**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №18.7

Вариант №11

Выполнил студент группы РИС-20-2б

Савельев Виталий Антонович

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –научиться работать с одним из важнейших механизмов языка программирования C++ обработки исключительных ситуаций, точнее корректный отлов ошибок и поведение программы при некорректно-введённых данных.

**Постановка задачи**

Задача – реализовать на языке программирования C++ программу и несколько типов исключений, реализовать исключительных ситуации и продемонстрировать их работу, отлов и вывод соответствующих сообщений.

**Исходные данные для варианта №11:**

* Исключение с недопустимым номеров элемента в массиве;
* Исключение с неравными длинами двух массивов;
* Исключение с размером массива;
* Используется класс списка из предыдущих лабораторных работ;

**Анализ задачи**

Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code
3. Система контроля версий Git

Класс ListInt, что был реализован ранее содержит в себе три перегруженных оператора:

1. Доступ к элементам по индексу;
2. Оператор присваивания;
3. Оператор почленного сложение двух массивов;

class ListInt

{

private:

int\* \_data;

int \_length;

public:

ListInt(int length, int fillValue);

ListInt(const ListInt& copied);

~ListInt();

int& operator [] (int index);

ListInt& operator = (const ListInt& assigned);

ListInt operator + (const ListInt& adder);

int Length() { return \_length; }

};

В каждом из операторов возможно появление исключительной ситуации:

* Попытка почленно сложить списки разной длины;
* Попытка работы с пустым списком;
* Неверный индекс, выходящий за пределы списка;

Для каждой исключительной ситуации создаётся свой класс-исключение, который в свою очередь содержит соответствующий метод с сообщением:

1. Исключение с индексом:

class IndexAccessExeption

{

public:

void Message();

};

#include "IndexAccessExeption.h"

#include <iostream>

void IndexAccessExeption::Message()

{

std::cout << "Out of index range exeption;\n";

}

1. Исключение не равным списком:

class NotEqualLengthExeption

{

public:

void Message();

};

#include "NotEqualLengthExeption.h"

#include <iostream>

void NotEqualLengthExeption::Message()

{

std::cout << "Not the same list lengths exeption;\n";

}

1. Исключение с размерами списка:

class SizeExeption

{

public:

void Message();

};

#include "SizeExeption.h"

#include <iostream>

void SizeExeption::Message()

{

std::cout << "Incorrect list size exeption\n";

}

Основная программа, которая через try catch обрабатывает соответствующие исключительные ситуации, которые намерено были добавлены в код программы:

#include <iostream>

#include "NotEqualLengthExeption.h"

#include "IndexAccessExeption.h"

#include "SizeExeption.h"

#include "ListInt.h"

using namespace std;

int main()

{

ListInt first(10, 5);

ListInt second (11, 73);

try

{

ListInt summ = first + second; //Equality exeption

}

catch (NotEqualLengthExeption e)

{

e.Message();

}

try {

ListInt good(23, 22);

good[-1] = 5;

}

catch (IndexAccessExeption e)

{

e.Message();

}

try

{

ListInt longList(100, 4); // SizeExeption

}

catch (SizeExeption e)

{

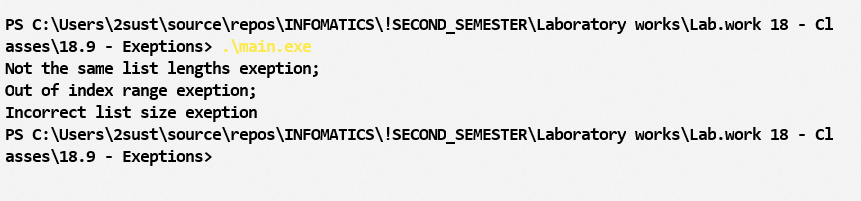
e.Message();

}

return 0;

}

**Скриншот выполненной программы:**



**Ответы на вопросы:**

**Что представляет собой исключение в С++?**

Исключение – это непредвиденное или аварийное событие.

**На какие части исключения позволяют разделить вычислительный процесс? Достоинства такого подхода?**

Исключения позволяют разделить вычислительный процесс на 2 части:

* обнаружение аварийной ситуации (неизвестно как обрабатывать);
* обработка аварийной ситуации (неизвестно, где она возникла).

Достоинства такого подхода:

* удобно использовать в программе, которая состоит из нескольких модулей;
* не требуется возвращать значение в вызывающую функцию

**Какой оператор используется для генерации исключительной ситуации?**

throw <выражение>, где <выражение> -

* либо константа,
* либо переменная некоторого типа,
* либо выражение некоторого типа.

**Что представляет собой контролируемый блок? Для чего он нужен?**

try

{

….

//ошибка

throw выражение

//генерируется исключение

Его задача: обнаружение аварийной ситуации.

**Что представляет собой секция-ловушка? Для чего она нужна?**

catch (……)

Её задача: обработка аварийной ситуации

**Какие формы может иметь спецификация исключения в секции ловушке? В каких ситуациях используются эти формы?**

Спецификация исключения может иметь три формы:

1. (тип имя)
2. (тип)
3. (…)

Тип – это встроенный тип или тип, определенный программистом.

Формы 1 и 2 обрабатывают конкретные исключения, а форма 3 перехватывает все исключения, такую ловушку надо помещать последней, тогда она будет обрабатывать все исключения, которые еще не были обработаны.

**Какой стандартный класс можно использовать для создания собственной иерархии исключений?**

Класс exception;

**Каким образом можно создать собственную иерархию исключений?**

Для создания собственной иерархии исключений надо объявить свой базовый класс- исключение, например:

class BaseException{};

Остальные классы будут наследниками этого класса, аналогично тому, как это сделано в иерархии стандартных исключений:

class Child\_Exception1:public BaseException{};

class Child\_Exception2:public BaseException{};

Класс BaseException можно унаследовать от стандартного класса exception class BaseException: public exception{};

**Если спецификация исключений имеет вид:**

**void f1()throw(int,double); то какие исключения может прождать функция f1()?**

Исключения типа int и double.

**Если спецификация исключений имеет вид: void f1()throw(); то какие исключения может прождать функция f1()?**

Исключения любых типов.

**В какой части программы может генерироваться исключение?**

Внутри блока try{}

**Полный исходный код программы на языке программирования C++:**

IndexAccessExeption.h

class IndexAccessExeption

{

public:

void Message();

};

IndexAccessExeption.cpp

#include "IndexAccessExeption.h"

#include <iostream>

void IndexAccessExeption::Message()

{

std::cout << "Out of index range exeption;\n";

}

NotEqualLengthExeption.h

class NotEqualLengthExeption

{

public:

void Message();

};

NotEqualLengthExeption.cpp

#include "NotEqualLengthExeption.h"

#include <iostream>

void NotEqualLengthExeption::Message()

{

std::cout << "Not the same list lengths exeption;\n";

}

SizeExeption.h

class SizeExeption

{

public:

void Message();

};

SizezExeption.cpp

#include "SizeExeption.h"

#include <iostream>

void SizeExeption::Message()

{

std::cout << "Incorrect list size exeption\n";

}

ListInt.h

class ListInt

{

private:

int\* \_data;

int \_length;

public:

ListInt(int length, int fillValue);

ListInt(const ListInt& copied);

~ListInt();

int& operator [] (int index);

ListInt& operator = (const ListInt& assigned);

ListInt operator + (const ListInt& adder);

int Length() { return \_length; }

};

ListInt.cpp

#include "ListInt.h"

#include "SizeExeption.h"

#include "NotEqualLengthExeption.h"

#include "IndexAccessExeption.h"

#include <iostream>

using namespace std;

const int MAX\_LENGTH = 30;

ListInt::ListInt(int length, int fillValue)

{

if ((length > MAX\_LENGTH) || (length <= 0))

throw SizeExeption();

\_length = length;

\_data = new int[\_length];

for (int i = 0; i < \_length; i++)

\_data[i] = fillValue;

}

ListInt::ListInt(const ListInt& copied)

{

\_length = copied.\_length;

\_data = new int[\_length];

for (int i = 0; i < \_length; i++)

\_data[i] = copied.\_data[i];

}

ListInt::~ListInt()

{

delete[] \_data;

\_data = nullptr;

}

int& ListInt::operator[](int index)

{

if ((index < 0) || (index >= \_length))

throw IndexAccessExeption();

return \_data[index];

}

ListInt& ListInt::operator=(const ListInt& assigned)

{

\_length = assigned.\_length;

delete[] \_data;

\_data = new int[\_length];

for (int i = 0; i < \_length; i++)

\_data[i] = assigned.\_data[i];

return \*this;

}

ListInt ListInt::operator+(const ListInt& adder)

{

if (\_length != adder.\_length)

throw NotEqualLengthExeption();

ListInt temporarilyList(\_length, 0);

for (int i = 0; i < \_length; i++)

temporarilyList[i] = \_data[i] + adder.\_data[i];

return temporarilyList;

}

main.cpp

#include <iostream>

#include "NotEqualLengthExeption.h"

#include "IndexAccessExeption.h"

#include "SizeExeption.h"

#include "ListInt.h"

using namespace std;

int main()

{

ListInt first(10, 5);

ListInt second (11, 73);

try

{

ListInt summ = first + second; //Equality exeption

}

catch (NotEqualLengthExeption e)

{

e.Message();

}

try {

ListInt good(23, 22);

good[-1] = 5;

}

catch (IndexAccessExeption e)

{

e.Message();

}

try

{

ListInt longList(100, 4); // SizeExeption

}

catch (SizeExeption e)

{

e.Message();

}

return 0;

}