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 2 курса

группы ПО-7

Комиссаров Андрей

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: 6. Hаписать программу, в которой описана иерархия классов: ошибка в программе («недостаточно привилегий», «ошибка преобразования», «невозможно преобразовать значение», «невозможно привести к интерфейсу»). 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:

**Файл main.cpp:**

#include <iostream>

#include <iomanip>

#include <string.h>

#include "errors.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

std::cout << "6. Hаписать программу, в которой описана иерархия классов: ошибка в программе («недостаточно привилегий», «ошибка преобразования», «невозможно преобразовать значение», «невозможно привести к интерфейсу»). 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.\n" << std::endl;

AccessError AError("недостаточно привелегий");

TranslateError TError1("ошибка преобразования 1");

TranslateError TError2("ошибка преобразования 2");

ValueTranslateError VTError("невозможно преобразовать значение");

InterfaceTranslateError ITError("невозможно привести к интерфейсу");

ErrorArray EA;

std::cout << "\nСписок элементов:" << endl;

EA.AddToList(&AError);

EA.AddToList(&TError1);

EA.AddToList(&TError2);

EA.AddToList(&VTError);

EA.AddToList(&ITError);

EA.PrintArray();

std::cout << "\nПерегрузка операции [] для списка:" << endl;

EA[2];

EA[1];

std::cout << "\nПерегрузка операции сравнения (==) и приравнивания (=): \nДо приравнивания:" << endl;

std::cout << (TError1 == TError2) << std::endl;

TError1 = TError2;

std::cout << "После приравнивания:" << endl;

std::cout << (TError1 == TError2) << std::endl;

std::cout << "Перегрузка неравенства (!=):" << endl;

std::cout << (TError1 != TError2) << endl;

return 0;

}

**Файл Errors.H:**

#ifndef ERRORS\_H

#define ERRORS\_H

class Error {

protected:

std::string errMsg;

public:

static int count;

static Error\*\* items;

Error(std::string message);

~Error();

void PrintMsg();

bool operator==(const Error& other) {

bool result;

result = (typeid(\*this) == typeid(other)) && (errMsg == other.errMsg);

if (result == 1){

std::cout << "\"" << this->errMsg << "\" равно \"" << other.errMsg << "\"" << std::endl;

}

else {

std::cout <<"\"" << this->errMsg << "\" НЕ равно \"" << other.errMsg << "\"" << std::endl;

}

return result;

}

bool operator!=(const Error& other) {

return !(\*this == other);

}

Error& operator=(const Error& other) {

errMsg = other.errMsg;

std::cout << "Присваиваем один элемент к другому... " << std::endl;

return \*this;

}

};

class AccessError : public Error {

protected:

int accessLevelNeeded;

public:

AccessError(std::string message);

~AccessError();

};

class TranslateError : public Error {

public:

TranslateError(std::string message);

~TranslateError();

virtual void PrintTypes();

};

class ValueTranslateError : public TranslateError {

public:

ValueTranslateError(std::string message);

~ValueTranslateError();

void PrintTypes() override;

};

class InterfaceTranslateError : public TranslateError {

public:

InterfaceTranslateError(std::string message);

~InterfaceTranslateError();

void PrintTypes() override;

};

class ErrorArray {

Error\*\* ptrArray; //PointerArray

int count; //ArrayElementCount

public:

ErrorArray();

int Length() const;

void AddToList(Error\* error);

void PrintArray() const;

void operator[](int i);

};

#endif

**Файл Errors.CPP:**

#include <iostream>

#include <iostream>

#include <iomanip>

#include <string.h>

#include "errors.h"

Error::Error(std::string message) : errMsg(message) {

std::cout << "created error " << errMsg << std::endl;

}

Error::~Error() {}

void Error::PrintMsg() { std::cout << errMsg << std::endl; }

AccessError::AccessError(std::string message) : Error(message), accessLevelNeeded(10) {}

AccessError::~AccessError() {}

TranslateError::TranslateError(std::string message) : Error(message) {}

TranslateError::~TranslateError() {}

void TranslateError::PrintTypes() {

std::cout << "translateError message" << std::endl;

}

ValueTranslateError::ValueTranslateError(std::string message) : TranslateError(message) {}

ValueTranslateError::~ValueTranslateError() {}

void ValueTranslateError::PrintTypes() {

std::cout << "value\_TranslateError message" << std::endl;

}

InterfaceTranslateError::InterfaceTranslateError(std::string message) : TranslateError(message) {}

InterfaceTranslateError::~InterfaceTranslateError() {}

void InterfaceTranslateError::PrintTypes() {

std::cout << "interface\_TranslateError message" << std::endl;

}

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

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

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

if (count == 0) { count = 1; ptrArray = new Error\*; ptrArray[0] = error; std::cout << "Added to list ( created list )" << std::endl; }

else {

count++;

auto temp = ptrArray;

ptrArray = new Error \* [count];

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

ptrArray[count - 1] = error;

delete[] temp;

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

}

}

void ErrorArray::PrintArray() const {

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

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

ptrArray[i]->PrintMsg();

}

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

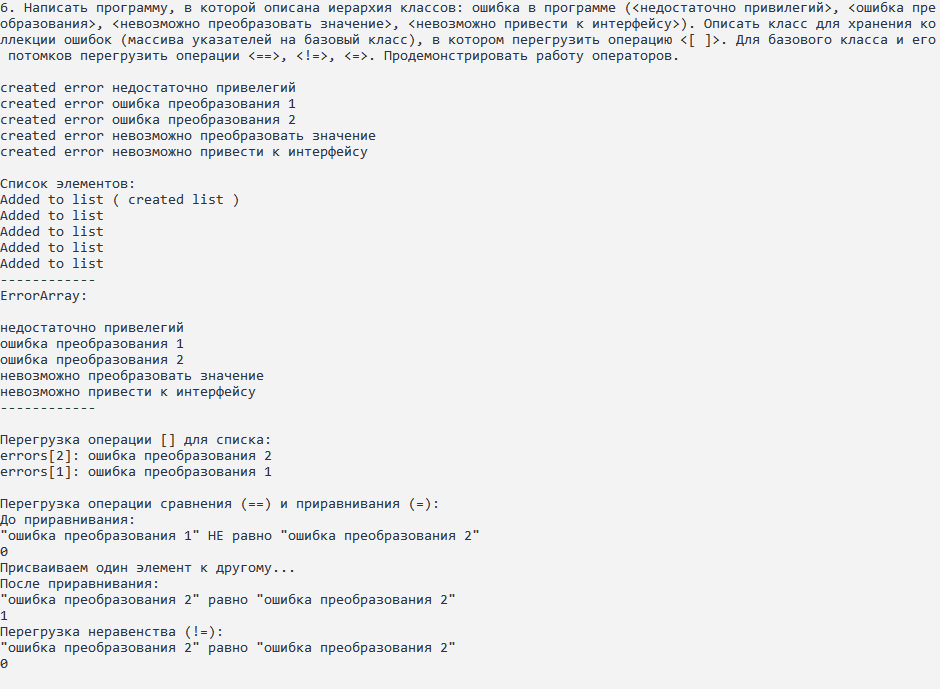
}

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

std::cout << "errors[" << i << "]: "; ptrArray[i]->PrintMsg();

}

# Результат работы программы:



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