Министерство образования Республики Беларусь

УО «Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №3**

По дисциплине: “Языки программирования”

Тема: “Перегрузка операций.Исключения”**Вариант №1**

**Выполнил**:

студент 2 курса группы ПО-7 Белев С. В.

**Проверила:**

Дряпко А. В.

Брест 2021

**Цель:**

Изучение правил перегрузки и принципов обработки исключений в С++.

**Постановка задачи:**

Написать программу, в которой описана иерархия классов: ошибка в программе(ошибка доступа к памяти, математическая, деление на ноль, переполнение). 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:**

#include <iostream>

#include <climits>

#define \_CRT\_SECURE\_NO\_WARNINGS

class EBaseError {

public:

virtual void Print() = 0;

virtual void Read() = 0;

};

class EAccessViolation : public EBaseError {

void\* m\_badAddr;

public:

void Print();

void Read();

EAccessViolation(void\* badAddr);

bool operator ==(const EAccessViolation& right) {

return m\_badAddr == right.m\_badAddr;

}

bool operator != (const EAccessViolation& right) {

return !(\*this == right);

}

EAccessViolation& operator= (const EAccessViolation& right) {

m\_badAddr = right.m\_badAddr;

return \*this;

}

};

class EMathError : public EBaseError {};

class EZeroDivide : public EMathError {

double m\_divident;

public:

void Print();

void Read();

EZeroDivide(const double& divident);

bool operator == (const EZeroDivide& right) {

return m\_divident == right.m\_divident;

}

bool operator != (const EZeroDivide& right) {

return !(\*this == right);

}

EZeroDivide& operator= (const EZeroDivide& right) {

m\_divident = right.m\_divident;

return \*this;

}

};

class EOverflow : public EMathError {

int m\_operand1, m\_operand2;

public:

void Print();

void Read();

EOverflow(const int& operand1, const int& operand2);

bool operator == (const EOverflow& right) {

return (m\_operand1 == right.m\_operand1) && (m\_operand2 == right.m\_operand2);

}

bool operator != (const EOverflow& right) {

return !(\*this == right);

}

EOverflow& operator= (const EOverflow& right) {

m\_operand1 = right.m\_operand1;

m\_operand2 = right.m\_operand2;

return \*this;

}

};

typedef EBaseError\* pError;

class CErrors {

pError\* m\_errs;

int m\_cnt;

public:

CErrors(int errCount) {

m\_errs = new pError[errCount];

for (int i = 0; i < errCount; i++)

m\_errs[i] = NULL;

m\_cnt = errCount;

}

~CErrors() {

for (int i = 0; i < m\_cnt; i++)

if (m\_errs[i] != NULL) delete m\_errs[i];

delete[] m\_errs;

}

EBaseError\* operator[] (int n) const {

if (n < 0 || n >= m\_cnt) throw EAccessViolation((void\*)&m\_errs[n]);

return m\_errs[n];

}

EBaseError\*& operator[] (int n) {

if (n < 0 || n >= m\_cnt) throw EAccessViolation((void\*)&m\_errs[n]);

return m\_errs[n];

}

};

EAccessViolation::EAccessViolation(void\* badAddr) {

m\_badAddr = badAddr;

}

void EAccessViolation::Print() {

printf("AccessViolation read of address %p!", m\_badAddr);

}

void EAccessViolation::Read(){

printf("Simulate EAccessViolation, enter badAddress ");

scanf\_s("%p", &m\_badAddr);

}

EZeroDivide::EZeroDivide(const double &divident) {

m\_divident = divident;

}

void EZeroDivide::Print() {

printf("There was a try to divide %lf by zero!", m\_divident);

}

void EZeroDivide::Read() {

printf("Simulate EZeroDivide, enter divident");

scanf\_s("%lf", &m\_divident);

}

EOverflow::EOverflow(const int &operand1, const int &operand2) {

m\_operand1 = operand1;

m\_operand2 = operand2;

}

void EOverflow::Print() {

printf("There was an overflow during some operation between %d and %d!",m\_operand1, m\_operand2);

}

void EOverflow::Read() {

printf("Simulate EOverflow, enter two operands");

scanf\_s("%d %d", &m\_operand1, &m\_operand2);

}

void disposeError(EBaseError\*\* e) {

if (\*e != NULL) delete\* e;

\*e = NULL;

}

int main() {

setlocale(LC\_ALL, "Russian");

EBaseError\* e = NULL;

CErrors errList(10);

int curErrIndex = 0;

char c = 0, tmp;

while (c != 27) {

printf("\nвыберите действие:\n 1- эмулировать ошибку доступа\n");

printf(" 2 - попытаться поделить два числа \n 3 - попытаться умножить два числа \n");

printf(" 4 - получить ошибку по номеру\n 5 - сравнить две ошибки\n");

printf(" 6 - напечатать все ошибки \n");

printf(" 0 - выйти из программы\n");

try {

scanf\_s("%c", &c);

switch (c) {

case '1': {e = new EAccessViolation((void\*)&c);

break; }

case '2': {

double a, b;

printf("\nвведите делимое ");

scanf\_s(" %lf", &a);

printf("\nвведите делитель ");

scanf\_s(" %lf", &b);

if (b == 0.0) e = new EZeroDivide(a);

else printf("\nрезультат = %lf", a / b);

break;

}

case '3': {

int i, j;

long long res;

printf("\nвведите первый множитель ");

scanf\_s("%d", &i);

printf("\nвведите второй множитель ");

scanf\_s("%d", &j);

res = i;

res \*= j;

if (res > INT\_MAX || res < INT\_MIN) e = new EOverflow(i, j);

else printf("\nрезультат = %d", res);

break;

}

case '4': {

int inx;

printf("\nвведите номер ошибки ");

scanf\_s("%d", &inx);

EBaseError\* tmpe = errList[inx];

if (tmpe == NULL) printf("\nОшибка пуста\n");

else {

printf("\n>\t");

tmpe->Print();

}

break;

}

case '5': {

EAccessViolation ea1(0), ea2(0);

ea1.Read();

ea2.Read();

if (ea1 == ea2) printf("\nошибки равны\n");

else printf("\nошибки не равны\n");

break;

}

case '6':

for (int i = 0; i < curErrIndex; i++)

if (errList[i] != NULL) {

printf("\n%d\t", i);

errList[i]->Print();

}

break;

case '0':c = 27;

}

scanf\_s("%c", &tmp);

if (e != NULL) {

printf("\n>\t");

e->Print();

printf("\n");

if (curErrIndex < 10) {

errList[curErrIndex++] = e;

e = NULL;

}

else disposeError(&e);

}

}

catch (EBaseError& re) {

printf("\n ошибка времени выполнения\t");

re.Print();

printf("\n");

}

catch (...) {

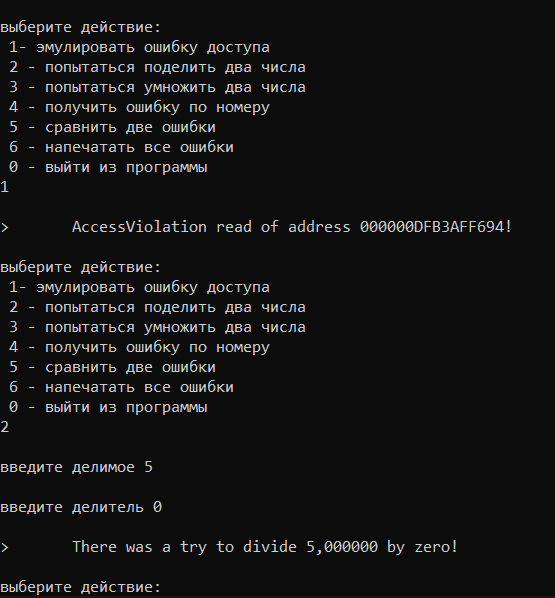
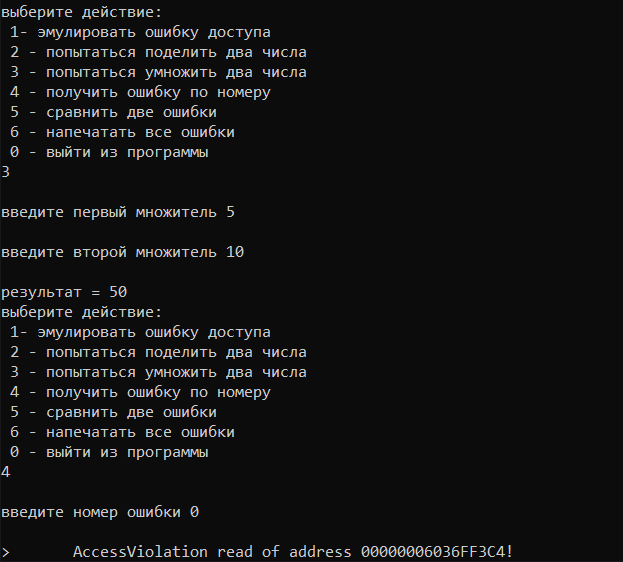
printf("\n неизвестная ошибка времени выполнения\n");

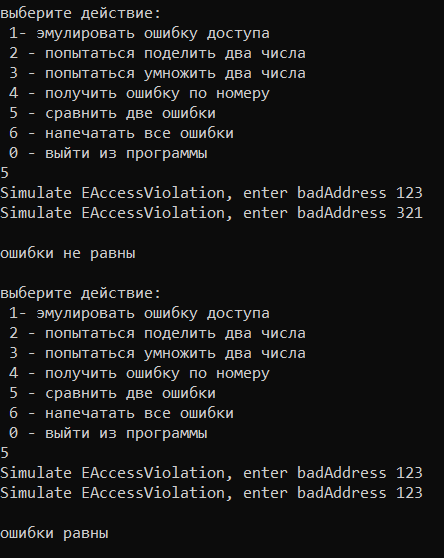
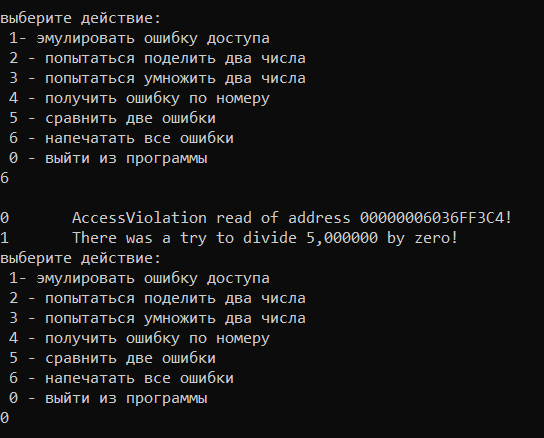
}

}

}

**Результат программы:**

**Вывод:**

Я изученил правила перегрузки и принципы обработки исключений в С++.