МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №2

по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент: *Сафонникова Анна Романовна, группа М8О-207Б-20*
Преподаватель: *Дорохов Евгений Павлович*

Условие

Задание: Вариант 22: N-Дерево (пятиугольник). Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру (колонка фигура 1), согласно вариантам задания. Классы должны удовлетворять следующим правилам:

1. Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
2. Классы фигур должны содержать набор следующих методов:
   - Перегруженный оператор ввода координат вершин фигуры из потока std::istream (»). Он должен заменить конструктор, принимающий координаты вершин из стандартного потока.
   - Перегруженный оператор вывода в поток std::ostream («), заменяющий метод Print из лабораторной работы 1.
   - Оператор копирования (=)
   - Оператор сравнения с такими же фигурами (==)
3. Класс-контейнер должен соджержать объекты фигур “по значению” (не по ссылке).

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.

Описание программы

Исходный код лежит в 10 файлах:

1. src/main.cpp: основная программа
2. include/figure.h: родительский класс-интерфейс для 5-угольника

3. include/point.h: описание класса точки
4. include/pentagon.h: описание класса 5-угольника, наследующегося от figures
5. include/TNaryTree.h: структура общего дерева
6. include/TNaryTree_item.h: структура элемента дерева
7. include/point.cpp: реализация класса точки
8. include/pentagon.cpp: реализация класса 5-угольника, наследующегося от figures
9. include/TNaryTree.cpp: реализация общего дерева
10. include/TNaryTree_item.cpp: реализация элемента дерева

Дневник отладки

Ошибка:
Отсутствие метода GetItem
Исправление:
Написать метод GetItem

Ошибка:
Ошибка при инициализации конструктора дерева. Причина ошибки в отсутствии конструктора Пентагона, принимающего точки как параметр
Исправление:
Небольшие корректировки в функции очищения

Недочёты

Недочётов не заметила.

Вывод

Благодаря данной лабораторной работе я реализовала такой класс-контейнер, как N-дерево, а так же конструткторы и функции для работы с ним. Выполнила перегрузку оператора вывода

Исходный код

main.cpp

```cpp
#include <iostream>
#include "pentagon.h"
#include "TNaryTree.h"

int main() {
  TNaryTree t(5);
  t.Update(Pentagon(Point(0,0), Point(0,1), Point(1,2), Point(2,1), Point(2,0)), "");
  t.Update(Pentagon(Point(0,0), Point(0,4), Point(4,5), Point(5,4), Point(5,0)), "b");
  t.Update(Pentagon(Point(0,0), Point(0,4), Point(4,5), Point(5,4), Point(5,0)), "bb");
  t.Update(Pentagon(Point(0,0), Point(0,4), Point(4,5), Point(5,4), Point(5,0)), "bbc");
  t.Update(Pentagon(Point(0,0), Point(0,4), Point(4,5), Point(5,4), Point(5,0)), "c");
  std::cout << t.size() << "\n";
  std::cout << t.Area("") << "\n";
  std::cout << t.size() << "\n";
  t.GetItem("").Print(std::cout);
  TNaryTree q(t);
  std::cout << q.size() << " " << q.Area("") << "\n";
  std::cout << t << '\n' << q;
  t.RemoveSubTree("");
  std::cout << t.Area("") << "\n";
}
```

figure.h

```cpp
#ifndef FIGURE_H
#define FIGURE_H

#include "point.h"

class Figure {
    public:
        virtual size_t VertexesNumber() = 0;
        virtual double Area() = 0;
        virtual void Print(std::ostream& os) = 0;
        ~Figure() {};
};

#endif
```

point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point {
    public:
        Point();
        Point(std::istream &is);
        Point(double x, double y);

        double dist(Point& other);
        double X();
        double Y();
        friend std::istream& operator>>(std::istream& is, Point& p);
        friend std::ostream& operator<<(std::ostream& os, Point& p);

    private:
        double x_;
        double y_;
};

#endif // POINT_H
```

point.cpp

```
#include "point.h"
#include <cmath>

Point::Point() : x_(0.0), y_(0.0) {}

Point::Point(double x, double y) : x_(x), y_(y) {}

Point::Point(std::istream &is) {
  is >> x_ >> y_;
}

double Point::dist(Point& other) {
  double dx = (other.x_ - x_);
  double dy = (other.y_ - y_);
  return std::sqrt(dx*dx + dy*dy);
}
```

```
double Point::X(){
  return x_;
};

double Point::Y(){
  return y_;
};

std::istream& operator>>(std::istream& is, Point& p) {
  is >> p.x_ >> p.y_;
  return is;
}

std::ostream& operator<<(std::ostream& os, Point& p) {
  os << "(" << p.x_ << ", " << p.y_ << ")";
  return os;
}
```

pentagon.h

```
#ifndef PENTAGON_H
#define PENTAGON_H

#include <iostream>

#include "figure.h"

class Pentagon : public Figure {

 private:
  Point t1;
  Point t2;
  Point t3;
  Point t4;
  Point t5;

 public:
  Pentagon();
  Pentagon(Point t1,Point t2,Point t3,Point t4,Point t5);
  Pentagon(const Pentagon &pentagon);
  Pentagon(std::istream &is);
  size_t VertexesNumber();
  double Area();
```

```
  void Print(std::ostream &os);
  static double Heron(Point, Point, Point);

};

#endif // PENTAGON_H
```

pentagon.cpp

```
#include "pentagon.h"

#include <iostream>
#include <cmath>

Pentagon::Pentagon()
    : t1(0, 0), t2(0, 0), t3(0, 0), t4(0, 0),
      t5(0, 0) {}

Pentagon::Pentagon(const Pentagon &pentagon) {
  this->t1 = pentagon.t1;
  this->t2 = pentagon.t2;
  this->t3 = pentagon.t3;
  this->t4 = pentagon.t4;
  this->t5 = pentagon.t5;
}

Pentagon::Pentagon(Point t1_,Point t2_,Point t3_,Point t4_,Point t5_):
t1(t1_), t2(t3_), t3(t3_), t4(t4_), t5(t5_){}

Pentagon::Pentagon(std::istream &is) {
  is >> t1 >> t2 >> t3 >> t4 >> t5;
}

size_t Pentagon::VertexesNumber() {
  return 5;
}

double Pentagon::Heron(Point A, Point B, Point C) {
  double AB = A.dist(B);
  double BC = B.dist(C);
  double AC = A.dist(C);
  double p = (AB + BC + AC) / 2;
  return sqrt(p * (p - AB) * (p - BC) * (p - AC));
```

```
}

double Pentagon::Area() {
  double area1 = Heron(t1, t2, t3);
  double area2 = Heron(t1, t4, t3);
  double area3 = Heron(t1, t4, t5);
  return area1 + area2 + area3;
}

void Pentagon::Print(std::ostream &os) {
  std::cout << "Pentagon: " << t1 << " " << t2 << " " << t3 << " " << t4
            << " " << t5 << " " << "\n";
}
```

TNaryTree.h

```
#ifndef TNARYTREE_H
#define TNARYTREE_H

#include "TNaryTree_item.h"
#include "pentagon.h"

class TNaryTree {
    public:
        TNaryTree(int n);
        TNaryTree(const TNaryTree& other);
        void RemoveSubTree(std::string &&tree_path = "");
        // Проверка наличия в дереве вершин
        bool Empty();
        // Подсчет суммарной площади поддерева
        double Area(std::string &&tree_path);
        int size();
        Pentagon  GetItem(const std::string&& tree_path="");
        // Вывод дерева в формате вложенных списков, где каждый вложенный список являетс
        // "S0: [S1: [S3, S4: [S5, S6]], S2]", где Si - площадь фигуры
        friend std::ostream& operator<<(std::ostream& os, const TNaryTree& tree);
        virtual ~TNaryTree();
    private:
        int curr_number;
        int max_number;
        Item* root;
};

#endif
```

TNaryTree.cpp

```cpp
#include "TNaryTree.h"
#include <string>
#include <stdexcept>

TNaryTree::TNaryTree(int n) {
    max_number = n;
    curr_number = 0;
    root = nullptr;
};

bool TNaryTree::Empty(){
    return curr_number ? 0 : 1;
}

void TNaryTree::Update(Pentagon &&polygon, std::string &&tree_path){
    if(tree_path != "" && curr_number == 0){
        throw std::invalid_argument("Error, there is not a root value\n");
        return;
    } else if(tree_path == "" && curr_number == 0){
        Item* q = (new Item(polygon));
        root = q;
        curr_number++;
    } else if(curr_number + 1 > max_number){
        throw std::out_of_range("Current number of elements equals maximal
        number of elements in tree\n");
        return;
    } else {
        Item* tmp = root;
        for(int i = 0; i < tree_path.length() - 1; i++){
            if(tree_path[i] == 'b'){
                Item* q = tmp->Get_bro();
                if(q == nullptr){
                    throw std::invalid_argument("Path does not exist\n");
                    return;
                }
                tmp = q;
            } else if(tree_path[i] == 'c'){
                Item* q = tmp->Get_son();
                if(q == nullptr){
                    throw std::invalid_argument("Path does not exist\n");
                    return;
```

```
                }
                tmp = q;
            } else {
                throw std::invalid_argument("Error in path\n");
                return;
            }
        }
        Item* item(new Item(polygon));
        if(tree_path.back() == 'b'){
            tmp->Set_bro(item);
            curr_number++;
        } else if(tree_path.back() == 'c'){
            tmp->Set_son(item);
            curr_number++;
        } else {
            throw std::invalid_argument("Error in path\n");
            return;
        }
    }
}

Item* copy(Item* root){
    if(!root){
        return nullptr;
    }
    Item *root_copy = new Item(root);
    root_copy->Set_bro(copy(root->Get_bro()));
    root_copy->Set_son(copy(root->Get_son()));
    return root_copy;
}

TNaryTree::TNaryTree(const TNaryTree& other){
    curr_number = 0;
    max_number = other.max_number;
    root = copy(other.root);
    curr_number = other.curr_number;
;}

int TNaryTree::size(){
    return curr_number;
}
```

```
int clear(Item* node) {
    if (!node) {
        return 0;
    }
    int temp_res = clear(node->Get_bro()) + clear(node->Get_son()) + 1;
    delete node;
    return temp_res;
}

Pentagon TNaryTree::GetItem(const std::string&& tree_path) {
  Item* tmp = root;
  for(int i = 0; i < tree_path.length(); i++){
    if(tree_path[i] == 'b'){
      Item* q = tmp->Get_bro();
      if(q == nullptr){
        throw std::invalid_argument("Path does not exist\n");
        return Pentagon();
      }
      tmp = q;
    } else if(tree_path[i] == 'c'){
      Item* q = tmp->Get_son();
      if(q == nullptr){
        throw std::invalid_argument("Path does not exist\n");
        return Pentagon();
      }
      tmp = q;
    } else {
      throw std::invalid_argument("Error in path\n");
      return Pentagon();
    }
  }
  return tmp->Get_data();
}
void TNaryTree::RemoveSubTree(std::string &&tree_path){
    Item* prev_tmp = nullptr;
    Item* tmp = root;
    if (tree_path.empty()) {
        clear(root);
        curr_number = 0;
        root = nullptr;
        return;
    }
```

```
    for(int i = 0; i < tree_path.length(); i++){
        if(tree_path[i] == 'b'){
            Item* q = tmp->Get_bro();
            if(q == nullptr){
                throw std::invalid_argument("Path does not exist\n");
                return;
            }
            prev_tmp = tmp;
            tmp = q;
        } else if(tree_path[i] == 'c'){
            Item* q = tmp->Get_son();
            if(q == nullptr){
                throw std::invalid_argument("Path does not exist\n");
                return;
            }
            prev_tmp = tmp;
            tmp = q;
        } else {
            throw std::invalid_argument("Error in path\n");
            return;
        }
    }
    if(tmp == prev_tmp->Get_son()) {
        prev_tmp->Set_son(nullptr);
    } else {
        prev_tmp->Set_bro(nullptr);
    }
    curr_number -= clear(tmp);
}

double area(Item* node){
    if(!node){
        return 0;
    }
    return node->Area() + area(node->Get_bro()) + area(node->Get_son());
}

double TNaryTree::Area(std::string &&tree_path){
    Item* tmp = root;
    for(int i = 0; i < tree_path.length(); i++){
        if(tree_path[i] == 'b'){
            Item* q = tmp->Get_bro();
```

```
            if(q == nullptr){
                throw std::invalid_argument("Path does not exist\n");
                return -1;
            }
            tmp = q;
        } else if(tree_path[i] == 'c'){
            Item* q = tmp->Get_son();
            if(q == nullptr){
                throw std::invalid_argument("Path does not exist\n");
                return -1;
            }
            tmp = q;
        } else {
            throw std::invalid_argument("Error in path\n");
            return -1;
        }
    }
    return area(tmp);
}

// Вывод дерева в формате вложенных списков, где каждый вложенный список является:
// "S0: [S1: [S3, S4: [S5, S6]], S2]", где Si - площадь фигуры

void print(std::ostream& os, Item* node){
    if(!node){
        return;
    }
    if(node->Get_son()){
        //os <<  <<node->pentagon.GetArea() <<  : ]" <<
        os << node->Area() << ": [";
        print(os, node->Get_son());
        if(node->Get_bro()){
            if(node->Get_bro()){
                os << ", ";
                print(os, node->Get_bro());
            }
        }
        os << "]";
    } else if (node->Get_bro()) {
       os << node->Area() << ": [";
        print(os, node->Get_bro());
        if(node->Get_son()){
```

```
            if(node->Get_son()){
                os << ", ";
                print(os, node->Get_son());
            }
        }
        os << "]";
    }
    else {
        os << node->Area();
    }
}

std::ostream& operator<<(std::ostream& os, const TNaryTree& tree){
    print(os, tree.root);
    os << "\n";
  return os;
}

TNaryTree::~TNaryTree(){
  RemoveSubTree();
};
```

TNaryTree_item.h

```
#ifndef TNARYTREE_ITEM_H
#define TNARYTREE_ITEM_H

#include "pentagon.h"

class Item {
 public:
  Item(Point t1, Point t2, Point t3, Point t4, Point t5);
  Item(Pentagon a);
  Item(Item *a);
  Item();
  void Print(std::ostream &os);
  Item *Get_bro();
  Item *Get_son();
  Pentagon Get_data();
  void Set_bro(Item *a);
  void Set_son(Item *a);
  double Area();
  ~Item();
```

```
 private:
  Item *bro = nullptr;
  Item *son = nullptr;
  Pentagon data;
};

#endif
```

TNaryTree_item.cpp

```
#include "TNaryTree_item.h"

Item::Item(Point t1, Point t2, Point t3, Point t4, Point t5){
    data = Pentagon(t1,t2,t3,t4,t5);
}

Item::Item(){
    data = Pentagon();
}


Item::Item(Pentagon a){
    data = a;
}

Item::Item(Item* a){
    bro = a->bro;
    son = a->son;
    data = a->data;
}

Item* Item::Get_bro(){
    return bro;
}

Pentagon Item::Get_data(){
  return data;
}

Item* Item::Get_son(){
    return son;
}
```

```
void Item::Set_bro(Item* a){
    bro = a;
}

void Item::Set_son(Item* a){
    son = a;
}

void Item::Print(std::ostream &os){
    os << data.Area();
}

double Item::Area(){
    return data.Area();
}

Item::~Item(){};
```