МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра информационные технологии и автоматизированные системы

**Дисциплина Информатика**

**Обработка исключительных ситуаций**

Выполнил студент ИВТ-22-2б:

Мифтахов Марат Ринатович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Пермь, 2023

**Постановка задачи**

1. Реализовать класс, перегрузить для него операции, указанные в варианте.
2. Определить исключительные ситуации.
3. Предусмотреть генерацию исключительных ситуаций.

**Задание**

1. Класс-контейнер список с ключевыми значениями типа int.
2. Реализовать операции:
3. [] – доступ по индексу;
4. Int() – определение размера списка;
5. +n – переход вправо к элементу с номером n (с помощью класса-итератора).

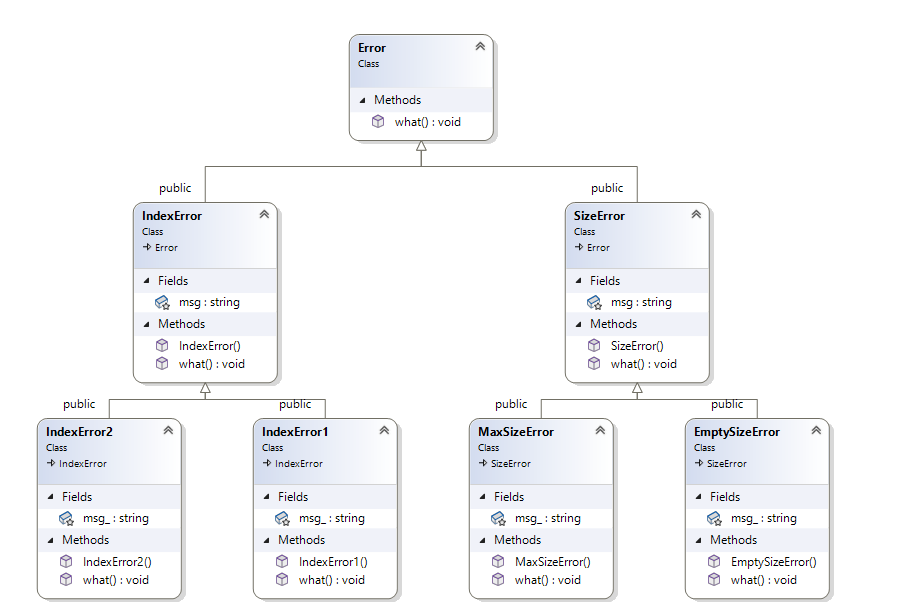
**Описание**

Генерация исключительных ситуаций происходит в следующих случаях:

1. Выход за границы размерного диапазона [] справа (size+ 1)
2. Выход за границы размерного диапазона [] слева (-1)
3. Выход итератора за границы размерного диапазона +n

**Диаграмма классов**

Ниже представлена UML-диаграмма классов.



**Программный код**

Файл Error.h:

#pragma once

#include <string>

#include <iostream>

using namespace std;

class Error {

public:

virtual void what() {};

};

class IndexError : public Error {

protected:

string msg;

public:

IndexError() {

msg = "Index Error\n";

}

virtual void what() { cout << msg; }

};

class SizeError: public Error {

protected:

string msg;

public:

SizeError() {

msg = "Size Error\n";

}

virtual void what() { cout << msg; }

};

class MaxSizeError : public SizeError {

protected:

string msg\_;

public:

MaxSizeError(): SizeError() {

msg\_ = "size>MAXSIZE\n";

}

virtual void what() { cout << msg << msg\_; }

};

class EmptySizeError : public SizeError {

protected:

string msg\_;

public:

EmptySizeError(): SizeError() {

msg\_ = "List is empty\n";

}

virtual void what() { cout << msg << msg\_; }

};

class IndexError1 : public IndexError {

protected:

string msg\_;

public:

IndexError1(): IndexError() {

msg\_ = "index < 0\n";

}

virtual void what() { cout << msg << msg\_; }

};

class IndexError2 : public IndexError {

protected:

string msg\_;

public:

IndexError2(): IndexError() {

msg\_ = "index>size\n";

}

virtual void what() { cout << msg << msg\_; }

};

Файл List.h:

#include <iostream>

#include "Error.h"

class Iterator;

class List;

class Node {

friend class List;

friend class Iterator;

public:

Node();

Node(int);

~Node();

int getData();

Node\* getPtr();

private:

int data;

Node\* ptrNext;

};

class Iterator {

public:

Iterator();

Iterator(const Iterator&);

void operator++();

void operator+(int);

int& operator\*();

private:

friend class List;

Node\* curr;

};

class List {

public:

List();

List(int);

List(const List&);

~List();

int& operator[](int);

List& operator=(const List&);

size\_t operator()();

friend List operator\*(List&, List&);

friend std::ostream& operator<<(std::ostream&, const List&);

friend std::istream& operator>>(std::istream&, List&);

void push(int);

Iterator first() { return beg; }

private:

Node\* head;

size\_t size;

Iterator beg;

};

Node::Node() {

data = 0;

ptrNext = nullptr;

}

Node::Node(int data) {

this->data = data;

ptrNext = nullptr;

}

Node::~Node() {

if (ptrNext != nullptr) {

delete ptrNext;

}

}

int Node::getData() {

return data;

}

Node\* Node::getPtr() {

return ptrNext;

}

/// <summary>

/// Node\* head;

/// </summary>

List::List() {

head = nullptr;

size = 0;

beg.curr = head;

}

List::List(int data) {

head = new Node(data);

size = 1;

beg.curr = head;

}

List::List(const List& tList) {

Node\* curr = tList.head;

beg.curr = tList.head;

while (curr != nullptr) {

int a = curr->data;

this->push(a);

curr = curr->ptrNext;

}

}

List::~List() {

if (head != nullptr) {

delete head;

head = head->ptrNext;

}

size = 0;

beg.curr = head;

}

void List::push(int data) {

if (head == nullptr) {

head = new Node(data);

beg.curr = head;

}

else {

Node\* curr = head;

while (curr->ptrNext != nullptr) {

curr = curr->ptrNext;

}

curr->ptrNext = new Node(data);

}

size++;

}

int& List::operator[](int indx) {

if (indx < 0) throw IndexError1();

Node\* curr = head;

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

curr = curr->ptrNext;

if (curr == nullptr) throw IndexError2();

}

return curr->data;

}

size\_t List::operator()() {

return size;

}

List operator\*(List& fList, List& sList) {

List tmpList;

int maxSize = fList.size > sList.size ? fList.size : sList.size;

int minSize = fList.size <= sList.size ? fList.size : sList.size;

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

int a = fList[i];

int b = sList[i];

tmpList.push(a \* b);

}

for (int i = 0; i < maxSize - minSize; i++) {

tmpList.push(0);

}

return tmpList;

}

List& List::operator=(const List& tList) {

Node\* curr = tList.head;

beg.curr = tList.head;

while (curr != nullptr) {

int a = curr->data;

this->push(a);

curr = curr->ptrNext;

}

return \*this;

}

std::ostream& operator<<(std::ostream& out, const List& tList) {

Node\* curr = tList.head;

for (int i = 0; i < tList.size; i++) {

out << curr->getData() << " ";

curr = curr->getPtr();

}

return out;

}

std::istream& operator>>(std::istream& in, List& tList) {

int a;

in >> a;

tList.push(a);

return in;

}

/// <summary>

/// Node\* curr

/// </summary>

Iterator::Iterator() {

curr = nullptr;

}

Iterator::Iterator(const Iterator& it) {

curr = it.curr;

}

void Iterator::operator++() {

curr = curr->ptrNext;

}

void Iterator::operator+(int n) {

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

curr = curr->ptrNext;

if (curr->ptrNext == nullptr) throw MaxSizeError();

}

}

int& Iterator::operator\*() {

return curr->data;

}

Файл main\_main.cpp:

#include <iostream>

#include "List.h"

using namespace std;

int main() {

try {

List a;

int n;

cout << "n? "; cin >> n;

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

int data;

cin >> data;

a.push(data);

}

int indx;

cout << "indx? " << endl; cin >> indx;

cout << "your elem: " << a[indx] << endl;

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

Iterator beg = a.first();

cout << "elem? " << endl;

cin >> n;

beg + --n;

cout << \*beg;

}

catch (Error &e) {

e.what();

}

return 0;

}

Файл main.cpp:

#include <iostream>

#include "List.h"

using namespace std;

int main() {

try {

List a;

int n;

cout << "n? "; cin >> n;

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

int data;

cin >> data;

a.push(data);

}

int indx;

cout << "indx? " << endl; cin >> indx;

cout << "your elem: " << a[indx] << endl;

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

Iterator beg = a.first();

cout << "elem? " << endl;

cin >> n;

beg + --n;

cout << \*beg;

}

catch (int& c) {

if (c == 1) { cout << "indx < 0" << endl; }

else if (c == 2) { cout << "indx > maxsize" << endl; }

else if (c == 3) { cout << "maxsize error" << endl; }

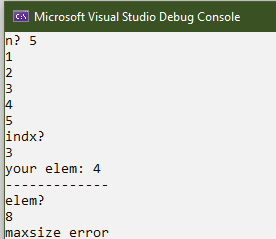
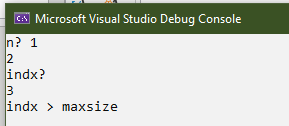
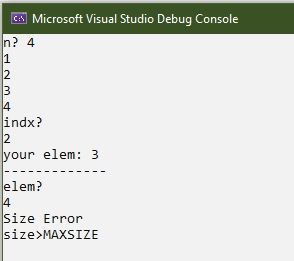
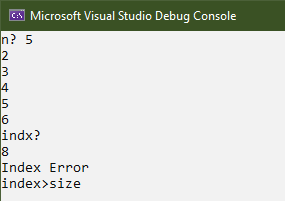
}

return 0;

}

**Вывод программы**

Ниже представлен вывод программы на консоль (main\_main.cpp).

**\_**

Ниже представлен вывод программы на консоль (main.cpp).

**Контрольные вопросы**

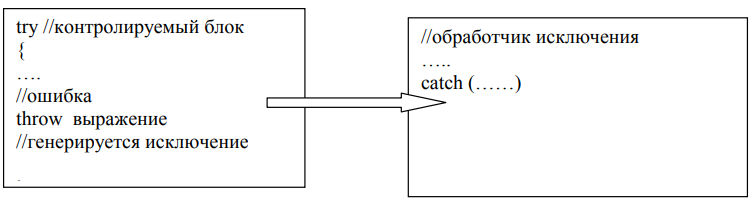
1. Что представляет собой исключение в С++?

Ответ: исключение — объект, генерируемый системой при возникновении исключительной ситуации. Генерация такого объекта создает исключительную ситуацию.

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

Ответ: исключения позволяют разделить вычислительный процесс на 2 части: обнаружение аварийной ситуации и обработка аварийной ситуации.

Достоинства: удобство использования в программе из нескольких модулей, отсутствие требования возвращать значение в вызывающую функцию.



1. Какой оператор используется для генерации исключительной ситуации?

Ответ: оператор throw <значение>. Значение — константа, переменная, выражение.

1. Что представляет собой контролируемый блок? Для чего он нужен?

Ответ: контролируемый блок представляет собой ключевое слово try с блоком кода, помещаемым для проверки на исключительную ситуацию. В нем описывается оператор throw, который передает исключения в блок обработки исключения catch{}

1. Что представляет собой секция-ловушка? Для чего она нужна?

Ответ: блок catch – обработчик исключения. Он нужен для перехватки исключения, которое кидает оператор throw.

1. Какие формы может иметь спецификация исключения в секции ловушке, В каких ситуациях используются эти формы?

Ответ: Спецификация исключения может быть выражена типом и именем, типом и остальными исключениями. Первая и вторая формы перехватывают исключения выбранного типа. Третья форма обрабатывает все исключения, которые не попадают по выделенный тип исключения.

1. Какой стандартный класс можно использовать для создания собственной иерархии исключений?

Ответ: Для создания собственной иерархии исключений можно использовать класс exception в качестве базового класса.

1. Каким образом можно создать собственную иерархию исключений?

Ответ: либо описав свои собственные классы, либо использовать класс exception в качестве базового класса.

1. Если спецификация исключений имеет вид: void f1()throw(int,double); то какие исключения может порождать функция f1()?

Ответ: int и double

1. Если спецификация исключений имеет вид: void f1()throw(); то какие исключения может порождать функция f1()?

Ответ: любые

1. В какой части программы может генерироваться исключение?

Ответ: внутри оператора try