Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 4

Студент: Попов Илья Павлович
Группа: 80-206
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2021

1. Постановка задачи

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий **одну фигуру** (одна из фигур ЛР3 на выбор). Вариант структуры данных для контейнера выбрать из документа “**Варианты структур данных**” **(контейнер 1-го уровня)** согласно своему номеру из **Варианты ЛР4.**.

- Классы должны удовлетворять следующим правилам:
- · Требования к классу фигуры аналогичны требованиям из лабораторной работы 3
- · 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).

Вариант 18:

18.	Бинарное- Дерево	Стек

2. Описание программы

В качестве класса, которые будет являться Node’ми дерева, я выбрал класс трапеций из 3 лабораторной работы:

class Trapezoid : {...}

Фигуры считываются со стандартного ввода поточечно(пользователь последовательно вводит пары координат точек фигуры), далее происходит проверка введенных значений, и, если фигура считана корректно, она отправляется дерево. Так как хотелось использовать “бинарность” дерева, я перегрузил операторы сравнения для двух трапеций (больше та, чья площадь больше).

Для дерева реализованы следующие методы:
BinTree() : root(NULL) {}; - конструктор
~BinTree() { recursive_delete(root); }; - деструктор

bool empty() {...} - проверка на путоту

void insert(const T& data) { recursive_insert(&root, data); } - добавление элемента

void print() const { recursive_print(root, 0, false); } - печать дерева

void remove(const string& path) {recursive_search(&root, path);} - удаление из дерева

Для взаимодействия с деревом предусмотрены следующие команды:

1. add - добавить фигуру
2. del - удалить фигуру
3. print - распечатывать дерево
4. menu - вывести меню повторно
5. exit - прекратить взаимодействие

3. Набор тестов

Тест №1:

Демонстрирует корректность работы программы

Доступные функции:
add - добавить фигуру
get - получить фигуру
del - удалить фигуру
menu - вывести меню повторно
exit - выход

add
Введите в порядке последовательного обхода фигуры точки в вида x y
0 0
0 1
1 1
7 0
Фигура успешно добавлена в дерево

add
Введите в порядке последовательного обхода фигуры точки в вида x y
0 0
0 1
1 1
4 0
Фигура успешно добавлена в дерево

add
Введите в порядке последовательного обхода фигуры точки в вида x y
0 0
0 1
1 1
3 0
Фигура успешно добавлена в дерево

add
Введите в порядке последовательного обхода фигуры точки в вида x y
0 0
0 1
1 1
5 0
Фигура успешно добавлена в дерево

add
Введите в порядке последовательного обхода фигуры точки в вида x y
0 0
0 1
1 1
99 0
Фигура успешно добавлена в дерево

print
**------|Coordinates: (0, 0) (0, 1) (1, 1) (99, 0) Area: 50*

#------|Coordinates: (0, 0) (0, 1) (1, 1) (7, 0) Area: 4

**------|Coordinates: (0, 0) (0, 1) (1, 1) (5, 0) Area: 3*

**------|Coordinates: (0, 0) (0, 1) (1, 1) (4, 0) Area: 2.5*

**------|Coordinates: (0, 0) (0, 1) (1, 1) (3, 0) Area: 2*

del
**------|Coordinates: (0, 0) (0, 1) (1, 1) (99, 0) Area: 50*

#------|Coordinates: (0, 0) (0, 1) (1, 1) (7, 0) Area: 4

**------|Coordinates: (0, 0) (0, 1) (1, 1) (5, 0) Area: 3*

**------|Coordinates: (0, 0) (0, 1) (1, 1) (4, 0) Area: 2.5*

**------|Coordinates: (0, 0) (0, 1) (1, 1) (3, 0) Area: 2*

Введите путь до желаемой фигуры (в формате lrl...)
Если необходимо удалить корень, введите 'root'

lr
print
**------|Coordinates: (0, 0) (0, 1) (1, 1) (99, 0) Area: 50*

#------|Coordinates: (0, 0) (0, 1) (1, 1) (7, 0) Area: 4

**------|Coordinates: (0, 0) (0, 1) (1, 1) (4, 0) Area: 2.5*

*------|Coordinates: (0, 0) (0, 1) (1, 1) (3, 0) Area: 2

del

*------|Coordinates: (0, 0) (0, 1) (1, 1) (99, 0) Area: 50

#------|Coordinates: (0, 0) (0, 1) (1, 1) (7, 0) Area: 4

*------|Coordinates: (0, 0) (0, 1) (1, 1) (4, 0) Area: 2.5

*------|Coordinates: (0, 0) (0, 1) (1, 1) (3, 0) Area: 2

Введите путь до желаемой фигуры (в формате lrl...)
Если необходимо удалить корень, введите 'root'

root
print
#------|Coordinates: (0, 0) (0, 1) (1, 1) (99, 0) Area: 50

*------|Coordinates: (0, 0) (0, 1) (1, 1) (4, 0) Area: 2.5

*------|Coordinates: (0, 0) (0, 1) (1, 1) (3, 0) Area: 2

get
#------|Coordinates: (0, 0) (0, 1) (1, 1) (99, 0) Area: 50

*------|Coordinates: (0, 0) (0, 1) (1, 1) (4, 0) Area: 2.5

*------|Coordinates: (0, 0) (0, 1) (1, 1) (3, 0) Area: 2

Введите путь до желаемой фигуры (в формате lrl...)
Если необходимо удалить корень, введите 'root'

ll
Coordinates: (0, 0) (0, 1) (1, 1) (3, 0) Area: 2
menu
Доступные функции:
add - добавить фигуру
get - получить фигуру
del - удалить фигуру

menu - вывести меню повторно
exit - выход

exit
Завершение работы

Тест №2:

Демонстрирует устойчивость программы к ошибкам

Доступные функции:
add - добавить фигуру
get - получить фигуру
del - удалить фигуру
menu - вывести меню повторно
exit - выход

sfsa
Некорректный ввод

add
Введите в порядке последовательного обхода фигуры точки в вида x y
sgagegv
sdgs
Ошибка!Некорректный ввод.

get
В дереве нет элементов!

del
В дереве нет элементов!

add
Введите в порядке последовательного обхода фигуры точки в вида x y
1 1
2 2
3 3
9 1
Введенная фигура не является трапецией

add
Введите в порядке последовательного обхода фигуры точки в вида x y
0 0
0 1

1 1
5 0
Фигура успешно добавлена в дерево

del
#------|Coordinates: (0, 0) (0, 1) (1, 1) (5, 0) Area: 3

Введите путь до желаемой фигуры (в формате lrl...)
Если необходимо удалить корень, введите 'root'

lrllr
Некорректный путь до элемента!

dad
Некорректный ввод

exit
Завершение работы

4. Результаты выполнения тестов

Представлены выше, с целью упростить прочтение.

5. Листинг программы

main.cpp

```
/*
Вариант 18:
Контейнер - Бинарное- Дерево

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру (одна из фигур ЛР3 на выбор).
Вариант структуры данных для контейнера выбрать из документа “Варианты структур данных”
(контейнер 1-го уровня) согласно своему номеру из  Варианты ЛР4..
*/

#include <iostream>
#include <vector>
#include <string>
#include <cmath>
```

```
using namespace std;

#include "BinTree.h"

bool is_number(const string& s) {
    bool point = false;
    for (int i = 0; i < s.length(); ++i) {
        if (s[i] == '-' && i == 0) {
            continue;
        }
        else if (s[i] == '.') {
            if ((i == 0 || i == s.length() - 1) || point) {
                return false;
            }
            else {
                point = true;
            }
        }
        else if (s[i] < '0' || s[i] > '9') { return false; }
    }
    return true;
}

class Point {
private:
    double x;
    double y;
public:
    Point() : x(0), y(0) {}
    Point(double _x, double _y) : x(_x), y(_y) {}

    double getX() const { return x; }
    double getY() const { return y; }
    friend istream& operator>> (istream& in, Point& p);
};

istream& operator>> (istream& in, Point& p) {

    string _x, _y; cin >> _x >> _y;

    if (!is_number(_x) || !is_number(_y)) {
        throw invalid_argument("Ошибка!Некорректный ввод.\n");
        //cout << "Ошибка! Некорректный ввод.\n";
        //exit(1);
    }

    p.x = stod(_x); p.y = stod(_y);
    return in;
```

```
}
ostream& operator<< (ostream& out, const Point& p) {
    cout << "(" << p.getX() << ", " << p.getY() << ")  ";
    return out;

}

double fabss(double a) {
    if (a >= 0) { return a; }
    else { return -a; }
}

double koef_nakl(const Point& a, const Point& b) {
    double k = (a.getY() - b.getY()) / (a.getX() - b.getX());
    return k;
}
bool trapezoid_check(const Point& a, const Point& b, const Point& c, const Point& d) {
    //case 1: ab || cd
    if (koef_nakl(a, b) == koef_nakl(c, d)) { return true; }
    //case 2: bc || ad
    else if (koef_nakl(b, c) == koef_nakl(d, a)) { return true; }
    return false;
}

class Trapezoid {
    Point a, b, c, d;
    vector<Point> v;
public:
    Trapezoid(Point _a, Point _b, Point _c, Point _d) : a(_a), b(_b), c(_c), d(_d) {
        v.push_back(_a);
        v.push_back(_b);
        v.push_back(_c);
        v.push_back(_d);
    }

    ~Trapezoid() {}

    double Area() const {
        double cur_res = 0;
        double res = 0;
        int i;
        for (i = 0; i < v.size() - 1; i++) {
            cur_res += v[i].getX() * v[i + 1].getY() - v[i].getY() * v[i + 1].getX();
        }

        cur_res += v[i].getX() * v[0].getY() - v[i].getY() * v[0].getX();
```

```
        res = fabss(cur_res) / 2;
        return res;
    }

    Point Geometric_center() const {
        double sumX = 0;
        double sumY = 0;
        for (int i = 0; i < v.size(); i++) {
            sumX += v[i].getX();
            sumY += v[i].getY();
        }
        Point a(sumX / v.size(), sumY / v.size());
        return a;
    }

    void Coordinates() const {
        cout << a << b << c << d;
    }

    void print() const {
        cout << "Coordinates: ";
        this->Coordinates();
        cout << "Area: " << this->Area();
        /*
        cout << "-----------------------------------------------\n";
        cout << "| Площадь: " << this->Area() << endl;
        Point cur = this->Geometric_center();
        cout << "| Геометрический центр:\n" << "| x = " << cur.getX() << "\n| y =  " << cur.getY()
<< endl;
        cout << "| Координаты: ";
        this->Coordinates();
        cout << "\n-----------------------------------------------";
        */
    }
};

bool operator< (const Trapezoid& a, const Trapezoid& b) {
    if (a.Area() < b.Area()) return true;
    else return false;
}

bool operator<= (const Trapezoid& a, const Trapezoid& b) {
    if (a.Area() <= b.Area()) return true;
    else return false;
}

bool operator> (const Trapezoid& a, const Trapezoid& b) {
    if (a.Area() > b.Area()) return true;
```

```
    else return false;
}

bool operator>= (const Trapezoid& a, const Trapezoid& b) {
    if (a.Area() >= b.Area()) return true;
    else return false;
}

bool operator== (const Trapezoid& a, const Trapezoid& b) {
    if (a.Area() == b.Area()) return true;
    else return false;
}

ostream& operator<< (ostream& out, const Trapezoid& t) {
    t.print();
    return out;

}

void menu() {
    cout << "Доступные функции:\nadd - добавить фигуру\nget - получить фигуру\ndel - удалить
фигуру\nmenu - вывести меню повторно\nexit - выход\n\n\n";
}

int main() {
        setlocale(LC_ALL, "rus");

    BinTree<Trapezoid> tree;
    string str, path;
    bool exit = false;
    menu();


    while (!exit) {
        try {
            cin >> str;
            if (str == "add") {
                Point a1, b1, c1, d1;
                cout << "Введите в порядке последовательного обхода фигуры точки в вида x y\n";
                cin >> a1 >> b1 >> c1 >> d1;
                if (trapezoid_check(a1, b1, c1, d1)) {
                    tree.insert(Trapezoid(a1, b1, c1, d1));
                    cout << "Фигура успешно добавлена в дерево\n\n";
                }
                else {
                    throw invalid_argument("Введенная фигура не является трапецией\n\n");
                }
            }
```

```
        else if (str == "get") {
            if (tree.empty()) {
                throw invalid_argument("В дереве нет элементов!\n\n");
            }
            else {
                tree.print();
                cout << "Введите путь до желаемой фигуры (в формате lrl...)\nЕсли необходимо удалить корень, введите 'root'\n\n";
                cin >> path;
                if (path == "root") {
                    tree.get("");
                }
                else {
                    tree.get(path);
                }
            }
        }
        else if (str == "del") {
            if (tree.empty()) {
                throw invalid_argument("В дереве нет элементов!\n\n");
            }
            else {
                tree.print();
                cout << "Введите путь до желаемой фигуры (в формате lrl...)\nЕсли необходимо удалить корень, введите 'root'\n\n";
                cin >> path;
                if (path == "root") {
                    tree.remove("");
                }
                else {
                    tree.remove(path);
                }
            }
        }
        else if (str == "print") {
            tree.print();
            //cout << tree;
        }
        else if (str == "menu") {
            menu();
        }
        else if (str == "exit") {
            cout << "Завершение работы\n\n";
            exit = true;
        }
        else {
            throw invalid_argument("Некорректный ввод\n\n");
        }
```

```
        }
        catch (invalid_argument& arg) {
            cout << arg.what() << endl;
        }
    }

    return 0;
}
```

BinTree.h

```
#pragma once
#ifndef Binnodeee_H
#define Binnodeee_H

#include "Node.h"


template< typename T >
class BinTree {
public:
    BinTree() : root(NULL) {};
    ~BinTree() { recursive_delete(root); };

    bool empty() {
        return root == nullptr;
    }

    void insert(const T& data) { recursive_insert(&root, data); }

    /*
    void remove(const T& data) {
        recursive_remove(&root, data, NULL);
    }*/


    void print() const { recursive_print(root, 0, false); }

    void get(const string& path) {
        recursive_search(&root, path, 0);
    }

    void remove(const string& path) {
        recursive_search(&root, path, 1);
    }
```

```
private:
    Node< T >* root;

    void recursive_delete(Node< T >* node) {
        if (node != NULL) {
            recursive_delete(node->lhs);
            recursive_delete(node->rhs);

            delete node;
        }
    }

    //по умолчанию при равенстве нового элемента и node дерева, новый элемент считаеся меньшим
    void recursive_insert(Node< T >** node, const T& data) {
        if (*node == NULL)
            *node = new Node< T >(data);
        else {
            if ((*node)->value >= data)
                recursive_insert(&((*node)->lhs), data);
            else if ((*node)->value < data)
                recursive_insert(&((*node)->rhs), data);
        }
    }

    void recursive_search(Node< T >** node, const string& path, int flag) {
        if ((path != "") && (*node == NULL)) {
            throw invalid_argument("Некорректный путь до элемента!\n\n");
        }

        if (path[0] == 'l')
            recursive_search(&((*node)->lhs), path.substr(1), flag);
        else if (path[0] == 'r')
            recursive_search(&((*node)->rhs), path.substr(1), flag);
        else if (path == "") {
            if (flag == 0) {
                cout << (*node)->value << endl;
            }
            else if (flag == 1){
                recursive_remove(node, (*node)->value, NULL);
            }
        }
        else {
            throw invalid_argument("Некорректный элемент описывает путь!\n\n");

            //cout << "Error string\n";
            //return;
```

```
    }
  }

  Node< T >* minimum(Node< T >* node)
  {
    if (!node->lhs->lhs) return node;
    return minimum(node->lhs);
  }

  void recursive_remove(Node< T >** node, const T& data, Node< T >* parent){
    if (!(*node)) return;

    if (data < (*node)->value)
      recursive_remove(&((*node)->lhs), data, *node);
    else if (data > (*node)->value)
      recursive_remove(&((*node)->rhs), data, *node);
    else {

      if (!(*node)->lhs && !(*node)->rhs) {///Если детей у удаляемого узла нет, то перед нами самый простой случай - листовой узел.
        if (parent) {///Родителю данного узла надо сообщить о том, что потомка у него теперь нет
          if (parent->lhs) {
            if (parent->lhs->value == (*node)->value) { //Если удаляется левый потомок
              parent->lhs = NULL;
            }
          }
          if (parent->rhs) {
            if (parent->rhs->value == (*node)->value) { //Если удаляется правый потомок
              parent->rhs = NULL;
            }
          }
        }
        delete *node; // Теперь можно освободить память
        *node = NULL;
      }
      else if (!(*node)->lhs || !(*node)->rhs) { // Если у удаляемой вершины есть хотя бы один потомок
        Node< T >* nodeToRemove = NULL;
        if ((*node)->lhs) { //Находим того самого единственного потомка удаляемой вершины
          nodeToRemove = (*node)->lhs;
        }
        else {
          nodeToRemove = (*node)->rhs;
        }
        //Скопировать все данные из единственного потомка удаляемой вершины
        (*node)->lhs = nodeToRemove->lhs;
        (*node)->rhs = nodeToRemove->rhs;
```

```
            (*node)->value = nodeToRemove->value;
            //Освободить память, выделенную ранее для данного потомка
            delete nodeToRemove;
        }
        else { //Если у удаляемой вершины есть оба потомка, то согласно алгоритму необходимо найти наименьший элемент в правом поддереве удаляемого элемента

            if (!(*node)->rhs->lhs) { //Если у правого поддерева нет левых потомков, то это означает, что у всех потомков значение ключа больше, а значит надо просто скопировать значения из правого потомка в удаляемый элемент

                (*node)->value = (*node)->rhs->value; // Скопировать значение из правого потомка
                Node< T >* righnodeIghtChild = (*node)->rhs->rhs;
                delete (*node)->rhs; // Освбодить память, выделенную для правого потомка
                (*node)->rhs = righnodeIghtChild;
            }
            else {
                Node< T >* minNodeParent = minimum((*node)->rhs); //Поиск наименьшего элемента в правом поддереве (он обязательно найдётся, случай когда его нет был разобран выше)
                (*node)->value = minNodeParent->lhs->value; //Скопировать значение из наименьшего жлемента в правом поддереве в удаляемый элемент
                delete minNodeParent->lhs;
                minNodeParent->lhs = NULL;
            }
        }
    }
}

void recursive_print(Node< T >* node, int level, bool isleft) const {
    if (node == NULL) {
        return;
    }

    recursive_print(node->rhs, level + 1, false);
    for (int i = 0; i < level; i++) {
        cout << "       ";
    }
    if (level != 0) {
        if (isleft) { cout << "*"; }
        else { cout << "*"; }
    }
    else { cout << "#"; }

    cout << "------|";
    cout << node->value << "\n\n";

    recursive_print(node->lhs, level + 1, true);
```

```
    }

};



#endif
```

Node.h

```
#pragma once
#ifndef Node_H
#define Node_H

template< typename T > class BinTree;

template< typename T >
class Node{
    friend class BinTree< T >;

public:
    Node() : lhs(NULL), rhs(NULL) {};
    Node(const T& data) :value(data), lhs(NULL), rhs(NULL) {};

    T getvalue() const{ return value; }
    Node*& get_left() {
        return lhs;
    }
    Node*& get_right() {
        return rhs;
    }
private:
    T value;
    Node< T >* lhs;
    Node< T >* rhs;
};

#endif
```

ВЫВОДЫ

В ходе этой лабораторной работы я улучшил свои навыки работы с классами и создания простых динамических структур данных. Несмотря на то, что на первом курсе я уже

сталкивался с такой задачей, как построить динамическое бинарное дерево, построить его на C++ с нынешним багажом знаний было интересно и являлось хорошей тренировкой.

ЛИТЕРАТУРА

1. Справочник по языку C++ [Электронный ресурс]. URL: https://ravesli.com (дата обращения:17.10.2021).