
        }
        else {
            std::cout << "Error: no such parent_id." << std::endl;
        }
    }
}

template <class T>
std::shared_ptr<TNTreeItem<T>>
TNTree<T>::FindNode(std::shared_ptr<TNTreeItem<T>> treeItem,
std::string id)
{
    if (treeItem->getId() == id) {
        return treeItem;
    }
    if (treeItem->GetSon()) {
        std::shared_ptr<TNTreeItem<T>> tr = FindNode(treeItem-
>GetSon(), id);
        if (tr != nullptr) {
            return tr;
        }
```

```
    }
    if (treeItem->GetBrother()) {
        std::shared_ptr<TNTreeItem<T>> tr = FindNode(treeItem-
>GetBrother(), id);
        if (tr != nullptr) {
            return tr;
        }
    }
    return nullptr;
}

template <class T>
bool TNTree<T>::IsEmpty() const
{
    return head == nullptr;
}

template <class T>
void TNTree<T>::node_remove(std::shared_ptr<TNTreeItem<T>> treeItem,
std::string id)
{
    if (treeItem->GetSon()) {
        if (treeItem->GetSon()->getId() == id) {
            std::shared_ptr<TNTreeItem<T>> tr = treeItem->GetSon();
            treeItem->SetSon(treeItem->GetSon()->GetBrother());
            SetNull(tr->GetBrother());
            return;
        }
        else {
            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());
            return;
        }
        else {
            node_remove(treeItem->GetBrother(), id);
        }
    }
```

```
}

template <class T>
void TNTree<T>::SetNull(std::shared_ptr<TNTreeItem<T>> treeItem)
{
    treeItem = nullptr;
}

template <class T>
std::shared_ptr<TNTreeItem<T>> TNTree<T>::getHead()
{
    return this->head;
}

template <class T>
void TNTree<T>::Remove(std::string id)
{
    if (head->getId() == id) {
        head = head->GetSon();
    } else {
        TNTree<T>::node_remove(head, id);
   }
}

template <class T>
void TNTree<T>::PrintTree() const
{
    if (this->head != nullptr) {
        print_nodes(this->head, 0);
    }
}

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("\t");
        }
        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)
{
    std::shared_ptr<TNTreeItem<T>> tmp = FindNode(this->head, id);
    if(tmp) {
        std::cout << tmp->getId();
        tmp->GetFigure()->Print();
    }
}
```

TNTreeItem.h

```
#ifndef TNTREEITEM_H
#define TNTREEITEM_H

#include <string>
#include <memory>

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

    void SetSon(std::shared_ptr<TNTreeItem<T>> son);
    void SetBrother(std::shared_ptr<TNTreeItem<T>> brother);
    std::shared_ptr<TNTreeItem<T>> GetSon();
    std::shared_ptr<TNTreeItem<T>> GetBrother();
    std::shared_ptr<T> GetFigure() const;
    std::string getId();

    virtual ~TNTreeItem() {};
private:
    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 <string>
#include <iostream>

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

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

template <class T>
TNTreeItem<T>::TNTreeItem() {
    this->brother = nullptr;
    this->son = nullptr;
    this->id = "";
}

template <class T>
TNTreeItem<T>::TNTreeItem(std::string id, const std::shared_ptr<T>
&figure) {
    this->id = id;
    this->figure = figure;

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

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

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;
}

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

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;
}
```

4.5. Тестирование

```
➜  lab_4 ./run
Type 'h' or 'help' to get help.
> h
'h'  or 'help'      - display the help.
'r'  or 'Remove'    - Remove the trapezium with area s.
'd'  or 'destroy'   - delete the tree.
'p'  or 'print'     - output the tree.
'it'                - insert a trapezium into the tree.
'ir'                - insert a rhombus into the tree.
'ip'                - insert a pentagon into the tree.
'pi' or 'printitem' - print item by id.
'q'  or 'quit'      - exit the program.
> it
Enter parent id: 0
Enter node id: 0
Enter the sides of the Trapezium:
1 2 3 4
> ip
Enter parent id: 0
Enter node id: 1
Enter a side of the Pentagon:
12
> p
0
```

```
     1
> pi 1
Enter item id: 1Sides = 12
> q
```

4.6. Выводы

Шаблоны являются одним из мощнейших и самых выразительных средств языка программирования C++. Параметризация классов и функций позволяет переиспользовать уже готовый и оттестированный код, что значительно ускоряет время разработки, один из важнейших параметров. Шаблоны языка C++ являются Тьюринг-полным языком, что позволяет писать поистине поразительные вещи, например игру "Жизнь", которая выполняется на этапе компиляции. Впрочем, к сожалению, есть и некоторые недостатки: проблемы портируемости, отсутствие поддержки отладки или ввода/вывода в процессе инстанцирования шаблонов, длительное время компиляции.

5. ЛАБОРАТОРНАЯ РАБОТА №5

5.1. Цель работы

Целью лабораторной работы является:

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

5.2. Задание

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛРNo4) спроектировать и разработать Итератор для динамической структуры данных.

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

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

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

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

5.3. Описание

Итератор— интерфейс, предоставляющий доступ к элементам коллекции (массива или контейнера) и навигацию по ним. В различных системах итераторы могут иметь разные общепринятые названия. В терминах систем управления базами данных итераторы называются курсорами. В простейшем случае итератором в низкоуровневых языках является указатель. Использование итераторов в обобщённом программировании позволяет реализовать универсальные алгоритмы работы с контейнерами.
Главное предназначение итераторов заключается в предоставлении возможности пользователю обращаться к любому элементу контейнера при сокрытии внутренней структуры контейнера от пользователя. Это позволяет контейнеру хранить элементы любым способом при допустимости работы

пользователя с ним как с простой последовательностью или списком. Проектирование класса итератора обычно тесно связано с соответствующим классом контейнера. Обычно контейнер предоставляет методы создания итераторов.
Итератор похож на указатель своими основными операциями: он указывает на отдельный элемент коллекции объектов (предоставляет доступ к элементу) и содержит функции для перехода к другому элементу списка (следующему или предыдущему). Контейнер, который реализует поддержку итераторов, должен предоставлять первый элемент списка, а также возможность проверить, перебраны ли все элементы контейнера (является ли итератор конечным). В зависимости от используемого языка и цели, итераторы могут поддерживать дополнительные операции или определять различные варианты поведения.

5.4. Исходный код

TIterator.h

```
#ifndef TITERATOR_H
#define TITERATOR_H

#include <memory>
#include <iostream>

template <class N, class T>
class TIterator
{
public:
    TIterator(std::shared_ptr<N> n) {
        cur = n;
    }

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

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

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

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

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

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

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

#endif
```

5.5. Тестирование

```
➜  lab_5 ./run
Type 'h' or 'help' to get help.
> h
'h'  or 'help'      - display the help.
'r'  or 'Remove'    - Remove the trapezium with area s.
'd'  or 'destroy'   - delete the tree.
'p'  or 'print'     - output the tree.
'it'                - insert a trapezium into the tree.
'ir'                - insert a rhombus into the tree.
'ip'                - insert a pentagon into the tree.
'pi' or 'printitem' - print item by id.
'q'  or 'quit'      - exit the program.
> it
Enter parent id: 0
Enter node id: 0
Enter the sides of the Trapezium:
1 2 3 4
> ip
Enter parent id: 0
Enter node id: 1
Enter a side of the Pentagon:
12
```

```
> p
0
     1
> pi 0
Enter item id: Smaller base = 1, larger base = 2, left side = 3, right
side = 4
> q
```

5.6. Выводы

В данной работе необходимо было создать итератор для n-дерева. Итераторы для списков, очередей, стеков являются хорошо известными и понятными объектами, но не для n-дерева. Одним из наиболее понятных способов является использование итератора для перечисления братьев каждой вершины. Так же возможно линеаризовать дерево и записать его в список, очередь или очередь.

6. ЛАБОРАТОРНАЯ РАБОТА №6

6.1. Цель работы

Целью лабораторной работы является:

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

6.2. Задание

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

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

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

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

6.3. Описание

При помощи аллокатора происходит выделение и освобождение памяти в требуемых количествах определённым образом. std::allocator -- пример реализации аллокатора из стандартной библиотеки, просто использует new и delete, которые обычно обращаются к системным вызовам malloc и free. Но в чем-то задание лабораторной работы похоже на паттерн проектирования "Object Pool" - набор инициализированных и готовых к использованию объектов. Когда системе требуется объект, он не создаётся, а берётся из пула. Когда объект больше не нужен, он не уничтожается, а возвращается в пул.

6.4. Исходный код

TAllocationBlock.h

```
#ifndef TALLOCATIONBLOCK_H
#define TALLOCATIONBLOCK_H

#include <iostream>
#include <cstdlib>

#include "TList.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;
    TList<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.Insert(ptr);
    }
}

void *TAllocationBlock::Allocate()
{
```

```
    if(!_free_blocks.IsEmpty()) {
        void *res = _free_blocks.First();
        _free_blocks.DeleteLast();
        return res;
    } else {
        throw std::bad_alloc();
    }
}

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

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

int32_t TAllocationBlock::Size()
{
    return _free_blocks.GetSize();
}

TAllocationBlock::~TAllocationBlock()
{
    while(!_free_blocks.IsEmpty()) {
        _free_blocks.DeleteLast();
    }
    free(_used_blocks);
}
```

6.5. Тестирование

```
➜  lab_6 ./run
Type 'h' or 'help' to get help.
> h
'h'  or 'help'      - display the help.
'r'  or 'Remove'    - Remove the trapezium.
'd'  or 'destroy'   - delete the tree.
'p'  or 'print'     - output the tree.
'it'                - insert a trapezium into the tree.
'ir'                - insert a rhombus into the tree.
'ip'                - insert a pentagon into the tree.
'pi' or 'printitem' - print item by id.
```

```
'q'  or 'quit'      - exit the program.
> it
Enter parent id: 0
Enter node id: 0
Enter the sides of the Trapezium:
1 2 3 4
> ip
Enter parent id: 0
Enter node id: 1
Enter a side of the Pentagon:
12
> p
0
     1
> pi 1
Enter item id: 1
Sides = 12
> q
```

6.6. Выводы

В некоторых случаях кастомные аллокаторы памяти бывают весьма полезны. Для ускорения программы бывает полезно сначала создать некоторое количество объектов заранее, а затем использовать их. Несомненно, память — это драгоценный ресурс, которым нужно тщательно управлять.

7. ЛАБОРАТОРНАЯ РАБОТА №7

7.1. Цель работы

Целью лабораторной работы является:

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

7.2. Задание

Необходимо реализовать динамическую структуру данных – «Хранилище объектов» и алгоритм работы с ней. «Хранилище объектов» представляет собой контейнер, одного из следующих видов (Контейнер 1-го уровня):

1. Массив
2. Связанный список
3. Бинарное- Дерево.
4. N-Дерево (с ограничением не больше 4 элементов на одном уровне).
5. Очередь
6. Стек

Каждым элементом контейнера, в свою, является динамической структурой данных одного из следующих видов (Контейнер 2-го уровня):

1. Массив
2. Связанный список
3. Бинарное- Дерево
4. N-Дерево (с ограничением не больше 4 элементов на одном уровне).
5. Очередь
6. Стек

Таким образом у нас получается контейнер в контейнере. Т.е. для варианта (1,2) это будет массив, каждый из элементов которого – связанный список. А для варианта (5,3) – это очередь из бинарных деревьев.
Элементом второго контейнера является объект-фигура, определенная вариантом задания.

При этом должно выполняться правило, что количество объектов в контейнере второго уровня не больше 5.
Т.е. если нужно хранить больше 5 объектов, то создается еще один контейнер второго уровня. Например, для варианта (1,2) добавление объектов будет выглядеть следующим образом:

1. Вначале массив пустой.
2. Добавляем Объект 1: В массиве по индексу 0 создается элемент с типом список, в список добавляется Объект 1.

3. Добавляем Объект 2: Объект добавляется в список, находящийся в массиве по индекс 0.

4. Добавляем Объект 3: Объект добавляется в список, находящийся в массиве по индекс 0.

5. Добавляем Объект 4: Объект добавляется в список, находящийся в массиве по индекс 0.

6. Добавляем Объект 5: Объект добавляется в список, находящийся в массиве по индекс 0.

7. Добавляем Объект 6: В массиве по индексу 1 создается элемент с типом список, в список добавляется Объект 6.

Объекты в контейнерах второго уровня должны быть отсортированы по возрастанию площади объекта (в том числе и для деревьев).

При удалении объектов должно выполняться правило, что контейнер второго уровня не должен быть пустым. Т.е. если он становится пустым, то он должен удалится.

Нельзя использовать:

- Стандартные контейнеры std.

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).

7.3. Описание

Контейнером называют объект, который содержит другие объекты.

Например в stl существуют контейнеры-последовательности (такие как vector, array, deque, list), ассоциативный контейнеры (map, set), контейнеры-адаптеры (queue, stack).

7.4. Исходный код

TList.h

```
#ifndef TLIST_H
#define TLIST_H

#include <functional>
```

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

#include "TAllocator.h"

template <typename T> class TList {
 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 TAllocator nodeAllocator;
  };

  template <typename N, typename 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;
  auto psort(std::shared_ptr<TNode>&);
  auto pparsort(std::shared_ptr<TNode>& head);
  auto partition(std::shared_ptr<TNode>&);

 public:
  TList();
```

```
  bool PushFront(const std::shared_ptr<T>&);
  bool Push(const std::shared_ptr<T>&, const int);
  bool PopFront();
  bool Pop(const int);
  bool IsEmpty() const;
  int GetLength() const;
  auto& getHead();
  auto&& getTail();
  void sort();
  void parSort();

  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);};

  template <typename A> friend std::ostream& operator<<
(std::ostream&, TList<A>&);
};

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

TList.cpp

```
#ifdef TLIST_H

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

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

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

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

```
template <typename Q, typename O> void TTree<Q,
O>::recRemByType(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) {
      recRemByType(node->sibling, type);
    }
    if (node->son) {
      recRemByType(node->son, type);
    }

  }
}

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

template <typename Q, typename O> void TTree<Q,
O>::recInsert(std::shared_ptr<Node>& node, const O& item) {
  if (node->itemsInNode < 5) {
    node->data.PushFront(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));
        return;
      }
```

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

      sibl = sibl->sibling;
    }

    if (node->son) {
      recInsert(node->son, item);
    } else {
      node->son = std::make_shared<Node>(Node(item));
    }
  }
}

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

template <typename Q, typename O> void TTree<Q, O>::recInorder(const
std::shared_ptr<Node>& node) {
  if (node->itemsInNode) {
    node->data.sort();
    for (const auto& i: node->data) {
      i->Print();
    }
    std::cout << "\n";

    if (node->sibling) {
      recInorder(node->sibling);
    }
    if (node->son) {
      recInorder(node->son);
    }
  }
}

template <typename Q, typename O> void TTree<Q, O>::inorder() {
  if (root->son->son || root->son->sibling) {
    clear(root->son, root);
  }
  recInorder(root->son);
```

```
}

template <typename Q, typename O> void TTree<Q,
O>::recRemLesser(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->getSquare() < sqr) {
          node->data.Pop(k + 1);
          node->itemsInNode--;
          break;
        }
        ++iter;
      }
    }

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

  }
}

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

template <typename Q, typename O> void TTree<Q,
O>::clear(std::shared_ptr<Node>& node, std::shared_ptr<Node>& parent)
{
  if (node) {
    if (!node->itemsInNode) {
      auto orphan = node;
      auto orphanPar = parent;
      if (node->sibling) {
        orphan = node->sibling;
        orphanPar = node;
      } else if (node->son) {
        orphan = node->sibling;
```

```
        orphanPar = node;
      }

      while (orphan->sibling || orphan->son) {
        orphanPar = orphan;
        if (orphan->sibling) {
          orphan = orphan->sibling;
        } else if (orphan->son) {
          orphan = orphan->son;
        }
      }

      if (orphanPar->sibling == orphan) {
        std::swap(node->data, orphan->data);
        node->itemsInNode = orphan->itemsInNode;
        orphanPar->sibling = nullptr;
      } else if (orphanPar->son == orphan) {
        std::swap(node->data, orphan->data);
        node->itemsInNode = orphan->itemsInNode;
        orphanPar->son = nullptr;
      }

    }
  }

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

#endif
```

7.5. Тестирование

```
➜ lab_7 ./run
Type 'h' or 'help' to get help.
> h
'h'  or 'help'      - display the help.
'r'  or 'remove'    - remove the figure.
```

```
'p'  or 'print'     - output the tree.
'it'                - insert a trapezium into the tree.
'ir'                - insert a rhombus into the tree.
'ip'                - insert a pentagon into the tree.
'q'  or 'quit'      - exit the program.
> it
Enter the sides of the Trapezium:
1 2 3 4
> ip
Enter side: 1
> ip
Enter side: 3
> p
a = 1, b = 2, c = 3, d = 4, Square = 1.5
Sides = 1, Square = 1.72048
Sides = 2, Square = 6.88191
Sides = 3, Square = 15.4843
> ip
Enter side: 56
> ip
Enter side: 34
> p
a = 1, b = 2, c = 3, d = 4, Square = 1.5
Sides = 1, Square = 1.72048
Sides = 2, Square = 6.88191
Sides = 3, Square = 15.4843
Sides = 56, Square = 5395.42

Sides = 45, Square = 3483.97
>q
```

7.6. Выводы

В данной лабораторной работе были рассмотрены принципы работы с контейнерами. Также был рассмотрен важный принцип ООП, сформулированный Бертраном Мейером, OCP (The Open Closed Principle), устанавливающий следующее положение: «программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения».

8. ЛАБОРАТОРНАЯ РАБОТА №8

8.1. Цель работы

Целью лабораторной работы является:

- Знакомство с параллельным программированием в C++.

8.2. 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

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

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

8.3. Описание

Параллельное программирование - это техника программирования, которая использует преимущества многоядерных или многопроцессорных компьютеров. Оно включает в себя все черты более традиционного, последовательного программирования, но в параллельном программировании имеются три дополнительных, четко определенных этапа.

Определение параллелизма: анализ задачи с целью выделить подзадачи, которые могут выполняться одновременно.

Выявление параллелизма: изменение структуры задачи таким образом, чтобы можно было эффективно выполнять подзадачи. Для этого часто требуется найти зависимости между подзадачами и организовать исходный код так, чтобы ими можно было эффективно управлять

Выражение параллелизма: реализация параллельного алгоритма в исходном коде с помощью системы обозначений параллельного программирования.

В данной работе я использовал std::thread. Класс thread предоставляет один поток выполнения. Потоки позволяют нескольким фрагментам кода работать асинхронно и одновременно.

8.4. Исходный код

main.cpp

```
#include <iostream>

#include "TList.h"

#include "Trapezium.h"
#include "Rhombus.h"
#include "Pentagon.h"

void menu(void) {
    std::cout << "it    insert trapezium" << std::endl;
    std::cout << "ir    insert rhombus" << std::endl;
    std::cout << "ip    insert pentagon" << std::endl;
    std::cout << "p     print" << std::endl;
    std::cout << "s     sort" << std::endl;
    std::cout << "d     delete item" << std::endl;
    std::cout << "q     quit" << std::endl;
}

int main(void) {
    TList<Figure> list;
    int index;
    std::string action;

    menu();
    while (1) {
        std::cin.clear();
        std::cin.sync();
```

```
        std::cout << "> ";
        std::cin >> action;

        if (action == "q" || action == "quit") {
            break;
        } else if (action == "it") {
            Trapezium fig;

            std::cin >> fig >> index;
            list.Push(std::make_shared<Trapezium>(fig), index);
        } else if (action == "ir") {
            Rhombus fig;

            std::cin >> fig >> index;
            list.Push(std::make_shared<Rhombus>(fig), index);
        } else if (action == "ip") {
            Pentagon fig;

            std::cin >> fig >> index;
            list.Push(std::make_shared<Pentagon>(fig), index);
        } else if (action == "s") {
            std::cout << "1 regular sort\n2 parallel sort\n";
            std::cin >> index;
            if (index == 1) {
                list.sort();
            } else if (index == 2) {
                list.parallelSort();
            }
        } else if (action == "p") {
            for (const auto& i : list) {
                i->Print();
            }
        } else if (action == "d") {
            std::cout << "Enter index of item:" << std::endl;
            std::cin >> index;
            if (list.Pop(index))
                std::cout << "Item was removed" << std::endl;
            else
                std::cout << "Item was not removed" << std::endl;
        }
    }

    return 0;
}
```

TList.h

```
#ifndef TLIST_H
#define TLIST_H

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

#include "TAllocator.h"

template <typename T> class TList {
 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 TAllocator nodeAllocator;
  };

  template <typename N, typename 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;
  auto psort(std::shared_ptr<TNode>&);
```

```
  auto pparsort(std::shared_ptr<TNode>& head);
  auto partition(std::shared_ptr<TNode>&);
  std::mutex mutex;

 public:
  TList();
  bool PushFront(const std::shared_ptr<T>&);
  bool Push(const std::shared_ptr<T>&, const int);
  void Pop(const int);
  bool IsEmpty() const;
  int GetLength() const;
  auto& getHead();
  auto&& getTail();
  void sort();
  void parallelSort();

  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);};

  template <typename A> friend std::ostream& operator<<
(std::ostream&, TList<A>&);
};

#include "TList.hpp"
#include "TIterator.hpp"
#endif
```

TList.cpp

```
#ifdef TLIST_H

#include <unistd.h>

template <typename T> TList<T>::TNode::TNode() {
  item = std::shared_ptr<T>();
  next = nullptr;
}

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

```
template <typename T> TAllocator
TList<T>::TNode::nodeAllocator(sizeof(TList<T>::TNode), 100);

template <typename T> void* TList<T>::TNode::operator new(size_t size)
{
  return nodeAllocator.allocate();
}

template <typename T> void TList<T>::TNode::operator delete(void* ptr)
{
  nodeAllocator.deallocate(ptr);
}

template <typename T> TList<T>::TList() {
  head = std::make_shared<TNode>();
  length = 0;
}

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

template <typename T> auto& TList<T>::getHead() {
  return this->head->next;
}

template <typename T> auto&& TList<T>::getTail() {
  auto tail = head->next;
  while (tail->next != nullptr) {
    tail = tail->next;
  }

  return tail;
}

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

template <typename T> bool TList<T>::PushFront(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 <typename T> bool TList<T>::Push(const std::shared_ptr<T>&
obj, int pos) {
  if (pos == 1 || length == 0)
    return PushFront(obj);
  if (pos < 0 || pos > length + 1)
    return false;

  auto iter = head->next;
  int i = 0;

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

  auto Nitem = std::make_shared<TNode>(obj);
  std::swap(Nitem->next, iter->next);
  std::swap(iter->next, Nitem);
  length++;

  return true;
}

template <typename T> bool TList<T>::Pop(int pos) {
  if (pos < 1 || pos > length || IsEmpty())
    return false;
  if (pos == 1) {
    if (!IsEmpty()) {
      head = std::move(head->next);
      length--;
    }
  }

  auto iter = head->next;
  int i = 0;

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

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

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

template <typename T> auto TList<T>::TNode::GetItem() const {
  return this->item;
}

template <typename A> std::ostream& operator<< (std::ostream& os,
const TList<A>& list) {
  if (list.IsEmpty()) {
    os << "The list is empty!" << std::endl;
    return os;
}

  auto tmp = list.head->GetNext();
  while(tmp != nullptr) {
    tmp->GetItem()->Print();
    tmp = tmp->GetNext();
  }

  return os;
}

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

  auto partitionedEl = partition(head);
  auto leftPartition = partitionedEl->next;
  auto rightPartition = head;

  partitionedEl->next = nullptr;

  if (leftPartition == nullptr) {
    leftPartition = head;
    rightPartition = head->next;
    head->next = nullptr;
  }
```

```
  rightPartition = psort(rightPartition);
  leftPartition = psort(leftPartition);
  auto iter = leftPartition;
  while (iter->next != nullptr) {
    iter = iter->next;
  }

  iter->next = rightPartition;

  return leftPartition;
}

template <typename T> auto TList<T>::partition(std::shared_ptr<TNode>&
head) {
  std::lock_guard<std::mutex> lock(mutex);
  if (head->next->next == nullptr) {
    if (head->next->GetItem()->getSquare() > head->GetItem()-
>getSquare()) {
      return head->next;
    } else {
      return head;
    }
  } else {
    auto i = head->next;
    auto pivot = head;
    auto lastElSwapped = (pivot->next->GetItem()->getSquare()
                          >= pivot->GetItem()->getSquare()) ? pivot-
>next : pivot;

    while ((i != nullptr) && (i->next != nullptr)) {
      if (i->next->GetItem()->getSquare() >= pivot->GetItem()-
>getSquare()) {
        if (i->next == lastElSwapped->next) {
          lastElSwapped = lastElSwapped->next;
        } else {
          auto tmp = lastElSwapped->next;
          lastElSwapped->next = i->next;
          i->next = i->next->next;
          lastElSwapped = lastElSwapped->next;
          lastElSwapped->next = tmp;
        }
      }

      i = i->next;
```

```
    }

    return lastElSwapped;
  }

}

template <typename T> void TList<T>::sort() {
  head->next = psort(head->next);
}

template <typename T> void TList<T>::parallelSort() {
  head->next = pparsort(head->next);
}

template <typename T> auto TList<T>::pparsort(std::shared_ptr<TNode>&
head) {
  sleep(2);
  if (head == nullptr || head->next == nullptr) {
    return head;
  }

  auto partitionedEl = partition(head);
  auto leftPartition = partitionedEl->next;
  auto rightPartition = head;

  partitionedEl->next = nullptr;

  if (leftPartition == nullptr) {
    leftPartition = head;
    rightPartition = head->next;
    head->next = nullptr;
  }

  std::packaged_task<std::shared_ptr<TNode>(std::shared_ptr<TNode>&)>
    task1(std::bind(&TList<T>::pparsort, this,
std::placeholders::_1));
  std::packaged_task<std::shared_ptr<TNode>(std::shared_ptr<TNode>&)>
    task2(std::bind(&TList<T>::pparsort, this,
std::placeholders::_1));
  auto rightPartitionHandle = task1.get_future();
  auto leftPartitionHandle = task2.get_future();


  std::thread(std::move(task1), std::ref(rightPartition)).join();
```

```
  rightPartition = rightPartitionHandle.get();
  std::thread(std::move(task2), std::ref(leftPartition)).join();
  leftPartition = leftPartitionHandle.get();
  auto iter = leftPartition;
  while (iter->next != nullptr) {
    iter = iter->next;
  }

  iter->next = rightPartition;

  return leftPartition;
}

#endif
```

8.5. Тестирование

```
➜ lab_8 ./run
it    insert trapezium
ir    insert rhombus
ip    insert pentagon
p     print
s     sort
d     delete item
q     quit
> it
Enter the sides of the Trapezium:
1 2 3 4
1
> ip
Enter side: 3
2
> ip
Enter side: 1
3
> ir
Enter side: 34
Enter smaller angle: 45
4
> p
a = 1, b = 2, c = 3, d = 4, Square = 1.5
Sides = 3, Square = 15.4843
Sides = 1, Square = 1.72048
Side = 34, angle = 45, Square = 817.415
> s
```

```
1 regular sort
2 parallel sort
2
> p
a = 1, b = 2, c = 3, d = 4, Square = 1.5
Sides = 1, Square = 1.72048
Sides = 3, Square = 15.4843
Side = 34, angle = 45, Square = 817.415
> d
Enter index of item:
2
Item was removed
> p
a = 1, b = 2, c = 3, d = 4, Square = 1.5
Sides = 3, Square = 15.4843
Side = 34, angle = 45, Square = 817.415
> q
```

8.6. Выводы

Использование технологий параллельного программирования является стандартом написания современных приложений. К достоинствам многопоточности в программировании можно отнести следующее:

- Упрощение программы в некоторых случаях за счёт использования общего адресного пространства.
- Меньшие относительно процесса временны́е затраты на создание потока.
- Повышение производительности процесса за счёт распараллеливания процессорных вычислений и операций ввода-вывода.

9. ЛАБОРАТОРНАЯ РАБОТА №9

9.1. Цель работы

Целью лабораторной работы является:

- Знакомство с лямбда-выражениями

9.2. Задание

Используя структуры данных, разработанные для лабораторной работы No6 (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:
  - o Генерация фигур со случайным значением параметров;
  - o Печать контейнера на экран;
  - o Удаление элементов со значением площади меньше определенного числа;
- В контейнер второго уровня поместить цепочку команд.
- Реализовать цикл, который проходит по всем командам в контенере второго уровня и выполняет их, применяя к контейнеру первого уровня.

Для создания потоков использовать механизмы:

- future
- packaged_task/async

Для обеспечения потоко-безопасности структур данных использовать:

- mutex
- lock_guard

Нельзя использовать:

- Стандартные контейнеры std.

9.3. Описание

Лямбда-выражения в C++ — это краткая форма записи анонимных функторов. Они позволяют определить анонимный объект-функцию прямо внутри (inline) кода или передать в функцию в качестве аргумента.

9.4. Исходный код

main.cpp

```
#include <iostream>

#include "TTree.h"
#include "TList.h"

#include "Trapezium.h"
#include "Rhombus.h"
#include "Pentagon.h"

int main(void) {
  TList<Figure> list;
  TTree<std::shared_ptr<std::function<void(void)>>> tree(4);
  srand (time(NULL));

  std::function<void(void)> Insert = [&]() {
    std::cout << "Command: Insert" << std::endl;
    for (int i = 0; i < 10; i++) {
      int side  = std::rand() % 10 + 1;
      int angle = std::rand() % 10 + 1;
      if ((side % 2) == 0) {
        list.PushFront(std::make_shared<Trapezium>(Trapezium(side,
side+1, side+2, side+3)));
      } else if((side % 3) == 0) {
        list.PushFront(std::make_shared<Rhombus>(Rhombus(side,
angle)));
      } else {
        list.PushFront(std::make_shared<Pentagon>(Pentagon(side)));
      }
    }
  };

  std::function<void(void)> Print = [&]() {
    std::cout << "Command: Print" << std::endl;
    for (const auto& i : list) {
      i->Print();
    }
  };

  std::function<void(void)> Remove = [&]() {
    std::cout << "Command: Remove" << std::endl;
    double sqr = (double) (std::rand()) / RAND_MAX * 12;
    std::cout << "Lesser than " << sqr << std::endl;
```

```
    for (int i = 0; i < 10; i++) {
      auto iter = list.begin();

      for (int k = 0; k < list.GetLength(); k++) {
        if (iter->getSquare() < sqr) {
          list.Pop(k + 1);
          break;
        }
        ++iter;
      }
    }
  };

  tree.insert(std::shared_ptr<std::function<void(void)>>(&Insert,
[](std::function<void(void)>*){}));
  tree.insert(std::shared_ptr<std::function<void(void)>>(&Print,
[](std::function<void(void)>*){}));
  tree.insert(std::shared_ptr<std::function<void(void)>>(&Remove,
[](std::function<void(void)>*){}));
  tree.insert(std::shared_ptr<std::function<void(void)>>(&Print,
[](std::function<void(void)>*){}));
  tree.inorder();

  return 0;
}
```

9.5. Тестирование

```
➜ lab_9 ./run
Command: Insert
Command: Print
a = 10, b = 11, c = 12, d = 13, Square = 10.5
Side = 3, angle = 7, Square = 1.09682
a = 2, b = 3, c = 4, d = 5, Square = 2.5
Side = 9, angle = 1, Square = 1.41364
a = 10, b = 11, c = 12, d = 13, Square = 10.5
a = 2, b = 3, c = 4, d = 5, Square = 2.5
Sides = 1, Square = 1.72048
Sides = 1, Square = 1.72048
Side = 3, angle = 9, Square = 1.40791
Sides = 5, Square = 43.0119
Command: Remove
Lesser than 10.1798
Command: Print
a = 10, b = 11, c = 12, d = 13, Square = 10.5
```

```
a = 10, b = 11, c = 12, d = 13, Square = 10.5
Sides = 5, Square = 43.0119
```

9.6. Выводы

Лямбда-функции являются относительно новым, но весьма полезным дополнением стандарта C++. Например, они позволяют избежать написания функций/функторов которые нужны только в конкретном месте кода и больше нигде (это частный случай, но наиболее распространенный для применения лямбд).