Министерство образования Республики Беларусь

Учреждение образования

Брестский государственный технический университет

Кафедра ИИТ

Лабораторная работа №3

За 3 семестр

По дисциплине «Языки программирования»

Тема: «Перегрузка операций. Исключения»

Выполнил: студент 2 курса

Группы ПО-4(2)

Коташевич С.Н.

Проверил: Хацкевич М.В.

Брест 2020

**Лабораторная работа №3**

**Перегрузка операций. Исключения**

**Цель работы:** изучение правил перегрузки операций и принципов обработки исключений в С++

**Вариант 11**

Общие требования: в начале программы вывести задание; в процессе работы выводить подсказки пользователю (что ему нужно ввести, чтобы продолжить выполнение программы). Иерархию классов следует взять из лабораторной работы №3.

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 динамическая память должна быть освобождена.

**Задание:**

Написать программу, в которой описана иерархия классов: геометрические фигуры (**конус, шар, пирамида**). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[]». Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

**Текст программы:**

**Файл Geometric\_figures.h**

#ifndef GEOMETRIC\_FIGURES\_H

#define GEOMETRIC\_FIGURES\_H

#include <iostream>

#define pi 3.14

using namespace std;

class geometric\_figures

{

public:

virtual void Print() = 0;

virtual void Read() = 0;

virtual double Volume() = 0;

virtual double Square() = 0;

};

#endif // !GEOMETRIC\_FIGURES\_H

**Файл Ball.h**

#ifndef BALL\_H

#define BALL\_H

#include "Geometric\_figures.h"

class Ball : public geometric\_figures

{

private:

double R; //радиус

public:

Ball();

Ball(double R);

Ball(const Ball& smth);

void Print();

void Read();

double Volume();

double Square();

bool operator == (const Ball& other);

bool operator != (const Ball& other);

Ball& operator = (const Ball& other);

~Ball();

};

#endif // !BALL\_H

**Файл Ball.cpp**

#include "Ball.h"

Ball::Ball() : R(1)

{}

Ball::Ball(double Rad) : R(Rad)

{}

Ball::Ball(const Ball& smth): R(smth.R)

{}

void Ball::Print()

{

cout << "\nШар:\n"

<< "R = " << R << endl

<< "Площадь поверхности = " << Square() << endl

<< "Объём = " << Volume() << endl;

}

void Ball::Read()

{

cout << "Введите радиус шара: ";

cin >> R;

}

double Ball::Square()

{

return 4.0 \* pi \* pow(R, 2);

}

double Ball::Volume()

{

return 4.0 / 3.0 \* pi \* pow(R, 3);

}

bool Ball::operator==(const Ball& other)

{

return \*this == other;

}

bool Ball::operator!=(const Ball& other)

{

return !(\*this == other);

}

Ball& Ball::operator=(const Ball& other)

{

return \*this = other;

}

Ball::~Ball()

{}

**Файл Cone.h**

#ifndef CONE\_H

#define CONE\_H

#include "Geometric\_figures.h"

class Cone : public geometric\_figures

{

private:

double R; //радиус основания

double L; //образующая конуса

double H; //высота конуса

public:

Cone();

Cone(double R, double L, double H);

Cone(const Cone& smth);

void Print();

void Read();

double Volume();

double Square();

bool operator == (const Cone& other);

bool operator != (const Cone& other);

Cone& operator = (const Cone& other);

~Cone();

};

#endif // !CONE\_H

**Файл Cone.cpp**

#include "Cone.h"

Cone::Cone() : R(1), L(1), H(1)

{}

Cone::Cone(double Rad, double Len, double Hig) : R(Rad), L(Len), H(Hig)

{}

Cone::Cone(const Cone& smth) : R(smth.R), L(smth.L), H(smth.H)

{}

void Cone::Print()

{

cout << "\nКонус:\n"

<< "R = " << R << endl

<< "L = " << L << endl

<< "H = " << H << endl

<< "Площадь поверхности = " << Square() << endl

<< "Объём = " << Volume() << endl;

}

void Cone::Read()

{

cout << "Введите радиус основания конуса: ";

cin >> R;

cout << "Введите образующую конуса: ";

cin >> L;

cout << "Введите высоту конуса: ";

cin >> H;

}

double Cone::Square()

{

return pi \* R \* (R + L);

}

double Cone::Volume()

{

return 1.0 / 3.0 \* pi \* pow(R, 2) \* H;

}

bool Cone::operator==(const Cone& other)

{

return \*this == other;

}

bool Cone::operator!=(const Cone& other)

{

return !(\*this == other);

}

Cone& Cone::operator=(const Cone& other)

{

return \*this = other;

}

Cone::~Cone()

{}

**Файл Pyramid.h**

#ifndef PYRAMID\_H

#define PYRAMID\_H

#include "Geometric\_figures.h"

//Предположим, что в основаниии квадрат

class Pyramid : public geometric\_figures

{

private:

double H; //высота пирамиды

double h; //апофема

double A; //сторона основания

public:

Pyramid();

Pyramid(double A, double H, double h);

Pyramid(const Pyramid& smth);

void Print();

void Read();

double Volume();

double Square();

bool operator == (const Pyramid& other);

bool operator != (const Pyramid& other);

Pyramid& operator = (const Pyramid& other);

~Pyramid();

};

#endif // !PYRAMID\_H

**Файл Pyramid.cpp**

#include "Pyramid.h"

Pyramid::Pyramid() : A(1), H(1), h(1)

{}

Pyramid::Pyramid(double Hih, double Len, double Apo) : A(Len), H(Hih),h(Apo)

{}

Pyramid::Pyramid(const Pyramid& smth) : A(smth.A), H(smth.H), h(smth.h)

{}

void Pyramid::Print()

{

cout << "\nПирамида:\n"

<< "A = " << A << endl

<< "H = " << H << endl

<< "h = " << h << endl

<< "Площадь поверхности = " << Square() << endl

<< "Объём = " << Volume() << endl;

}

void Pyramid::Read()

{

cout << "Введите сторону основания пирамиды: ";

cin >> A;

cout << "Введите высоту пирамиды: ";

cin >> H;

cout << "Введите образующую пирамиды: ";

cin >> h;

}

double Pyramid::Square()

{

return pow(A, 2) + 1.0 / 2.0 \* 4.0 \* A \* h;

}

double Pyramid::Volume()

{

return 1.0 / 3.0 \* pow(A, 2) \* H;

}

bool Pyramid::operator==(const Pyramid& other)

{

return \*this == other;

}

bool Pyramid::operator!=(const Pyramid& other)

{

return !(\*this == other);

}

Pyramid& Pyramid::operator=(const Pyramid& other)

{

return \*this = other;

}

Pyramid::~Pyramid()

{}

**Файл Collection\_figures.h**

#ifndef COLECTION\_FIGURES\_H

#define COLECTION\_FUGURES\_H

#include "Geometric\_figures.h"

#include "error.h"

class Colection

{

private:

geometric\_figures\*\* fig;

int c\_count;

public:

Colection(int count);

geometric\_figures\* operator[] (int n) const;

geometric\_figures\*& operator[] (int n);

int get\_count();

void add\_end(geometric\_figures\* figure);

void add\_index(geometric\_figures\* figure, int index);

void del\_end();

void del\_index(int index);

~Colection();

};

#endif // !COLECTION\_FIGURES\_H

**Файл Geometric\_figures.cpp**

#include "Colection\_figures.h"

Colection::Colection(int count)

{

fig = new geometric\_figures\*[count];

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

fig[i] = NULL;

}

c\_count = count;

}

geometric\_figures\* Colection::operator[] (int n) const

{

if (n < 0 || n >= c\_count) {

throw Index\_Error("Вы вышли за границы массива");

}

return fig[n];

}

geometric\_figures\*& Colection::operator[] (int n)

{

if (n < 0 || n >= c\_count) {

throw Index\_Error("Вы вышли за границы массива");

}

return fig[n];

}

int Colection::get\_count()

{

return c\_count;

}

void Colection::add\_end(geometric\_figures\* figure)

{

geometric\_figures\*\* temp = new geometric\_figures \* [c\_count + 1];

for (int i = 0; i < c\_count; i++) {

temp[i] = fig[i];

}

fig = temp;

temp[c\_count] = figure;

c\_count++;

}

void Colection::add\_index(geometric\_figures\* figure, int index)

{

if (index < 0 || index > c\_count) {

add\_end(figure);

throw Index\_Error("Вы вышли за границы массива, фигура будет добавлена в конец массива");

}

geometric\_figures\*\* temp = new geometric\_figures \* [c\_count + 1];

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

temp[i] = fig[i];

}

temp[index] = figure;

for (int i = index; i < c\_count; i++) {

temp[i + 1] = fig[i];

}

fig = temp;

c\_count++;

}

void Colection::del\_end()

{

geometric\_figures\*\* temp = new geometric\_figures \* [c\_count - 1];

for (int i = 0; i < c\_count - 1; i++) {

temp[i] = fig[i];

}

fig = temp;

c\_count--;

}

void Colection::del\_index(int index)

{

geometric\_figures\*\* temp = new geometric\_figures \* [c\_count - 1];

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

temp[i] = fig[i];

}

for (int i = index + 1; i < c\_count; i++)

{

temp[i - 1] = fig[i];

}

fig = temp;

c\_count--;

}

Colection::~Colection()

{

for (int i = 0; i < c\_count; i++) {

if (fig[i] != nullptr) {

delete fig[i];

}

}

}

**Файл error.h**

#ifndef ERROR\_H

#define ERROR\_H

#include <iostream>

class Index\_Error

{

private:

char\* message;

public:

Index\_Error(const char\* message);

~Index\_Error();

char\* get\_message();

};

#endif // !ERROR\_H

**Файл error.cpp**

#include "error.h"

Index\_Error::Index\_Error(const char\* message)

{

this->message = \_strdup(message);

}

Index\_Error::~Index\_Error()

{

delete this->message;

}

char\* Index\_Error::get\_message()

{

return message;

}

**Файл mian.cpp**

#include "Geometric\_figures.h"

#include "Colection\_figures.h"

#include "Cone.h"

#include "Ball.h"

#include "Pyramid.h"

using namespace std;

void con(Colection& colection);

void ball(Colection& colection);

void pyr(Colection& colection);

void cmp();

void show(Colection& colection);

void del(Colection& colection);

void del\_in(Colection& colection);

int main()

{

setlocale(LC\_ALL, "rus");

int choise;

Colection colection(0);

while (1)

{

cout <<

"Написать программу, в которой...... ";

cout << "\nВыберите действие:\n"

<< "1. создать конус\n"

<< "2. создать шар\n"

<< "3. создать пирамиду\n"

<< "4. сравнить 2 шара\n"

<< "5. вывести все объекты\n"

<< "6. удалить объект с конца\n"

<< "7. удалить объект по индексу\n"

<< "0 exit\n";

try {

cin >> choise;

switch (choise)

{

case 1:

con(colection);

break;

case 2:

ball(colection);

break;

case 3:

pyr(colection);

break;

case 4:

cmp();

break;

case 5:

show(colection);

break;

case 6:

del(colection);

break;

case 7:

del\_in(colection);

default:

return 0;

break;

}

}

catch (Index\_Error& e) {

cout << "Ошибка индекса: " << e.get\_message() << endl;

}

catch (...)

{

cout << "Другая ошибка." << endl;

}

}

return 0;

}

void con(Colection& colection)

{

Cone \*c1 = new Cone;

c1->Read();

c1->Print();

cout << "добавить по индексу(1) или в конец(0)?\n";

bool choise = false;

cin >> choise;

if (choise) {

int index;

cout << "Введите индекс: ";

cin >> index;

colection.add\_index(c1, index);

}

else {

colection.add\_end(c1);

}

}

void ball(Colection& colection)

{

Ball \*b1 = new Ball(7);

b1->Print();

cout << "добавить по индексу(1) или в конец(0)?\n";

bool choise = false;

cin >> choise;

if (choise) {

int index;

cout << "Введите индекс: ";

cin >> index;

colection.add\_index(b1, index);

}

else {

colection.add\_end(b1);

}

}

void pyr(Colection& colection)

{

Pyramid\* p1 = new Pyramid;

p1->Print();

colection.add\_end(p1);

cout << "добавить по индексу(1) или в конец(0)?\n";

bool choise = false;

cin >> choise;

if (choise) {

int index;

cout << "Введите индекс: ";

cin >> index;

colection.add\_index(p1, index);

}

else {

colection.add\_end(p1);

}

}

void cmp()

{

Ball\* b1, \* b2;

b1 = new Ball;

b1->Read();

b2 = b1;

b2->Print();

if (b2 == b1) {

cout << "Они равны\n";

}

}

void show(Colection& colection)

{

for (int i = 0; i < colection.get\_count(); i++) {

colection[i]->Print();

}

}

void del(Colection& colection)

{

colection.del\_end();

cout << "Объект удалён\n";

}

void del\_in(Colection& colection)

{

int index;

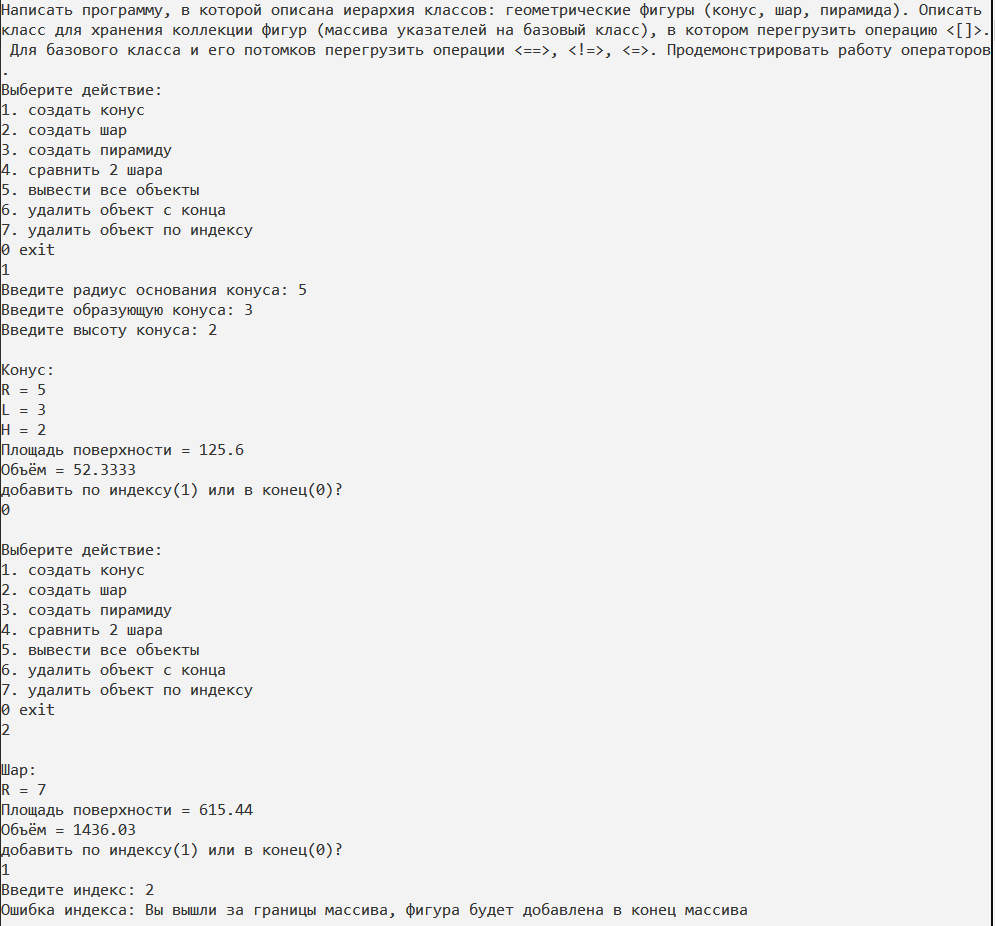
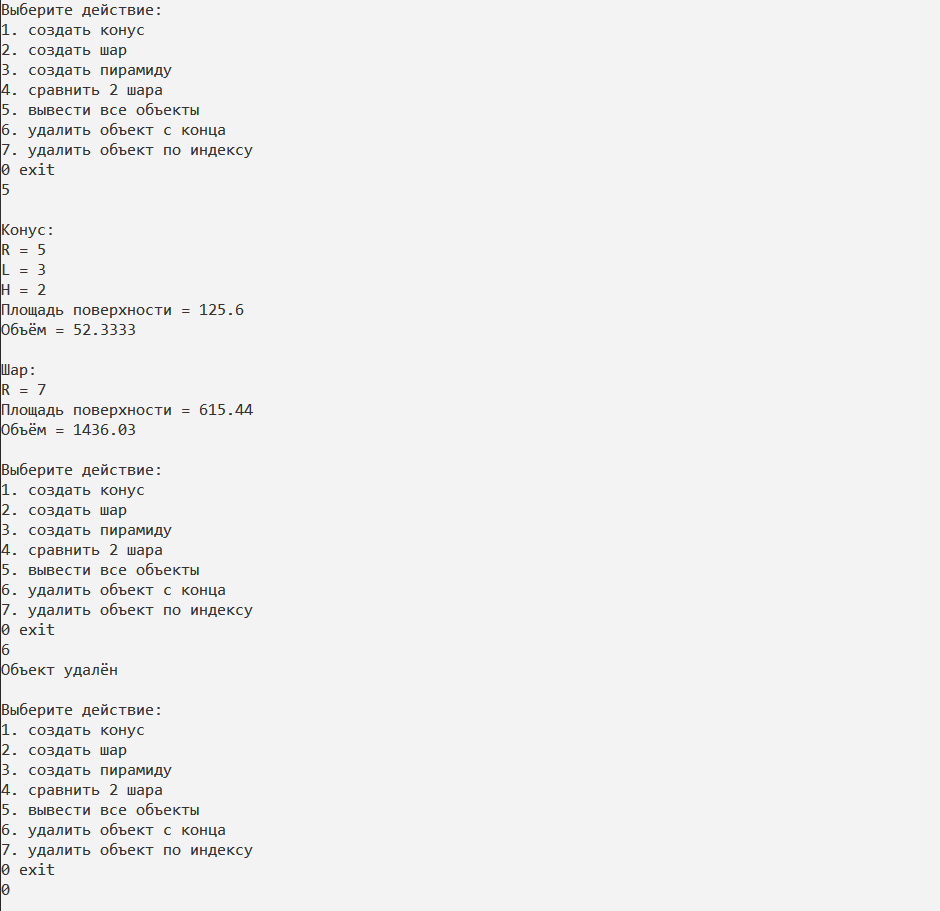
cout << "Введите индекс: ";

cin >> index;

colection.del\_index(index);

}

**Пример работы программы:**

**Вывод:** В ходе работы научились использовать перегрузки операторов и обучлся принципам обработки исключения.