**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 3**

Тема: Умные указатели в языке С++

Студент: Ваньков Д.А.

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2018

1. Постановка задачи

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

Классы должны удовлетворять следующим правилам:

· Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.

· Класс-контейнер должен содержать объекты используя std:shared\_ptr<…>.

· Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.

· Класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).

· Класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера).

· Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<).

· Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.

· Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

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.

1. Решения задачи

Нужно четко понимать, чего мы хотим от программы, поэтому необходимо составить предварительные тесты, на которые впоследствии нужно опираться.

Пример предварительных нестандартных тестов, показывающих как программа должна работать:

После написания тестов, опираясь на них, можно писать код.

По итогу, выполнив все шаги, в директории должно оказаться 13 файлов: *Figure.h, Octagon.cpp, Octagon.h, Hexagon.cpp,Hexagon.h, Pentagon.h, Pentagon.cpp, Tree.cpp, Tree.h, Item.cpp, Item.h, main.cpp* и исполняемый *a.out*

1. Руководство по использованию программы

Краткий обзор модулей программы:

Figure.h (модуль отвечающий за описание общего родительского класса фигур)

Octagon.h (заголовочный файл, описывающий класс “Восьмиугольник”)

Octagon.cpp (файл, отвечающий за операции над восьмиугольником, содержащий все необходимые функции и переменные, для этого класса)

Pentagon.h (заголовочный файл, описывающий класс “Пятиугольник”)

Pentagon.cpp (файл, отвечающий за операции над пятиугольником, содержащий все необходимые функции и переменные, для этого класса)

Hexagon.h (заголовочный файл, описывающий класс “Шестиугольник”)

Hexagon.cpp (файл, отвечающий за операции над квадратом, содержащий все необходимые функции и переменные, для этого класса)

Tree.h (заголовочный файл, описывающий контейнер “дерево”)

Tree.cpp (файл, отвечающий за операции над деревом, такие как вставка, удаление и т.д.)

Item.h (заголовочный файла Item.cpp)

Item.cpp (файл, отвечающий за связь стека с фигурами, которыми он будет заполняться)

main.cpp (модуль, отвечающий за “лицо” программы, содержащий интерфейс, позволяющий работать с написанным в других файлах кодом)

Пример выходных данных:

C:\Users\dvankov\CLionProjects\OOP\_lab3\cmake-build-debug\OOP\_lab3.exe

Press 1 to get help

Press 2 to add figure in tree

Press 3 to get a number of nodes on level

Press 4 to print tree

Press 5 to delete figure from tree

Press 0 to exit

2

Choose the figure:

1 - Pentagon

2 - Hexagon

3 - Octagon

2

Insert side of Hexagon:

6

Tree item: created

2

Choose the figure:

1 - Pentagon

2 - Hexagon

3 - Octagon

1

Insert side of Pentagon:

2

Tree item: created

4

4

Printed tree:

[Hexagon with side = 6]

[Pentagon with side a = 2]

5

Enter side of figure to delete it:

2

Tree item: deleted

4

Printed tree:

[Hexagon with side = 6]

После запуска исполняемого файла, программа предлагает в помощь пользовательское меню работы с кодом, где каждая введенная цифра является командой начала процесса. В примере ниже предлагается меню, после чего выбирается цифра 2, соответствующая команде “добавить фигуру в дерево”. Цифрой 2 выбираем из списка фигур шестиугольник. Далее получаем просьбу ввести сторону шестиугольника, вводим. Повторяем предыдущие 2 шага тем самым добавляя еще 2 фигуры в дерево. Далее идет команда 4, отвечающая за печать контейнера, затем цифрой 5 удаляем фигуру. Чтобы убедиться в том, что фигура удалена опять печатаем дерево. Затем вводим 0 и выходим из программы.

1. Листинг программы

**main.cpp**

//Ваньков Денис М80-207Б-17

//Вариант 22(дерево, пятиугольник, шестиугольник, восьмиугольник)

#include <cstdlib>

#include <iostream>

#include <memory>

#include "Pentagon.h"

#include "Hexagon.h"

#include "Octagon.h"

#include "Item.h"

#include "Tree.h"

void help(){

std::cout << "Press 1 to get help" << std::endl;

std::cout << "Press 2 to add figure in tree" << std::endl;

std::cout << "Press 3 to get a number of nodes on level" << std::endl;

std::cout << "Press 4 to print tree" << std::endl;

std::cout << "Press 5 to delete figure from tree" << std::endl;

std::cout << "Press 0 to exit" << std::endl;

}

int main(int argc, const char \* argv[]) {

size\_t act;

TTree s;

help();

while (true) {

std::cin >> act;

if (act == 0) break;

if (act > 5) {

std::cout << "Error: enter another parameter (0-5)\n";

continue;

}

switch (act) {

case 1: {

help();

break;

}

case 2: {

size\_t type;

std::cout << "Choose the figure: \n";

std::cout << "1 - Pentagon\n";

std::cout << "2 - Hexagon\n";

std::cout << "3 - Octagon\n";

std::cin >> type;

if (type > 0) {

if (type > 3) {

std::cout << "Error: enter another parameter (0-3)\n";

continue;

}

switch (type) {

case 1: {

s.Push(std::shared\_ptr<TTreeItem> (new TTreeItem (std::shared\_ptr <Pentagon>(new Pentagon(std::cin)))));

break;

}

case 2: {

s.Push(std::shared\_ptr<TTreeItem> (new TTreeItem (std::shared\_ptr <Hexagon>(new Hexagon(std::cin)))));

break;

}

case 3: {

s.Push(std::shared\_ptr<TTreeItem> (new TTreeItem (std::shared\_ptr <Octagon>(new Octagon(std::cin)))));

break;

}

}

}

break;

}

case 3: {

int number = 0;

std::cout << "Enter level to get a number of nodes" << std::endl;

std::cin >> number;

std::cout << "On level " << number << " nodes: " << s.GetNodesOnLevel(number) << std::endl;

break;

}

case 4: {

std::cout << s << "\n";

break;

}

case 5: {

std::cout << "Enter side of figure to delete it: " << std::endl;

size\_t value;

std::cin >> value;

s.Pop(value);

break;

}

default: {

std::cerr << "Error: incorrect data\n";

break;

}

}

}

return 0;

}

**Tree.cpp**

#include <iostream>

#include "Tree.h"

#include "Item.h"

#include "Figure.h"

TTree::TTree() {

this->root = nullptr;

}

void TreeDel(std::shared\_ptr<TTreeItem> item) { // Деструктор для дерева

if(item) {

TreeDel(item->GetBrother());

TreeDel(item->GetSon());

item.reset();

}

}

TTree::~TTree() {

TreeDel(root);

std::cout << "Tree deleted" << std::endl;

}

int TTree::GetNodesOnLevel(std::shared\_ptr<TTreeItem> item, int level) { //Функция показывающаю, сколько элементов на определенном уровне

if (item == nullptr) return 0;

if (level <= 0) return 0;

return GetNodesOnLevel(item->GetSon(), level - 1) + (level == 1) + GetNodesOnLevel(item->GetBrother(), level);

}

bool TTree::empty() {

return root == nullptr;

}

std::shared\_ptr<TTreeItem> TTree::Push(std::shared\_ptr<TTreeItem> item)

{

if (this->empty()) {

this->root = item;

return this->root;

} else {

if (root->GetSon()) {

std::shared\_ptr<TTreeItem> tmp = root->GetSon();

while (tmp->GetBrother())

tmp = tmp->GetBrother();

tmp->SetBrother(item);

return tmp->GetBrother();

}

else {

root->SetSon(item);

return root->GetSon();

}

}

}

std::shared\_ptr<TTreeItem> tree\_find\_parent(std::shared\_ptr<TTreeItem> item, int value) { //функция по поиску предка по значению

if (item == nullptr) {

return nullptr;

}

if(item->GetFigure()->GetSide() == value) {

return item;

}

std::shared\_ptr<TTreeItem> parent\_node = tree\_find\_parent(item->GetSon(), value);

if (parent\_node == nullptr) {

parent\_node = tree\_find\_parent(item->GetBrother(), value);

}

return parent\_node;

}

int tree\_find\_parent1(std::shared\_ptr<TTreeItem> item, int value) { // Функция поиска предка по указателю на значение

int idx = 0;

if (item->GetFigure()->GetSide() == value) {

return 1;

}

if (item->GetSon() != nullptr){

idx = tree\_find\_parent1(item->GetSon() , value);

}

if (item->GetBrother() != nullptr){

idx = tree\_find\_parent1(item->GetBrother(), value);

}

if (idx == 1) {

item->GetBrother().reset();

item->GetBrother() = nullptr;

idx = 0;

}

return 0;

}

void split(std::shared\_ptr<TTreeItem> item) { //Функция для переприсваивания брата (перескок)

std::shared\_ptr<TTreeItem> tmp = item->GetBrother();

item->GetSon() = tmp->GetSon();

item->GetFigure() = tmp->GetFigure();

item->GetBrother() = tmp->GetBrother();

tmp.reset();

tmp = nullptr;

}

TTreeItem \*destroy\_Tree(std::shared\_ptr<TTreeItem> pointer) {

if(pointer == nullptr) {

return nullptr;

}

if (pointer->GetSon() != nullptr) {

destroy\_Tree(pointer->GetSon());

}

if (pointer->GetBrother() != nullptr) {

destroy\_Tree(pointer->GetBrother());

}

if (pointer->GetSon() == nullptr && pointer->GetBrother() == nullptr) {

pointer.reset();

pointer = nullptr;

}

return nullptr;

}

void TTree::Pop(size\_t value)

{

if (root->GetFigure()->GetSide() == value) {

root = nullptr;

}

else {

Pop(root, value);

}

}

void TTree::Pop(std::shared\_ptr<TTreeItem> item, size\_t value)

{

if (item->GetSon()) {

if (item->GetSon()->GetFigure()->GetSide() == value) {

std::shared\_ptr<TTreeItem> ptr = item->GetSon();

item->SetSon(item->GetSon()->GetBrother());

ptr->SetBrother(nullptr);

return;

}

else {

Pop(item->GetSon(), value);

}

}

if (item->GetBrother()) {

if (item->GetBrother()->GetFigure()->GetSide() == value) {

std::shared\_ptr<TTreeItem> ptr = item->GetBrother();

item->SetBrother(item->GetBrother()->GetBrother());

ptr->SetBrother(nullptr);

return;

}

else {

Pop(item->GetBrother(), value);

}

}

}

void TSpace(size\_t n){ // Функция расставляющая пробелы

for (size\_t i=0;i<=n;i++)

std::cout << " ";

}

void TreeRun(std::ostream &os, std::shared\_ptr<TTreeItem> item, size\_t space){ //Функция обхода дерева для печати

if (item) {

TSpace(space);

//os << \*item << std::endl;

std::cout << "[";

item->GetFigure()->Print();

std::cout << "]" << "\n";

if (item->GetBrother() != nullptr) {

TreeRun(os, item->GetBrother(),space);

}

if (item->GetSon() != nullptr) {

TreeRun(os, item->GetSon(),space+1);

}

}

}

std::ostream& operator<<(std::ostream &os, const TTree &tree) { //Оператор вывода дерева

std::shared\_ptr<TTreeItem> obj = tree.root;

os << "Printed tree:" << std::endl;

TreeRun(os, obj,1);

return os;

}

**Tree.h**

#ifndef OOP\_LAB3\_TREE\_H

#define OOP\_LAB3\_TREE\_H

#include "Figure.h"

#include "Item.h"

#include <stdbool.h>

#include <memory>

class TTree{

public:

TTree();

int GetNodesOnLevel(int level) {

return GetNodesOnLevel(root, level);

}

std::shared\_ptr<TTreeItem> Push(std::shared\_ptr<TTreeItem> item);

bool empty();

void Pop(int value){

Pop(root, value);

}

void Pop(size\_t value);

void Pop(std::shared\_ptr<TTreeItem> tree, size\_t value);

friend std::ostream &operator<<(std::ostream &os, const TTree &tree);

virtual ~TTree();

private:

void Pop(std::shared\_ptr<TTreeItem> item, int value);

int GetNodesOnLevel(std::shared\_ptr<TTreeItem> item, int level);

std::shared\_ptr<TTreeItem> root;

};

#endif //OOP\_LAB3\_TREE\_H

**Item.h**

#ifndef OOP\_LAB3\_ITEM\_H

#define OOP\_LAB3\_ITEM\_H

#include <stdio.h>

#include <memory>

#include "Figure.h"

class TTreeItem {

public:

TTreeItem(const std::shared\_ptr<Figure> &figure);

friend std::ostream& operator<<(std::ostream &os, const TTreeItem &obj);

std::shared\_ptr<TTreeItem> GetSon();

std::shared\_ptr<TTreeItem> GetBrother();

void SetSon(std::shared\_ptr<TTreeItem> son);

void SetBrother(std::shared\_ptr<TTreeItem> brother);

std::shared\_ptr<Figure> GetFigure() const;

virtual ~TTreeItem();

private:

std::shared\_ptr<Figure> figure;

std::shared\_ptr<TTreeItem> son;

std::shared\_ptr<TTreeItem> brother;

};

#endif //OOP\_LAB3\_ITEM\_H

**Item.cpp**

#include <iostream>

#include <cstdlib>

#include "Item.h"

#include "Figure.h"

TTreeItem::TTreeItem(const std::shared\_ptr<Figure> &figure) {

this->figure = figure;

this->son = nullptr;

this->brother = nullptr;

std::cout << "Tree item: created" << std::endl;

}

std::shared\_ptr<Figure> TTreeItem::GetFigure() const {

return this->figure;

}

std::shared\_ptr<TTreeItem> TTreeItem::GetSon() {

return this->son;

}

void TTreeItem::SetSon(std::shared\_ptr<TTreeItem> son) {

this->son = son;

}

void TTreeItem::SetBrother(std::shared\_ptr<TTreeItem> brother) {

this->brother = brother;

}

std::shared\_ptr<TTreeItem> TTreeItem::GetBrother() {

return this->brother;

}

TTreeItem::~TTreeItem() {

std::cout << "Tree item: deleted" << std::endl;

}

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

os << "[" << obj.GetFigure() << " " << obj.GetFigure()->Square() << "]";

return os;

}

**Hexagon.h**

#ifndef OOP\_LAB3\_HEXAGON\_H

#define OOP\_LAB3\_HEXAGON\_H

#include <iostream>

#include <cstdlib>

#include <cstdint>

#include "Figure.h"

class Hexagon : public Figure{

public:

Hexagon();

Hexagon(std::istream &is);

Hexagon(size\_t i);

size\_t GetSide() override;

double Square() override;

void Print() override;

bool operator==(const Hexagon &obj) const;

bool operator!=(const Hexagon &obj) const;

Hexagon &operator=(const Hexagon &obj);

friend std::ostream &operator<<(std::ostream &os, const Hexagon &obj);

friend std::istream &operator>>(std::istream &is, Hexagon &obj);

virtual ~Hexagon();

private:

size\_t side\_a;

};

#endif //OOP\_LAB3\_HEXAGON\_H

**Hexagon.cpp**

#include "Hexagon.h"

#include <istream>

#include <cmath>

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

}

Hexagon::Hexagon(size\_t i) : side\_a(i){

std::cout << "Hexagon created: " << side\_a << std::endl;

}

double Hexagon::Square() {

return 1.5 \* pow(side\_a,2) \* sqrt(3);

}

void Hexagon::Print() {

std::cout << "Hexagon with side = " << side\_a;

}

size\_t Hexagon::GetSide(){

return side\_a;

}

Hexagon::~Hexagon() {}

Hexagon::Hexagon(std::istream &is) {

while (true) {

is >> \*this;

if (std::cin.peek() == '\n') {

std::cin.get();

break;

}

else {

std::cerr << "Error: insert data" << "\n";

std::cin.clear();

while (std::cin.get() != '\n') {}

}

}

}

Hexagon &Hexagon::operator=(const Hexagon &obj) {

if (this == &obj) return \*this;

side\_a = obj.side\_a;

return \*this;

}

bool Hexagon::operator==(const Hexagon &obj) const {

return side\_a == obj.side\_a;

}

bool Hexagon::operator!=(const Hexagon &obj) const {

return side\_a != obj.side\_a;

}

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

os << "Hexagon with side: " << obj.side\_a << "\n";

return os;

}

std::istream &operator>>(std::istream &is, Hexagon &obj) {

std::cout << "Insert side of Hexagon: \n";

is >> obj.side\_a;

return is;

}

**Figure.h**

#ifndef OOP\_LAB3\_FIGURE\_H

#define OOP\_LAB3\_FIGURE\_H

#include <stdlib.h>

#include <iostream>

class Figure {

public:

virtual size\_t GetSide() = 0;

virtual double Square() = 0;

virtual void Print() = 0;

virtual ~Figure(){};

};

#endif //OOP\_LAB3\_FIGURE\_H

5.Вывод

Умные указатели - это объекты, которые хранят указатели на динамически аллоцированные участки памяти произвольного типа. Причем они автоматически очищают память по выходу из области видимости. Что является безусловно полезным дополнением при написании программы. Ведь с необработанными указателями программист должен явно уничтожить объект, когда он больше не нужен, следовательно, нужно отслеживать такие объекты, а когда их очень много, то задача становится вполне трудоемкой. Поэтому на помощь приходят умные указатели, при создании объекта больше не нужно беспокоиться о его уничтожении, когда умный указатель больше не используется, память, на которую указывает точка, освобождается.

СПИСОК ЛИТЕРАТУРЫ

1. Бьёрн Страуструп. Программирование: Принципы и практика использования C++, 2011, Издательский дом “Вильямс”.