Министерство образования Республики Беларусь

УО «Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №3**

По дисциплине: “Языки программирования”

Тема: “Перегрузка операций. Исключения”**Вариант №11**

**Выполнил**: студент 2 курса группы ПО-7 Лобан К.Ю.

**Проверила:** Дряпко А. В.

Брест 2021

### Цель работы

Изучение правил перегрузки операций и принципов обработки исключений в C++

### 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 динамически выделенная память должна быть освобождена.

### Задание

11. 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 <<[ ]>>, а также реализовать функции подсчета общего объема и площади поверхности. Для базового класса и его потомков перегрузить операции <==>, <!=>, <=>. Продемонстрировать работу операторов.

### Код программы

# Figure.h

#pragma once

#include <iostream>

#include <math.h>

using namespace std;

class Figure

{

public:

virtual double V() = 0;

virtual double S() = 0;

virtual void Show() = 0;

void ShowVS();

bool operator==(Figure& figure);

bool operator!=(Figure& figure);

};

class Sphere : public Figure

{

const double pi = 3.1415;

double r;

public:

Sphere(double r);

double V() override;

double S() override;

void Show() override;

Sphere& operator=(Sphere& sphere);

};

class Pyramid : public Figure

{

double po;

protected:

double h;

double l;

double so;

public:

Pyramid(double h, double l, double po, double so);

double V() override;

double S() override;

void Show() override;

Pyramid& operator=(Pyramid& pyramid);

};

class Cone : public Pyramid

{

const double pi = 3.1415;

double r;

public:

Cone(double r, double h, double l, double so);

double S() override;

void Show() override;

Cone& operator=(Cone& cone);

};

class FiguresArray {

Figure\*\* figures;

int count;

public:

FiguresArray();

void AddBack(Figure\* figure);

void Show();

Figure\* operator[](int i);

};

**Cone.cpp**

#include "figure.h"

Cone::Cone(double r, double h, double l, double so) : r(r), Pyramid(h, l, pi\* r \* 2, so) {}

double Cone::S() {

return so + pi \* r \* l;

}

void Cone::Show() {

cout << "Cone. R = " << r << " H = " << h << " So = " << so << endl;

}

Cone& Cone::operator=(Cone& cone) {

this->r = cone.r;

this->h = cone.h;

this->l = cone.l;

this->so = cone.so;

return \*this;

}

**Figure.cpp**

#include "figure.h"

void Figure::ShowVS()

{

cout << "V = " << V() << " S = " << S() << endl;

}

bool Figure::operator==(Figure& figure)

{

if (this->S() == figure.S() && this->V() == figure.V())

return true;

else

return false;

}

bool Figure::operator!=(Figure& figure)

{

return !(\*this == figure);

}

**Pyramid.cpp**

#include "figure.h"

Pyramid::Pyramid(double h, double l, double po, double so) : h(h), l(l), po(po), so(so) {}

double Pyramid::V() {

return (so \* h) / 3;

}

double Pyramid::S() {

return (l \* po) / 2 + so;

}

void Pyramid::Show() {

cout << "Pyramid. H = " << h << " L = " << l << " Po = " << po << " So = " << so << endl;

}

Pyramid& Pyramid::operator=(Pyramid& pyramid)

{

this->h = pyramid.h;

this->l = pyramid.l;

this->so = pyramid.so;

this->po = pyramid.po;

return \*this;

}

**Sphere.cpp**

#include "figure.h"

Sphere::Sphere(double r) : r(r) {}

double Sphere::V(){

return (4 \* r \* r \* r \* pi) / 3;

}

double Sphere::S(){

return 4 \* pi \* r \* r;

}

void Sphere::Show(){

cout << "Sphere. R = " << r << endl;

}

Sphere& Sphere::operator=(Sphere& sphere) {

this->r = sphere.r;

return \*this;

}

**figuresArray**

#include "figure.h"

FiguresArray::FiguresArray() : figures(nullptr), count(0) {}

void FiguresArray::AddBack(Figure\* figure) {

count++;

Figure\*\* tempFigures = figures;

figures = new Figure \* [count];

for (int i = 0; i < count - 1; i++) {

figures[i] = tempFigures[i];

}

figures[count - 1] = figure;

delete[] tempFigures;

}

void FiguresArray::Show() {

cout << "Figures:" << endl;

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

figures[i]->Show();

}

}

Figure\* FiguresArray::operator[](int i)

{

if (i < 0 || i >= count)

throw - 1;

return figures[i];

}

**Sourse.spp**

#include "figure.h"

int main()

{

Sphere sphere(3.4);

sphere.Show();

sphere.ShowVS();

Sphere sphere2(5);

sphere2.Show();

sphere2.ShowVS();

Sphere sphere3(3.4);

sphere3.Show();

sphere3.ShowVS();

if (sphere == sphere2)

cout << "sphere == sphere2" << endl;

else

cout << "sphere != sphere2" << endl;

if (sphere != sphere3)

cout << "sphere != sphere3" << endl;

else

cout << "sphere == sphere3" << endl;

sphere = sphere2;

if (sphere == sphere2)

cout << "sphere == sphere2" << endl;

else

cout << "sphere != sphere2" << endl;

Pyramid pyramid(20,8,20,60);

pyramid.Show();

pyramid.ShowVS();

Cone cone(3,20, 28.2735, 60);

cone.Show();

cone.ShowVS();

FiguresArray array;

array.AddBack(&sphere);

array.AddBack(&pyramid);

array.AddBack(&cone);

array.Show();

try

{

cout << "Items" << endl;

for(int i = 0; i < 5;i++)

array[i]->Show();

}

catch(int i)

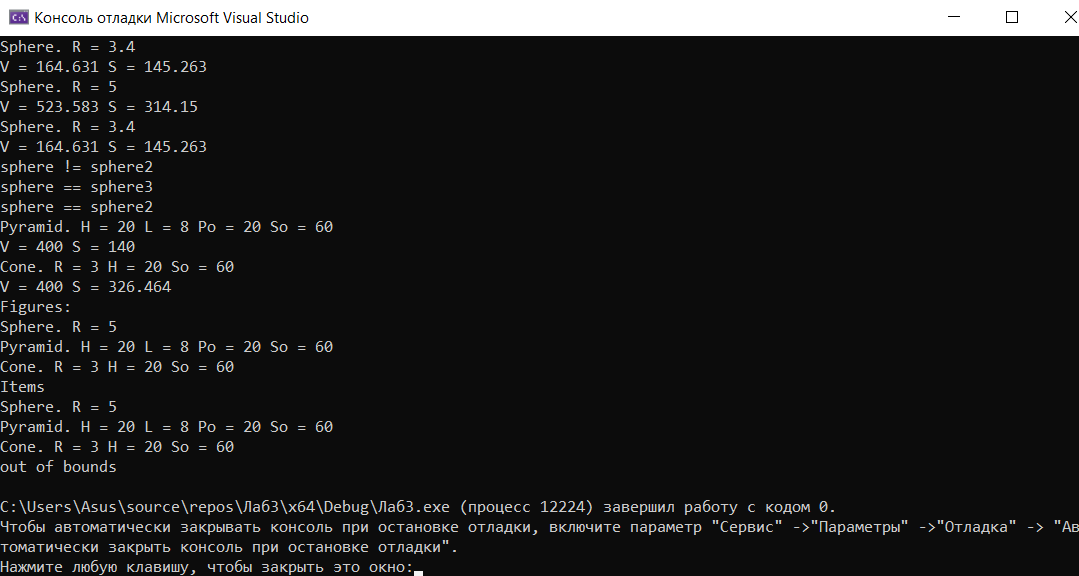
{

cout << "out of bounds" << endl;

}

}

# Результаты программы



### Вывод

Я изучил правила перегрузки операций и принципов обработки исключений в C++