Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

за 1 семестр

По дисциплине: «Языки программирования»

Тема: «Перегрузка операторов и обработка исключений»

Выполнил:

Студент 2 курса

Группы ПО-4(1)

Иваненко И. Л.

Проверил:

Хацкевич М. В.

2020

Лабораторная работа №1

Цель работы:

Вариант 10

Общие требования: В начале программы вывести задание; в процессе работы выводить подсказки пользователю (что ему нужно ввести, чтобы продолжить выполнение программы). Класс коллекция может не иметь методов для изменения количества хранимых объектов. При обращении к элементам с несуществующим индексом должно выбрасываться исключение. После работы программы вся динамически выделенная память должна быть освобождена.

Задание по варианту (Средний уровень сложности): Написать программу, в которой описана иерархия классов: геометрические фигуры (куб, цилиндр, тетраэдр). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общего объема и площади поверхности. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

Иерерахия классов:



Figure.h:

#pragma once

class Figure

{

protected:

float area;

float volume;

virtual void calculateArea() = 0;

virtual void calculateVolume() = 0;

public:

virtual float getArea() = 0;

virtual float getVolume() = 0;

virtual void infoOutput() = 0;

virtual ~Figure() {};

};

Cube.h:

#pragma once

#include "Figure.h"

class Cube : public Figure

{

private:

float edgeLength;

protected:

void calculateArea() override;

void calculateVolume() override;

public:

Cube(float edgeLength);

Cube();

float getArea() override;

float getVolume() override;

float getEdgeLength();

void infoOutput() override;

Cube& operator = (const Cube& other);

bool operator == (const Cube& other);

bool operator != (const Cube& other);

~Cube() {};

};

Cube.cpp:

#include "Cube.h"

#include <iostream>

#include <cmath>

using namespace std;

Cube::Cube() {

this->edgeLength = 1;

this->calculateArea();

this->calculateVolume();

}

Cube::Cube(float edgeLength) {

this->edgeLength = edgeLength;

this->calculateArea();

this->calculateVolume();

}

void Cube::calculateArea() {

this->area = 6 \* pow(this->edgeLength, 2);

}

void Cube::calculateVolume() {

this->volume = pow(this->edgeLength, 3);

}

float Cube::getArea() {

return this->area;

}

float Cube::getVolume() {

return this->volume;

}

float Cube::getEdgeLength() {

return this->edgeLength;

}

void Cube::infoOutput() {

cout << endl << "ДЛИНА РЕБРА КУБА: " << this->getEdgeLength() << endl;

cout << "ПЛОЩАДЬ КУБА: " << this->getArea() << endl;

cout << "ОБЪЕМ КУБА: " << this->getVolume() << endl;

}

Cube& Cube::operator = (const Cube& other) {

this->edgeLength = other.edgeLength;

this->area = other.area;

this->volume = other.volume;

return \*this;

}

bool Cube::operator == (const Cube& other) {

return this->edgeLength == other.edgeLength;

}

bool Cube::operator != (const Cube& other) {

return this->edgeLength != other.edgeLength;

}

Cylinder.h:

#pragma once

#include "Figure.h"

class Cylinder : public Figure

{

private:

float height;

float baseRadius;

protected:

void calculateArea() override;

void calculateVolume() override;

public:

Cylinder();

Cylinder(float height, float baseRadius);

float getArea() override;

float getVolume() override;

float getHeight();

float getBaseRadius();

void infoOutput() override;

Cylinder& operator = (const Cylinder& other);

bool operator == (const Cylinder& other);

bool operator != (const Cylinder& other);

~Cylinder() {};

};

Cylinder.cpp:

#define \_USE\_MATH\_DEFINES

#include "Cylinder.h"

#include <iostream>

#include "math.h"

using namespace std;

Cylinder::Cylinder() {

this->height = 1;

this->baseRadius = 1;

this->calculateArea();

this->calculateVolume();

}

Cylinder::Cylinder(float height, float baseRadius) {

this->height = height;

this->baseRadius = baseRadius;

this->calculateArea();

this->calculateVolume();

}

void Cylinder::calculateArea() {

this->area = 2 \* M\_PI \* this->baseRadius \* (this->baseRadius + this->height);

}

void Cylinder::calculateVolume() {

this->volume = M\_PI \* pow(this->baseRadius, 2) \* this->height;

}

float Cylinder::getArea() {

return this->area;

}

float Cylinder::getVolume(){

return this->volume;

}

float Cylinder::getBaseRadius() {

return this->baseRadius;

}

float Cylinder::getHeight() {

return this->height;

}

void Cylinder::infoOutput() {

cout << endl << "ВЫСОТА ЦИЛИНДРА: " << this->getHeight() << endl;

cout << "РАДИУС ОСНОВАНИЯ ЦИЛИНДРА: " << this->getBaseRadius() << endl;

cout << "ПЛОЩАДЬ ЦИЛИНДРА: " << this->getArea() << endl;

cout << "ОБЪЕМ ЦИЛИНДРА: " << this->getVolume() << endl;

}

Cylinder& Cylinder::operator = (const Cylinder& other) {

this->height = other.height;

this->baseRadius = other.baseRadius;

this->area = other.area;

this->volume = other.volume;

return \*this;

}

bool Cylinder::operator == (const Cylinder& other) {

return (this->height == other.height && this->baseRadius == other.baseRadius);

}

bool Cylinder::operator != (const Cylinder& other) {

return (this->height != other.height || this->baseRadius != other.baseRadius);

}

Tetrahedron.h:

#pragma once

#include "Figure.h"

class Tetrahedron : public Figure

{

private:

float edgeLength;

float height;

protected:

void calculateArea() override;

void calculateVolume() override;

void calculateHeight();

public:

Tetrahedron();

Tetrahedron(float edgeLength);

float getArea() override;

float getVolume() override;

float getEdgeLength();

float getHeight();

void infoOutput() override;

Tetrahedron& operator = (const Tetrahedron& other);

bool operator == (const Tetrahedron& other);

bool operator != (const Tetrahedron& other);

~Tetrahedron() {};

};

Tetrahedron.cpp:

#include "Tetrahedron.h"

#include <iostream>

#include <cmath>

using namespace std;

Tetrahedron::Tetrahedron() {

this->edgeLength = 1;

this->calculateHeight();

this->calculateArea();

this->calculateVolume();

}

Tetrahedron::Tetrahedron(float edgeLength) {

this->edgeLength = edgeLength;

this->calculateHeight();

this->calculateArea();

this->calculateVolume();

}

void Tetrahedron::calculateHeight() {

this->height = (this->edgeLength \* sqrt(6)) / 2;

}

void Tetrahedron::calculateArea() {

this->area = pow(this->edgeLength, 2) \* sqrt(6);

}

void Tetrahedron::calculateVolume() {

this->volume = (pow(this->edgeLength, 3) \* sqrt(2)) / 12;

}

float Tetrahedron::getEdgeLength() {

return this->edgeLength;

}

float Tetrahedron::getHeight() {

return this->height;

}

float Tetrahedron::getArea() {

return this->area;

}

float Tetrahedron::getVolume() {

return this->volume;

}

void Tetrahedron::infoOutput() {

cout << endl << "ДЛИНА РЕБРА ТЕТРАЭДРА: " << this->getEdgeLength() << endl;

cout << "ВЫСОТА ТЕТРАЭДРА: " << this->getHeight() << endl;

cout << "ПЛОЩАДЬ ТЕТРАЭДРА: " << this->getArea() << endl;

cout << "ОБЪЕМ ТЕТРАЭДРА: " << this->getVolume() << endl;

}

Tetrahedron& Tetrahedron::operator = (const Tetrahedron& other) {

this->edgeLength = other.edgeLength;

this->height = other.height;

this->area = other.area;

this->volume = other.volume;

return \*this;

}

bool Tetrahedron::operator == (const Tetrahedron& other) {

return this->height == other.height;

}

bool Tetrahedron::operator != (const Tetrahedron& other) {

return this->height != other.height;

}

Container.h:

#pragma once

#include "Figure.h"

class Container

{

private:

Figure\*\* container;

int size;

public:

Container(int size);

void push\_back(Figure\* figure);

void insert(Figure\* figure, int index);

void delete\_back();

void index\_delete(int index);

static Container\* containerInitialize();

void containerOutput();

int getSize();

Figure\*& operator[](const int index);

~Container();

};

Container.cpp:

#include "Container.h"

#include "Cube.h"

#include "Cylinder.h"

#include "Tetrahedron.h"

#include <iostream>

#include <cassert>

using namespace std;

Container::Container(int size) {

this->size = size;

this->container = new Figure\* [this->size];

}

void Container::push\_back(Figure\* figure) {

Figure\*\* tempContainer = new Figure \* [this->size + 1];

for (int i = 0; i < this->size; i++) {

tempContainer[i] = this->container[i];

}

tempContainer[this->size] = figure;

this->size++;

delete[] this->container;

this->container = tempContainer;

}

void Container::insert(Figure\* figure, int index) {

Figure\*\* tempContainer = new Figure \* [this->size + 1];

for (int i = 0; i < index; i++) {

tempContainer[i] = this->container[i];

}

tempContainer[index] = figure;

for (int i = index + 1; i < this->size + 1; i++) {

tempContainer[i] = this->container[i - 1];

}

this->size++;

delete[] this->container;

this->container = tempContainer;

}

void Container::index\_delete(int index) {

Figure\*\* tempContainer = new Figure \* [this->size - 1];

for (int i = 0; i < index; i++) {

tempContainer[i] = this->container[i];

}

for (int i = index; i < this->size; i++) {

tempContainer[i] = this->container[i + 1];

}

this->size--;

delete[] this->container;

this->container = tempContainer;

}

void Container::delete\_back() {

Figure\*\* tempContainer = new Figure \* [this->size - 1];

for (int i = 0; i < this->size - 1; i++) {

tempContainer[i] = this->container[i];

}

this->size--;

delete[] this->container;

this->container = tempContainer;

}

Container\* Container::containerInitialize() {

cout << "\nВведите размер контейнера >>> ";

int size;

cin >> size;

Container\* container = new Container(0);

cout << "\nВВОДИТЕ ДАННЫЕ\n";

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

cout << endl;

cout << "1 - КУБ\n";

cout << "2 - ЦИЛИНДР\n";

cout << "3 - ТЕТРАЕДР\n";

cout << "\nВыберите фигуру >>> ";

int figure;

cin >> figure;

switch (figure) {

case 1:

cout << "\nВведите длину ребра куба >>> ";

float edgeLength;

cin >> edgeLength;

container->push\_back(new Cube(edgeLength));

break;

case 2:

cout << "\nВведите высоту цилиндра >>> ";

float height;

cin >> height;

cout << "\nВведите радиус основания цилиндра >>> ";

float baseRadius;

cin >> baseRadius;

container->push\_back(new Cylinder(height, baseRadius));

break;

case 3:

cout << "\nВведите длину ребра тетраедра >>> ";

float edgeLength2;

cin >> edgeLength2;

container->push\_back(new Tetrahedron(edgeLength2));

break;

default:

cout << "\n НЕВЕРНЫЙ ВВОД!!! \n";

i--;

break;

}

}

return container;

}

void Container::containerOutput() {

for (int i = 0; i < this->size; i++) {

if (this->container[i] != nullptr) {

this->container[i]->infoOutput();

}

}

}

int Container::getSize() {

return this->size;

}

Figure\*& Container::operator [] (const int index) {

assert(index >= 0 && index < this->size);

return this->container[index];

}

Container::~Container() {

for (int i = 0; i < this->size; i++) {

if (this->container[i] != nullptr) {

delete this->container[i];

}

}

delete[] this->container;

}

menu.h:

#pragma once

void menu();

menu.cpp:

#include "menu.h"

#include "Cube.h"

#include "Cylinder.h"

#include "Tetrahedron.h"

#include "Container.h"

#include <iostream>

using namespace std;

/\*

1 - вывод массива

2 - вывод элемента по индексу

3 - удаление элемента массива

4 - добавление элемента в массив

5 - сравнение кубов

6 - выход

\*/

void menu() {

Container\* container = Container::containerInitialize();

Cube\* cube1;

Cube\* cube2;

int choice;

do {

cout << endl;

cout << "1 - вывод массива\n";

cout << "2 - вывод элемента по индексу\n";

cout << "3 - удаление элемента массива\n";

cout << "4 - добавление элемента в массив\n";

cout << "5 - сравнение кубов\n";

cout << "6 - демонстрация работы оператора '='\n";

cout << "7 - выход\n";

cout << "\nВЫБЕРИТЕ ДЕЙСТВИЕ >>> ";

cin >> choice;

cout << endl;

switch (choice) {

case 1:

container->containerOutput();

break;

case 2:

cout << "\nВведите индекс элемента для вывода >>> ";

int index;

cin >> index;

try {

if (index < 0 || index >= container->getSize()) {

throw(index);

}

container->operator[](index)->infoOutput();

}

catch (int) {

cout << "\nИндекс " << index << " выходит за пределы массива\n";

}

break;

case 3:

cout << "\nВведите индекс элемента для удаления >>> ";

int delIndex;

cin >> delIndex;

try {

if (delIndex < 0 || delIndex >= container->getSize()) {

throw(delIndex);

}

if (delIndex == container->getSize() - 1) {

container->delete\_back();

}

else {

container->index\_delete(delIndex);

}

}

catch (int) {

cout << "\nИндекс " << delIndex << " выходит за пределы массива\n";

}

break;

case 4:

cout << "\nВведите индекс элемента для вставки >>> ";

int insertIndex;

cin >> insertIndex;

try {

if (insertIndex < 0 || insertIndex > container->getSize()) {

throw(insertIndex);

}

cout << "1 - КУБ\n";

cout << "2 - ЦИЛИНДР\n";

cout << "3 - ТЕТРАЕДР\n";

cout << "\nВыберите фигуру >>> ";

int figure;

cin >> figure;

switch (figure) {

case 1:

cout << "\nВведите длину ребра куба >>> ";

float edgeLength;

cin >> edgeLength;

if (insertIndex == container->getSize()) {

container->push\_back(new Cube(edgeLength));

}

else {

container->insert(new Cube(edgeLength), insertIndex);

}

break;

case 2:

cout << "\nВведите высоту цилиндра >>> ";

float height;

cin >> height;

cout << "\nВведите радиус основания цилиндра >>> ";

float baseRadius;

cin >> baseRadius;

if (insertIndex == container->getSize()) {

container->push\_back(new Cylinder(height, baseRadius));

}

else {

container->insert(new Cylinder(height, baseRadius), insertIndex);

}

break;

case 3:

cout << "\nВведите длину ребра тетраедра >>> ";

float edgeLengthTetrahedron;

cin >> edgeLengthTetrahedron;

if (insertIndex == container->getSize()) {

container->push\_back(new Tetrahedron(edgeLengthTetrahedron));

}

else {

container->insert(new Tetrahedron(edgeLengthTetrahedron), insertIndex);

}

break;

default:

cout << "\n НЕВЕРНЫЙ ВВОД!!! \n";

break;

}

}

catch (int) {

cout << "\nИндекс " << insertIndex << " выходит за пределы массива\n";

}

break;

case 5:

cout << "\nВведите длину ребра 1 куба >>> ";

int cubeEdgeLength1;

cin >> cubeEdgeLength1;

cout << "\nВведите длину ребра 2 куба >>> ";

int cubeEdgeLength2;

cin >> cubeEdgeLength2;

cube1 = new Cube(cubeEdgeLength1);

cube2 = new Cube(cubeEdgeLength2);

if (cube1->operator==(\*cube2)) {

cout << "\nКубы равны\n";

}

else {

cout << "\nКубы не равны\n";

}

break;

case 6:

cout << "\nВведите длину ребра куба >>> ";

int len;

cin >> len;

cube1 = new Cube(len);

cube2 = new Cube();

cube2->operator=(\*cube1);

cout << "\nПервый куб:\n" << cube1;

cube1->infoOutput();

cout << "\nВторой куб:\n" << cube2;

cube2->infoOutput();

break;

case 7:

break;

default:

break;

}

} while (choice != 7);

}

Source.cpp:

#include "Cube.h"

#include "Cylinder.h"

#include "Tetrahedron.h"

#include "Container.h"

#include "menu.h"

#include <iostream>

using namespace std;

int main() {

setlocale(0, "");

cout << endl << "ЗАДАНИЕ --- ВАРИАНТ 10" << endl << endl;

cout << "Написать программу, в которой описана иерархия классов: геометрические фигуры\n";

cout << "(куб, цилиндр, тетраэдр).Описать класс для хранения коллекции фигур(массива указателей\n";

cout << "на базовый класс), в котором перегрузить операцию «[]», а также реализовать функции\n";

cout << "подсчёта общего объема и площади поверхности.Для базового класса и его потомков\n";

cout << "перегрузить операции « == », « != », « = ».Продемонстрировать работу операторов\n";

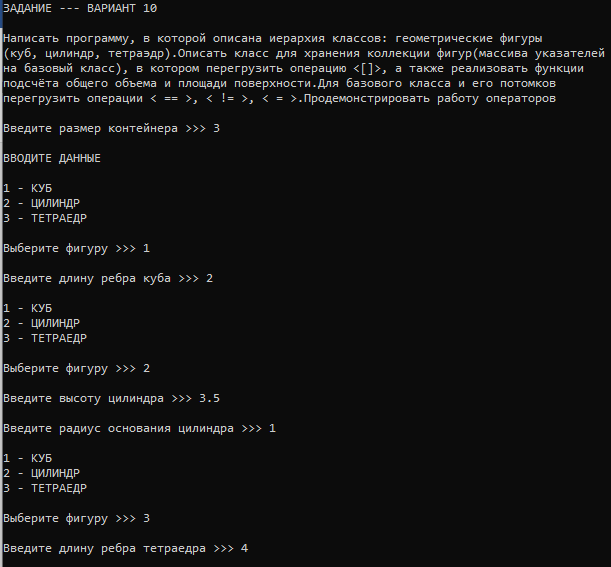
menu();

return 0;

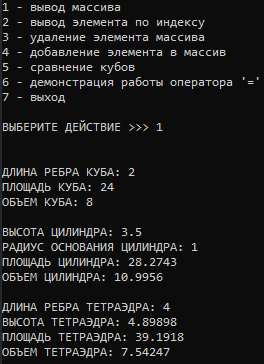
}

Результат выполнения:

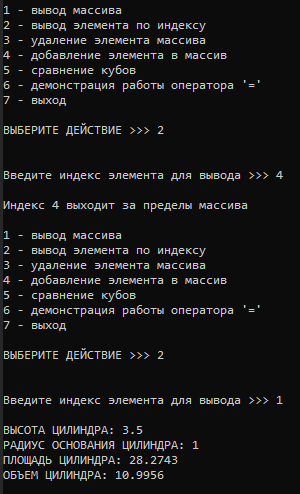
Ввод контейнера:



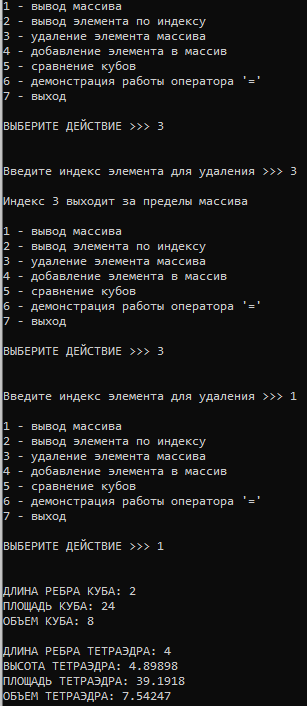
Вывод контейнера:



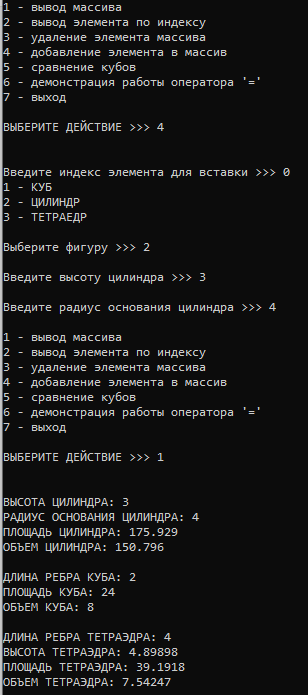
Вывод элемента по индексу:



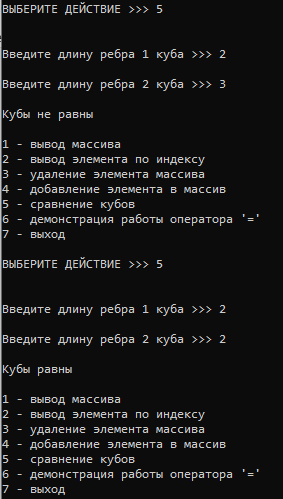
Удаление элемента по индексу:



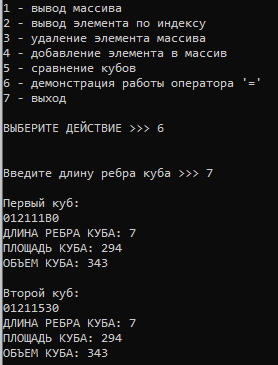
Вставка элемента:



Сравнение фигур:



Демонстрация работы оператора ‘=’:



Вывод: В ходе выполнения данной работы изучил особенности перегрузки операторов и обработки ошибок в С++.