Министерство образования Республики Беларусь

УО «Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №3**

По дисциплине: “Языки программирования”

Тема: “Перегрузка операций. Исключения”**Вариант №9**

**Выполнил**: студент 2 курса группы ПО-7 Крупенков Михаил Дмитриевич

**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.

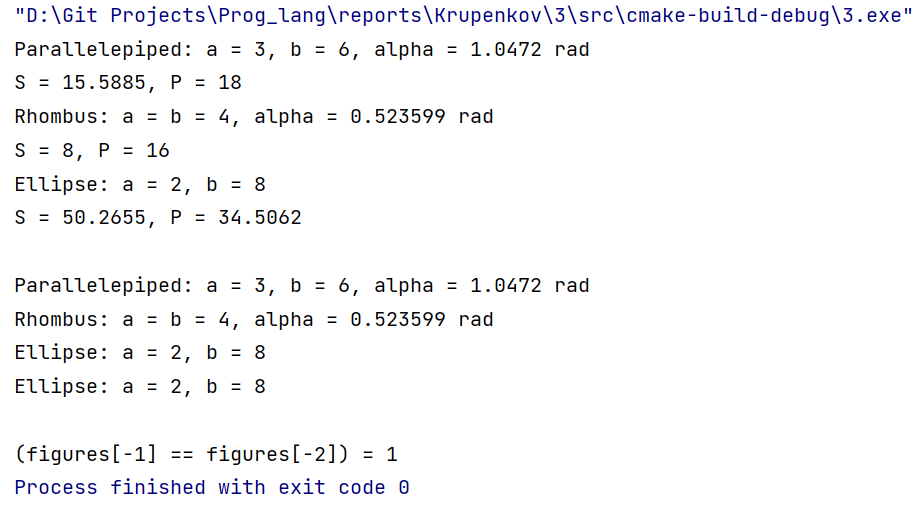
### Задание

**9.** Написать программу, в которой описана иерархия классов: геометрические фигуры (ромб, параллелепипед, эллипс). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общей площади и периметра. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

# Код программы

#include **<iostream>**#include **<cmath>  
  
using namespace** std;  
**const float** PI = 3.141592653589793;  
  
  
*// Фигуры:  
// Ромб, параллелепипед, эллипс***class** Figure {  
**protected**:  
 **float** a, b;  
  
**public**:  
 Figure(**float** a, **float** b) : a(a), b(b) {}  
  
 **virtual float** s() **const** = 0;  
  
 **virtual float** p() **const** = 0;  
  
 **virtual void** print() **const** = 0;  
  
 **virtual void** printSP() **const final** {  
 cout << **"S = "** << s() << **", P = "** << p() << **"\n"**;  
 }  
  
 **bool operator**==(**const** Figure &other) {  
 **return** (**typeid**(\***this**) == **typeid**(other)) && (a == other.a) && (b == other.b) && (s() == other.s());  
 }  
  
 **bool operator**!=(**const** Figure &other) {  
 **return** !(\***this** == other);  
 }  
  
 Figure &**operator**=(**const** Figure &other) {  
 a = other.a;  
 b = other.b;  
 **return** \***this**;  
 }  
};  
  
  
**class** Parallelepiped : **public** Figure {  
**protected**:  
 **float** alpha;  
  
**public**:  
 Parallelepiped(**float** a, **float** b, **float** alpha) : Figure(a, b), alpha(alpha) {}  
  
 **float** s() **const override** {  
 **return** a \* b \* sin(alpha);  
 }  
  
 **float** p() **const override** {  
 **return** (a + b) \* 2;  
 }  
  
 **void** print() **const override** {  
 cout << **"Parallelepiped: a = "** << a << **", b = "** << b << **", alpha = "** << alpha << **" rad\n"**;  
 }  
  
  
};  
  
  
**class** Rhombus : **public** Parallelepiped {  
**public**:  
 Rhombus(**float** a, **float** alpha) : Parallelepiped(a, a, alpha) {}  
  
 **void** print() **const override** {  
 cout << **"Rhombus: a = b = "** << a << **", alpha = "** << alpha << **" rad\n"**;  
 }  
};  
  
  
**class** Ellipse : **public** Figure {  
**public**:  
 Ellipse(**float** a, **float** b) : Figure(a, b) {}  
  
 **float** s() **const override** {  
 **return** PI \* a \* b;  
 }  
  
 **float** p() **const override** {  
 **return float**(4 \* (PI \* a \* b + pow((a - b), 2)) / (a + b));  
 }  
  
 **void** print() **const override** {  
 cout << **"Ellipse: a = "** << a << **", b = "** << b << **"\n"**;  
 }  
};  
  
  
**class** FiguresPtrArray {  
 Figure \*\*ptrArray; *// Массив указателей на фигуры* **int** count; *// Количество элементов в массиве указателей***public**:  
 FiguresPtrArray() : ptrArray(**nullptr**), count(0) {}  
  
 **int** len() **const** {  
 **return** count;  
 }  
  
 **void** addBack(Figure \*figure) {  
 **if** (count == 0) {  
 count = 1;  
 ptrArray = **new** Figure \*;  
 ptrArray[0] = figure;  
 } **else** {  
 count++;  
 **auto** temp = ptrArray;  
 ptrArray = **new** Figure \*[count];  
 **for** (**int** i = 0; i < count; i++) {  
 ptrArray[i] = temp[i];  
 }  
 ptrArray[count - 1] = figure;  
 **delete**[] temp;  
 }  
 }  
  
 **void** print() **const** {  
 **for** (**int** i = 0; i < count; i++) {  
 ptrArray[i]->print();  
 }  
 }  
  
 Figure \***operator**[](**int** i) {  
 **return** (i >= 0) ? ptrArray[i] : ptrArray[count + i];  
 }  
};  
  
  
**int** main() {  
 Parallelepiped parallelepiped(3, 6, PI / 3);  
 Rhombus rhombus(4, PI / 6);  
 Ellipse ellipse(2, 8);  
  
 FiguresPtrArray figures;  
 figures.addBack(&parallelepiped);  
 figures.addBack(&rhombus);  
 figures.addBack(&ellipse);  
  
 **for** (**int** i = 0; i < figures.len(); i++) {  
 figures[i]->print();  
 figures[i]->printSP();  
 }  
  
 cout << endl;  
 figures.addBack(figures[-1]);  
 figures.print();  
 cout << **"\n(figures[-1] == figures[-2]) = "** << (figures[-1] == figures[-2]);  
}

# Результаты программы



### Вывод

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