Министерство образования Республики Беларусь

УО «Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №3**

По дисциплине: “Языки программирования”

Тема: “Перегрузка операций. Исключения”  
**Вариант №6**

**Выполнил**:

студент группы ПО-7

Рекун И.В.

**Проверил:** Бойко Д.О.

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: 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.

# Код программы

**main.cpp**

#include "header.h"

int main() {

Tetrahedron a(2);

Cube b(2);

Cylinder c(7, 8);

bool check1 = a == b;

bool check2 = a != c;

std::cout << check1 << std::endl;

std::cout << check2 << std::endl;

Arrayptr ptr;

ptr.add(&a);

ptr.add(&b);

ptr.add(&c);

ptr.print();

\*ptr[0] = \*ptr[1];

std::cout << std::endl << "After =" << std::endl;

ptr.print();

}

**source.cpp**

#include <math.h>

#include "header.h"

double Geometrical\_figure::getV() const {

return V;

}

double Geometrical\_figure::getS() const {

return S;

}

///Cube

void Cube::Show() {

std::cout << "Cube V = " << V << std::endl;

std::cout << "Cube S = " << S << std::endl;

}

Cube::Cube(double new\_a) :a(new\_a) {

V = pow(new\_a, 3);

S = 6 \* pow(new\_a, 2);

}

void Cylinder::Show() {

std::cout << "Cylinder V = " << V << std::endl;

std::cout << "Cylinder S = " << S << std::endl;

}

Cylinder::Cylinder(double new\_r, double new\_h) :r(new\_r), h(new\_h) {

V = 3.14 \* pow(new\_r, 2) \* new\_h;

S = (2 \* 3.14 \* new\_h \* new\_r) + (2 \* 3.14 \* pow(new\_r, 2));

}

void Tetrahedron::Show() {

std::cout << "Tetrahedron V = " << V << std::endl;

std::cout << "Tetrahedron S = " << S << std::endl;

}

Tetrahedron::Tetrahedron(double new\_a) :\_a(new\_a) {

V = (pow(new\_a, 3) \* sqrt(2)) / 12;

S = pow(new\_a, 2) \* sqrt(3);

}

void Arrayptr::add(Geometrical\_figure\* figure) {

if (size == 0) {

size++;

ptrarr = new Geometrical\_figure \* [size];

ptrarr[0] = figure;

}

else {

size++;

Geometrical\_figure\*\* temp;

temp = ptrarr;

ptrarr = new Geometrical\_figure \* [size];

for (size\_t i = 0; i < size; i++) {

ptrarr[i] = temp[i];

}

ptrarr[size - 1] = figure;

}

}

void Arrayptr::print() {

for (size\_t i = 0; i < size; i++) {

std::cout << "-------------" << std::endl;

ptrarr[i]->Show();

}

}

Geometrical\_figure\* Arrayptr::operator[](int index) {

return(ptrarr[index]);

}

**Header.h**

#ifndef LAB3\_HEADER\_H

#define LAB3\_HEADER\_H

#include <iostream>

class Geometrical\_figure {

protected:

double V;

double S;

public:

double getV() const;

double getS() const;

Geometrical\_figure& operator = (const Geometrical\_figure& right) {

if (this == &right) {

return \*this;

}

V = right.V;

S = right.S;

return \*this;

}

bool operator == (const Geometrical\_figure& right) {

return (V == right.V) && (S == right.S);

}

bool operator != (const Geometrical\_figure& right) {

return !((V == right.V) && (S == right.S));

}

virtual void Show() = 0;

};

class Cube :public Geometrical\_figure {

private:

double a;

public:

Cube(double new\_a);//constructor with parameters

void Show() override;

};

class Cylinder :public Geometrical\_figure {

private:

double r, h;

public:

Cylinder(double new\_r, double new\_h);//constructor

void Show() override;

};

class Tetrahedron :public Geometrical\_figure {

private:

double \_a;

public:

Tetrahedron(double new\_a);

void Show() override;

};

class Arrayptr {

private:

Geometrical\_figure\*\* ptrarr;

int size = 0;

public:

void add(Geometrical\_figure\* figure);

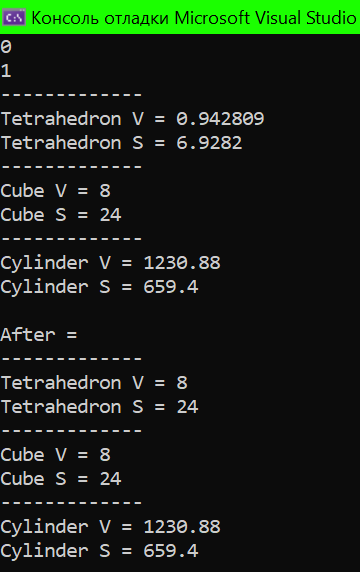
void print();

Geometrical\_figure\* operator [](int index);

};

#endif //LAB3\_HEADER\_H

# Результаты программы



### Вывод: я изучил правила перегрузки операций и принципов обработки исключений в C++.