Министерство образования Республики Беларусь

Учреждение образования

“Брестский государственный университет”

Кафедра ИИТ

Лабораторная работа №3

По дисциплине: “ЯП”

Тема: “Перегрузка операций. Исключения”

Выполнил:

Студент 2-го курса

Группы ПО-7

Смушко О.Р.

Проверила:

Дряпко А.В.

Брест, 2021

**Вариант 5**

**Цель работы:** Изучение правил перегрузки операций и принципов обработки исключений C++

**Задание:** Написать программу, в которой описана иерархия классов: ошибка в программе (ошибочный указатель, ошибка работы со списком, недопустимый индекс, список переполнен). Описать класс для хранения коллекции ошибок (массива указателей на базовый класс), в котором перегрузить операцию []. Для базового класса и его потомков перегрузить операции ==, !=, =. Продемонстрировать работу операторов.

**Код программы:**

**Определение классов:**

**PointerError.h**

#pragma once

#include "ArrayError.h"

class PointerError : public ArrayError

{

public:

PointerError(int \*ptr);

int\* Pointer;

bool operator == (const PointerError& right) { //overloading == operator

return Pointer == right.Pointer;

}

bool operator != (PointerError\* const right) { //overloading != operator

return this != right;

}

PointerError& operator = (const PointerError& right) { //overloading = operator

Pointer = right.Pointer;

return \*this;

}

};

**ErrorInList.h**

#pragma once

#include "ArrayError.h"

class ErrorInList :public ArrayError

{

public:

ErrorInList(int value);

int List[10];

};

**ErrorIndex.h**

#pragma once

#include "ArrayError.h"

class ErrorIndex : public ArrayError

{

public:

ErrorIndex(int index);

int List[10];

int Index;

bool operator == (const ErrorIndex& right) { //overloading == operator

return Index == right.Index;

}

bool operator != (ErrorIndex\* const right) { //overloading != operator

return this != right;

}

ErrorIndex& operator = (const ErrorIndex& right) { //overloading = operator

Index = right.Index;

return \*this;

}

};

**ErrorOverflow.h**

#pragma once

#define list\_max\_index 10

#include "ArrayError.h"

class ErrorOverflow :public ArrayError

{

public:

ErrorOverflow(int index);

int List[10];

int Index;

bool operator == (const ErrorOverflow& right) { //overloading == operator

return Index == right.Index;

}

bool operator != (ErrorOverflow\* const right) { //overloading != operator

return this != right;

}

ErrorOverflow& operator = (const ErrorOverflow& right) { //overloading = operator

Index = right.Index;

return \*this;

}

};

**all\_errors.h**

#include<string>

#include<typeinfo>

#include "ArrayError.h"

class ErrorArray {

ArrayError\*\* ptrArray; //PointerArray

int count; //ArrayElementCount

public:

ErrorArray();

int Length() const;

void AddToList(ArrayError\* error);

void PrintArray() const;

void operator[](int i);

};

**PointerError.cpp**

#include "PointerError.h"

PointerError::PointerError(int \*ptr) {

Pointer = ptr;

if (Pointer == NULL) {

ErrorText = "Pointer Error";

}

else {

ErrorText = "All done";

}

**ErrorInList.cpp**

#include "ErrorInList.h"

#include<iostream>

using namespace std;

ErrorInList::ErrorInList(int value) {

if (value<0) {

ErrorText = "Error while working with array";

}

else {

ErrorText = "All done.";

}

}

**ErrorIndex.cpp**

#include "ErrorIndex.h"

ErrorIndex::ErrorIndex(int index) {

Index = index;

if (index < 0 ) {

ErrorText = "Error while working with index.";

}

else {

ErrorText = "All done.";

}

}

**all\_erors.cpp**

#include "all\_errors.h"

#include<iostream>

using namespace std;

ErrorArray::ErrorArray() : ptrArray(nullptr), count(0) {}

int ErrorArray::Length() const { return count; }

void ErrorArray::AddToList(ArrayError\* error) {

if (count == 0) {

count = 1;

ptrArray = new ArrayError\*;

ptrArray[0] = error;

std::cout << "Added to list ( created list )" << std::endl;

}

else {

count++;

auto temp = ptrArray;

ptrArray = new ArrayError \* [count];

for (int i = 0; i < count; i++) { ptrArray[i] = temp[i]; }

ptrArray[count - 1] = error;

delete[] temp;

cout << "Added to list" << endl;

}

}

void ErrorArray::PrintArray() const {

cout << "------------\nErrorArray: \n" << endl;

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

cout << ptrArray[i]->ErrorText << endl;

}

cout << "------------" << endl;

}

void ErrorArray::operator[](int i) {

cout << "errors[" << i << "]: " << endl;

cout << ptrArray[i]->ErrorText << endl;

}

**Основная программа:**

**Main.cpp**

#include <iostream>

#include "ArrayError.h"

#include "PointerError.h"

#include "ErrorIndex.h"

#include "ErrorInList.h"

#include "ErrorOverflow.h"

#include "all\_errors.h"

int main() {

int\* arr = new int[10];

for (int i = 0; i < 10; i++) {

arr[i] = 1;

}

ArrayError a = ArrayError();

cout << a.ErrorText << endl;

int\* testptr = &arr[3];

PointerError pe = PointerError(testptr);

PointerError pe2 = pe;

bool check = pe == pe2;

cout << "Result: " << check << endl;

cout << pe.ErrorText << endl;

int index = 6;

ErrorIndex ei = ErrorIndex(index);

ErrorIndex ei2 = ErrorIndex(index);

bool check1 = ei == ei2;

cout << "Result: " << check1 << endl;

cout << ei.ErrorText << endl;

int value = -10;

ErrorInList eil = ErrorInList(value);

cout << eil.ErrorText << endl;

ErrorOverflow eo = ErrorOverflow(index);

cout << eo.ErrorText << endl;

ErrorArray ea;

ea.AddToList(&a);

ea.AddToList(&pe);

ea.AddToList(&eo);

ea.PrintArray();

return 0;

}

**Вывод:** В ходе лабораторной работы научился перегружать операторы, реализовал иерархию классов, а также научился “ловить” исключения.