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-6(1)

Мартынович Д. М.

Проверил:

Хацкевич М. В.

2021

Лабораторная работа №3

Перегрузка операций. Исключения

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

Вариант 19

Задание:

Написать программу, в которой описана иерархия классов: геометрические фигуры (круг, параллелепипед, трапеция). 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.

GeometricFigure.h

#pragma once

#include <iostream>

using namespace std;

class geometricFigures {

public:

virtual void Print() = 0;

virtual void Read() = 0;

virtual double Perimeter() = 0;

virtual double Square() = 0;

};

Rhombus.h

#pragma once

#define \_USE\_MATH\_DEFINES

#include "GeometricFigure.h"

class Rhombus :public geometricFigures {

private:

int d1,d2;

public:

Rhombus();

Rhombus(int d1, int d2);

Rhombus(const Rhombus& other);

~Rhombus();

bool operator == (const Rhombus& right);

bool operator != (const Rhombus& right);

Rhombus& operator = (const Rhombus& right);

void Print();

void Read();

double Perimeter();

double Square();

};

Rhombus.cpp

#include "Rhombus.h"

Rhombus::Rhombus() {}

Rhombus::Rhombus(int d1, int d2) {

this->d1 = d1;

this->d2 = d2;

}

Rhombus::Rhombus(const Rhombus& other) {

d1 = other.d1;

d2 = other.d2;

}

Rhombus::~Rhombus() {}

bool Rhombus::operator == (const Rhombus& right) {

return (d1 == right.d1) &&(d2 == right.d2);

}

bool Rhombus::operator != (const Rhombus& right) {

return !(\*this == right);

}

Rhombus& Rhombus::operator = (const Rhombus & right) {

d1 = right.d1;

d2 = right.d2;

return \*this;

}

void Rhombus::Print() {

cout << endl << "Ромб: " << "d1=" << d1 <<" d2="<<d2<< endl;

cout << "Площадь: " << "S=" << Square() << endl;

cout << "Периметр: " << "P=" << Perimeter() << endl;

}

void Rhombus::Read() {

cout << endl << "Введите первую диагональ ромба: ";

cin >> d1;

cout << endl << "Введите вторую диагональ ромба: ";

cin >> d2;

}

double Rhombus::Perimeter() {

return 2\*sqrt(pow(d1,2)+pow(d2,2));

}

double Rhombus::Square() {

return (d1 \* d2) / 2;

}

Rectangle.h

#pragma once

#include "GeometricFigure.h"

class Rectangle :public geometricFigures {

private:

int a, b;

public:

Rectangle();

Rectangle(int a, int b);

Rectangle(const Rectangle& other);

~Rectangle();

bool operator == (const Rectangle& right);

bool operator != (const Rectangle& right);

Rectangle& operator = (const Rectangle& right);

void Print();

void Read();

double Perimeter();

double Square();

};

Rectangle.cpp

#include "Rectangle.h"

Rectangle::Rectangle() {}

Rectangle::Rectangle(int a, int b) {

this->a = a;

this->b = b;

}

Rectangle::Rectangle(const Rectangle& other) {

a = other.a;

b = other.b;

}

Rectangle::~Rectangle() {}

bool Rectangle::operator == (const Rectangle& right) {

return (a == right.a) && (b == right.b);

}

bool Rectangle::operator != (const Rectangle & right) {

return !(\*this == right);

}

Rectangle& Rectangle::operator = (const Rectangle & right) {

a = right.a;

b = right.b;

return \*this;

}

void Rectangle::Print() {

cout << endl << "Прямоугольник: " << "a=" << a << ", b=" << b << endl;

cout << "Площадь: " << "S=" << Square() << endl;

cout << "Периметр: " << "P=" << Perimeter() << endl;

}

void Rectangle::Read() {

cout << endl << "Введите высоту прямоугольника: ";

cin >> a;

cout << "Введите ширину прямоугольника: ";

cin >> b;

}

double Rectangle::Perimeter() {

return (a+b)\*2;

}

double Rectangle::Square() {

return a\*b;

}

Ellipse.h

#pragma once

#define \_USE\_MATH\_DEFINES

#include "GeometricFigure.h"

class Ellipse :public geometricFigures {

private:

int a, b;

public:

Ellipse();

Ellipse(int a, int b);

Ellipse(const Ellipse& other);

~Ellipse();

bool operator == (const Ellipse& right);

bool operator != (const Ellipse& right);

Ellipse& operator = (const Ellipse& right);

void Print();

void Read();

double Perimeter();

double Square();

};

Ellipse.cpp

#include "Ellipse.h"

Ellipse::Ellipse() {}

Ellipse::Ellipse(int a, int b) {

this->a = a;

this->b = b;

}

Ellipse::Ellipse(const Ellipse& other) {

a = other.a;

b = other.b;

}

Ellipse::~Ellipse() {}

bool Ellipse::operator == (const Ellipse& right) {

return (a == right.a) && (b == right.b);

}

bool Ellipse::operator != (const Ellipse & right) {

return !(\*this == right);

}

Ellipse& Ellipse::operator = (const Ellipse & right) {

a = right.a;

b = right.b;

return \*this;

}

void Ellipse::Print() {

cout << endl << "Эллипс: " << "a=" << a << ", b=" << b <<endl;

cout << "Площадь: " << "S=" << Square() << endl;

cout << "Периметр: " << "P=" << Perimeter() << endl;

}

void Ellipse::Read() {

cout << endl << "Введите длину большей полуоси: ";

cin >> a;

cout << "Введите длину меньшей полуоси: ";

cin >> b;

}

double Ellipse::Perimeter() {

return 4\*(M\_PI\*a\*b+(a-b))/a+b;

}

double Ellipse::Square() {

return M\_PI\*a\*b;

}

Error.h

#pragma once

#include <iostream>

class IndexError {

protected:

char\* message;

public:

IndexError(const char\* message);

~IndexError();

char\* get\_message();

};

Error.cpp

#include "error.h"

IndexError::IndexError(const char\* message) {

this->message = \_strdup(message);

}

IndexError::~IndexError() {

delete this->message;

}

char\* IndexError::get\_message() {

return message;

}

MassiveFigure.h

#pragma once

#include "GeometricFigure.h"

#include "error.h"

class MassiveFigures {

private:

geometricFigures\*\* arr;

int count;

public:

MassiveFigures(int count);

~MassiveFigures();

geometricFigures\* operator[] (int n) const;

geometricFigures\*& operator[] (int n);

int get\_count();

void addToTheEnd(geometricFigures\* figure);

void add(int index, geometricFigures\* figure);

void deleteFromTheEnd();

void del(int index);

};

MassiveFigures.cpp

#include "MassiveFigures.h"

MassiveFigures::MassiveFigures(int count) {

arr = new geometricFigures \* [count];

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

arr[i] = nullptr;

}

this->count = count;

}

MassiveFigures::~MassiveFigures() {

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

if (arr[i] != nullptr) {

delete arr[i];

}

delete[]arr;

}

}

geometricFigures\* MassiveFigures::operator[] (int n) const {

if (n < 0 || n >= count) {

throw IndexError("Вы вышли за границы массива.");

}

return arr[n];

}

geometricFigures\*& MassiveFigures::operator[] (int n) {

if (n < 0 || n >= count) {

throw IndexError("Вы вышли за границы массива.");

}

return arr[n];

}

int MassiveFigures::get\_count() {

return count;

}

void MassiveFigures::addToTheEnd(geometricFigures\* figure) {

geometricFigures\*\* temp = new geometricFigures \* [count + 1];

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

temp[i] = arr[i];

}

arr = temp;

temp[count] = figure;

count++;

}

void MassiveFigures::add(int index, geometricFigures\* figure) {

if (index < 0 || index > count) {

addToTheEnd(figure);

throw IndexError("Вы вышли за границы массива. Фигура будет добавлена в конец массива");

}

geometricFigures\*\* temp = new geometricFigures \* [count + 1];

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

temp[i] = arr[i];

}

temp[index] = figure;

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

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

}

arr = temp;

count++;

}

void MassiveFigures::deleteFromTheEnd() {

geometricFigures\*\* temp = new geometricFigures \* [count - 1];

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

temp[i] = arr[i];

}

arr = temp;

count--;

}

void MassiveFigures::del(int index) {

geometricFigures\*\* temp = new geometricFigures \* [count - 1];

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

temp[i] = arr[i];

}

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

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

}

arr = temp;

count--;

}

Demonstration.cpp

#include "Ellipse.h"

#include "Rectangle.h"

#include "Rhombus.h"

#include "MassiveFigures.h"

#include "error.h"

void menu(MassiveFigures arr);

int main() {

setlocale(LC\_ALL, "ru");

system("color f0");

cout << "Задание. Вариант 19." << endl

<< "Написать программу, в которой описана иерархия классов: геометрические фигуры (ромб, прямоугольник, эллипс)."

<< "Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором пергрузить операцию [ ],"

<< "а также реализовать функцию подсчета общей площади и параметра. Для базового класса и его потомков перегрузить"

<< "операции ==, !=, =. Продеманстрировать работу операторов." << endl;

system("pause");

system("cls");

MassiveFigures arr(0);

menu(arr);

return 0;

}

void menu(MassiveFigures arr) {

bool off = false;

while (!off) {

cout << "\tМЕНЮ" << endl << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "1 - добавить ромб\n2 - добавить прямоугольник\n3 - добавить эллипс" << endl

<< "4 - сравнить прямоугольники\n5 - распечатать фигуру по индексу\n6 - распечатать все фигуры" << endl

<< "7 - удалить с конца\n8 - удалить по индексу\n0 - Exit" << endl;

try

{

int num;

cin >> num;

cout << endl;

switch (num)

{

case 1:

{

system("cls");

Rhombus\* obj1 = new Rhombus(0,0);

obj1->Read();

Rhombus\* obj2 = new Rhombus(0,0);

obj2 = obj1;

bool temp = false;

cout << "Добавить в конец(0) или по индексу(1)?";

cin >> temp;

if (temp) {

int index;

cout << "Введите индекс: ";

cin >> index;

arr.add(index, obj2);

}

else {

arr.addToTheEnd(obj2);

}

system("cls");

break;

}

case 2:

{

system("cls");

Rectangle\* obj = new Rectangle(0, 0);

obj->Read();

bool temp = false;

cout << "Добавить в конец(0) или по индексу(1)?";

cin >> temp;

if (temp) {

int index;

cout << "Введите индекс: ";

cin >> index;

arr.add(index, obj);

}

else {

arr.addToTheEnd(obj);

}

system("cls");

break;

}

case 3:

{

system("cls");

Ellipse\* obj = new Ellipse(0, 0);

obj->Read();

bool temp = false;

cout << "Добавить в конец(0) или по индексу(1)?";

cin >> temp;

if (temp) {

int index;

cout << "Введите индекс: ";

cin >> index;

arr.add(index, obj);

}

else {

arr.addToTheEnd(obj);

}

system("cls");

break;

}

case 4:

{

system("cls");

Rectangle obj1(0, 0);

obj1.Read();

Rectangle obj2(0, 0);

obj2.Read();

if (obj1 == obj2) {

cout << "Фигуры равны" << endl;

}

else if (obj1 != obj2) {

cout << "Фигуры не равны" << endl;

}

system("pause");

system("cls");

break;

}

case 5:

{

system("cls");

int index;

cout << "Введите индекс: ";

cin >> index;

if (arr[index] != 0) {

arr[index]->Print();

}

system("pause");

system("cls");

break;

}

case 6:

{

system("cls");

cout << "Количество фигур = " << arr.get\_count() << endl;

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

arr[i]->Print();

}

system("pause");

system("cls");

break;

}

case 7:

{

system("cls");

arr.deleteFromTheEnd();

system("cls");

break;

}

case 8:

{

system("cls");

int index;

cout << "Введите индекс: ";

cin >> index;

arr.del(index);

system("cls");

break;

}

case 0:

off = true;

}

cout << endl;

}

catch (IndexError & e)

{

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

system("pause");

system("cls");

}

catch (...)

{

cout << "Неизвестная ошибка." << endl;

system("pause");

system("cls");

}

}

}

Вывод: изучил правила перегрузки операций и принципов обработки исключений в С++.