**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №5**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: **Полиморфная логика**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Ковынев М.В. |
| Преподаватель |  | Терентьев А.О. |

Санкт-Петербург

2018

**Цель работы.**

Изучить умные указатели на примере *shared\_ptr.*

**Основные теоретические положения.**

Класс shared\_ptr описывает объект, который использует подсчет ссылок для управления ресурсами. Объект shared\_ptr фактически содержит указатель на ресурс, которым он владеет, или содержит пустой указатель (NULL). Обладать ресурсом могут несколько объектов shared\_ptr; при удалении последнего объекта shared\_ptr, обладающего тем или иным ресурсом, данный ресурс освобождается.

shared\_ptr прекращает владеть ресурсом при переназначении или сбросе.

Аргумент шаблона T может быть неполным типом, за исключением случаев, особо отмеченных для определенных функций-членов.

**Постановка задачи.**

Объединить предыдущие работы в приложении, использующем логику полиморфного хранения объектов. Необходимо сгененрировать контейнер из 1000 фигур, которые хранятся как shared\_ptr<Shape>, и применить к ним 2 стандартных алгоритма по вариантам. В качестве предиката использовать предикат из дополнительного задания 2-й лабораторной.

Если вы считаете, что вашу функцию из 2-й лабораторной невозможно использовать в качестве критерия для индивидуального задания (например, если она возвращает что-то отличное от булевых или числовых значений), то необходимо самостоятельно придумать предикат, использующий вируальные функции в дереве классов. Пример: а) сортировка фигур в порядке возрастания их площади, б) отбор фигур, пересекающихся с заданной эталонной фигурой, в) замена фигур, периметр которых меньше определённого порога, на окружность единичного радиуса.

Описание вашего вашего задания приложить в виде файла README. Корректность алгоритмов доказать с помощью юнит тестов и/или ручного тестирования. В случае юнит-тестов тестирование достаточно провести для контейнера небольшого размера (5-10) заданных фигур.

Вариант задания:

Немодифицирующие алгоритмы

* Проверка двух упорядоченных диапазонов на равенство по определенному критерию.

Модифицирующие алгоритмы

* Упорядочить элементы, пока первые n элементов не будут следовать в требуемом порядке.

**Ход работы.**

1. Написана функция *vector<shared\_ptr<Shape>> generateFigures(int size)*, которая по заданному размеру создает вектор указателей на класс Shape различных фигур. Фигуры: прямоугольник, круг, трапеция.
2. Написана функция компаратора для *qsort compareFiguresByName(const void \* x1, const void \* x2)*, которая сравнивает два принятых объекта по имени фигуры и возвращает их разность
3. Написана функция *isSimiliarRegion(Region r1, Region r2, const stepik::vector<shared\_ptr<Shape>>& vect\_a),* которая принимает две переменные типа Region, хранящие начальные и конечные значение диапазона типа int и ссылку на вектор, в котором будет произведен поиск. Изначально выделяются два подвектора из данного вектора по заданным диапазонам и сортируются быстрой сортировкой по имени фигуры. После этого, если имена фигур одного вектора идут в том же порядке, что и в другом, тогда диапазоны равны.
4. Написана функция *void sortFirstNElements(stepik::vector<shared\_ptr <Shape>> & vect\_a, int n)*, которая принимает вектор и целое число n. Функция сортирует данный контейнер пузырьковой сортировкой до момента, позициях не окажутся фигуры с минимальными площадями, упорядоченными по возрастанию.
5. Функция *void printVector(const stepik::vector<shared\_ptr<Shape>>& vect\_a*), выводит информацию о векторе (см. Приложение А).
6. Написаны тесты для заданных 2 модифицирующей и немодифицирующей функций (см. Приложение Б).
7. Написан makefile для удобной сборки программы (см. Приложение В).

**Выводы.**

В ходе выполнения данной лабораторной работы была изучена полиморфная логика.

Приложение А

**КОД ALGORITHMSFORSHAPES.CPP**

#include "AlgorithmsForShapes.h"

#include <string>

stepik::vector<shared\_ptr<Shape>> generateFigures(int size)

{

stepik::vector<shared\_ptr<Shape>> vect\_a(size);

for (int i = 0; i < size; i++)

{

int figureType = rand() % 3;

switch (figureType)

{

case 0:

{

vect\_a[i] = shared\_ptr<Shape>(new Rectangle({ 0,0 }, 20, 10));

break;

}

case 1:

{

vect\_a[i] = shared\_ptr<Shape>(new Circle({ 0,0 }, 4));

break;

}

case 2:

{

vect\_a[i] = shared\_ptr<Shape>(new Trapeze({ 2, 6 }, { 7, 6 }, { 8, 0 }, { 0,0 }));

break;

}

}

}

return vect\_a;

}

int compareFiguresBySquare(const void \* x1, const void \* x2)

{

return (\*(\*(shared\_ptr<Shape>\*)x1)).getSquare() - (\*(\*(shared\_ptr<Shape>\*)x2)).getSquare();

}

int compareFiguresByName(const void \* x1, const void \* x2)

{

return typeid((\*(\*(shared\_ptr<Shape>\*)x1))).name() - typeid((\*(\*(shared\_ptr<Shape>\*)x2))).name();

}

// Проверка двух упорядоченных диапазонов на равенство по определенному критерию

// Упорядоченные диапазоны по имени

// Проверка диапазонов на совпадение имен фигур

bool isSimiliarRegion(Region r1, Region r2, const stepik::vector<shared\_ptr<Shape>>& vect\_a)

{

stepik::vector<shared\_ptr<Shape>> a(vect\_a.begin() + r1.start, vect\_a.begin() + r1.finish);

stepik::vector<shared\_ptr<Shape>> b(vect\_a.begin() + r2.start, vect\_a.begin() + r2.finish);

qsort(&a[0], a.size(), sizeof(shared\_ptr<Shape>), compareFiguresByName);

qsort(&b[0], b.size(), sizeof(shared\_ptr<Shape>), compareFiguresByName);

for (int i = 0; i < a.size(); i++)

{

string name1 = typeid(\*a[i]).name();

string name2 = typeid(\*b[i]).name();

if (name1 != name2)

return false;

}

return true;

}

// Упорядочить элементы, пока первые n элементов не будут следовать в требуемом порядке.

// Критерий - увеличение площади фигур

void sortFirstNElements(stepik::vector<shared\_ptr<Shape>>& vect\_a, int n)

{

qsort(&vect\_a[0], n, sizeof(shared\_ptr<Shape>), compareFiguresBySquare);

}

void printVector(const stepik::vector<shared\_ptr<Shape>>& vect\_a)

{

for (int i = 0; i < vect\_a.size(); i++)

cout << i << " Figure Square: " << (\*vect\_a[i]).getSquare() << "\n";

}

Приложение Б

**КОД TEST.CPP**

#include "test.h"

#include "AlgorithmsForShapes.h"

void checkSimiliarRegion()

{

// ===================== CHECK SIMILIAR REGION ===================== //

stepik::vector<shared\_ptr<Shape>> test1(6);

test1[0] = shared\_ptr<Shape>(new Rectangle({ 0,0 }, 20, 10));

test1[1] = shared\_ptr<Shape>(new Circle({ 0,0 }, 10));

test1[2] = shared\_ptr<Shape>(new Rectangle({ 0,0 }, 5, 40));

test1[3] = shared\_ptr<Shape>(new Rectangle({ 0,0 }, 40, 5));

test1[4] = shared\_ptr<Shape>(new Rectangle({ 0,0 }, 200, 1));

test1[5] = shared\_ptr<Shape>(new Rectangle({ 0,0 }, 1, 200));

Region test\_r1 = { 0, 3 };

Region test\_r2 = { 2, 5 };

assert(isSimiliarRegion(test\_r1, test\_r2, test1) == 0);

test\_r2 = { 1, 4 };

assert(isSimiliarRegion(test\_r1, test\_r2, test1) == 1);

test\_r1 = { 2, 4 };

test\_r2 = { 3, 5 };

assert(isSimiliarRegion(test\_r1, test\_r2, test1) == 1);

// ===================== END CHECK SIMILIAR REGION ===================== //

}

void checkSort()

{

// ============================= CHECK SORT ============================ //

stepik::vector<shared\_ptr<Shape>> test2(8);

test2[0] = shared\_ptr<Shape>(new Circle({ 0,0 }, 6));

test2[1] = shared\_ptr<Shape>(new Circle({ 0,0 }, 5));

test2[2] = shared\_ptr<Shape>(new Circle({ 0,0 }, 4));

test2[3] = shared\_ptr<Shape>(new Circle({ 0,0 }, 3));

test2[4] = shared\_ptr<Shape>(new Circle({ 0,0 }, 2));

test2[5] = shared\_ptr<Shape>(new Circle({ 0,0 }, 1));

test2[6] = shared\_ptr<Shape>(new Circle({ 0,0 }, 0.01));

test2[7] = shared\_ptr<Shape>(new Circle({ 0,0 }, 0.02));

sortFirstNElements(test2, 3);

assert(fabs((\*test2[0]).getSquare() - 0.000314159) <= 0.001);

assert(fabs((\*test2[1]).getSquare() - 0.00125664) <= 0.001);

assert(fabs((\*test2[2]).getSquare() - 3.14159) <= 0.001);

assert(fabs((\*test2[3]).getSquare() - 113.097) <= 0.001);

assert(fabs((\*test2[4]).getSquare() - 78.5398) <= 0.001);

assert(fabs((\*test2[5]).getSquare() - 50.2655) <= 0.001);

assert(fabs((\*test2[6]).getSquare() - 28.2743) <= 0.001);

assert(fabs((\*test2[7]).getSquare() - 12.5664) <= 0.001);

// ========================= END CHECK SORT ========================== //

cout << "If you see this, assert tests complete successfully!\n";

}

Приложение В

**КОД MAKEFILE**

result: main.o AlgorithmsForShapes.o test.o

g++ main.o -o result.out AlgorithmsForShapes.o test.o

rm \*.o

./result.out

main.o: main.cpp

g++ -std=c++11 -c main.cpp

AlgorithmsForShapes.o: AlgorithmsForShapes.cpp AlgorithmsForShapes.h

g++ -std=c++11 -c AlgorithmsForShapes.cpp

test.o: test.cpp test.h

g++ -std=c++11 -c test.cpp