**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 4**

Тема: Построение шаблонов динамических структур данных 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++ **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 содержать объекты используя std:shared\_ptr<…>.

· Шаблон класса-контейнера должен иметь метод по добавлению фигуры в контейнер.

· Шаблон класса-контейнера должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).

· Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера).

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

· Шаблон класса-контейнера должен иметь деструктор, удаляющий все элементы контейнера.

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

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:

Попробовать добавить фигуру с отрицательной стороной.

Удалить корень и распечатать дерево.

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 |
| Hexagon.h | Заголовочный файл, описывающий класс “Шестиугольник” |
| Hexagon.cpp | Файл, отвечающий за операции над шестиугольником, 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 |
| 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 дерева 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 <cstdint>

#include "Pentagon.h"

#include "Hexagon.h"

#include "Octagon.h"

#include "Item.h"

#include "Tree.h"

#include "Figure.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<Figure> 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<Figure>> (new TTreeItem<Figure> (std::shared\_ptr <Pentagon>(new Pentagon(std::cin)))));

break;

}

case 2: {

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

break;

}

case 3: {

s.Push(std::shared\_ptr<TTreeItem<Figure>> (new TTreeItem<Figure> (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"

template <class T>

TTree<T>::TTree() {

this->root = nullptr;

}

template <class T>

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

if(item) {

TreeDel(item->GetBrother());

TreeDel(item->GetSon());

item.reset();

}

}

template <class T>

TTree<T>::~TTree() {

TreeDel(root);

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

}

template <class T>

int TTree<T>::GetNodesOnLevel(std::shared\_ptr<TTreeItem<T>> 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);

}

template <class T>

bool TTree<T>::empty() {

return root == nullptr;

}

template <class T>

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

{

if (this->empty()) {

this->root = item;

return this->root;

} else {

if (root->GetSon()) {

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

while (tmp->GetBrother())

tmp = tmp->GetBrother();

tmp->SetBrother(item);

return tmp->GetBrother();

}

else {

root->SetSon(item);

return root->GetSon();

}

}

}

template <class T>

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

if (parent\_node == nullptr) {

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

}

return parent\_node;

}

template <class T>

int tree\_find\_parent1(std::shared\_ptr<TTreeItem<T>> 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;

}

template <class T>

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

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

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

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

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

tmp.reset();

tmp = nullptr;

}

template <class T>

TTreeItem<T> \*destroy\_Tree(std::shared\_ptr<TTreeItem<T>> 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;

}

template <class T>

void TTree<T>::Pop(size\_t value) // Вызов функции удаления

{

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

root = nullptr;

}

else {

Pop(root, value);

}

}

template <class T>

void TTree<T>::Pop(std::shared\_ptr<TTreeItem<T>> item, size\_t value) // Удаление элемента из дерева

{

if (item->GetSon()) {

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

std::shared\_ptr<TTreeItem<T>> 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<T>> 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 << " ";

}

template <class T>

void TreeRun(std::ostream &os, std::shared\_ptr<TTreeItem<T>> 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);

}

}

}

template <class A>

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

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

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

TreeRun(os, obj,1);

return os;

}

template class TTree<Figure>;

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

**Tree.h**

#ifndef OOP\_LAB4\_TREE\_H

#define OOP\_LAB4\_TREE\_H

#include "Figure.h"

#include "Item.h"

#include <stdbool.h>

#include <memory>

#include <cstdint>

template <class T> class TTree{

public:

TTree();

int GetNodesOnLevel(int level) {

return GetNodesOnLevel(root, level);

}

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

bool empty();

void Pop(size\_t value);

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

template <class A>

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

virtual ~TTree();

private:

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

std::shared\_ptr<TTreeItem<T>> root;

};

#endif //OOP\_LAB4\_TREE\_H

**Item.h**

#ifndef OOP\_LAB4\_ITEM\_H

#define OOP\_LAB4\_ITEM\_H

#include <stdio.h>

#include <memory>

#include "Figure.h"

template <class T>

class TTreeItem {

public:

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

template <class A>

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

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

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

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

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

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

private:

std::shared\_ptr<T> figure;

std::shared\_ptr<TTreeItem<T>> son;

std::shared\_ptr<TTreeItem<T>> brother;

};

#endif //OOP\_LAB4\_ITEM\_H

**Item.cpp**

#include <iostream>

#include <cstdlib>

#include "Item.h"

#include "Figure.h"

template <class T>

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

this->figure = figure;

this->son = nullptr;

this->brother = nullptr;

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

}

template <class T>

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

return this->figure;

}

template <class T>

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

return this->son;

}

template <class T>

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

this->son = son;

}

template <class T>

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

this->brother = brother;

}

template <class T>

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

return this->brother;

}

template <class A>

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

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

return os;

}

5.Вывод

Шаблоны— средство языка [C++](https://ru.wikipedia.org/wiki/C%2B%2B), предназначенное для кодирования [обобщенных алгоритмов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), без привязки к некоторым параметрам (например, [типам данных](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), размерам буферов, значениям по умолчанию). В [C++](https://ru.wikipedia.org/wiki/C%2B%2B) возможно создание шаблонов [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). 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](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) создает отдельный экземпляр функции или класса. Как следствие, исчезает возможность совместного использования скомпилированного кода в рамках разделяемых библиотек.

СПИСОК ЛИТЕРАТУРЫ

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