МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ПНИПУ)

Факультет: Электротехнический

Кафедра: «Информационные технологии и автоматизированные системы» (ИТАС)

Направление: Разработка информационных систем (РИС)

**О Т Ч Ё Т**   
**о лабораторной работе №9**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Обработка исключительных ситуаций»

Выполнил:

Студент группы РИС-24-3б

Суяргулов А.О.

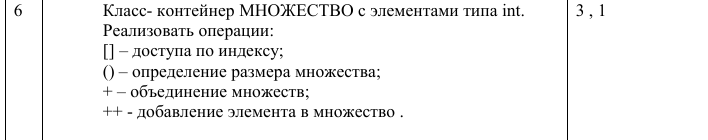
Проверил:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2025

ВВЕДЕНИЕ



ОСНОВНАЯ ЧАСТЬ

**Диаграмма классов:**

**Код:**

**Main (1 способ):**

#include <iostream>

#include <vector>

using namespace std;

class MySet {

private:

vector<int> elements; // Храним элементы в векторе

public:

// Добавление элемента в множество

void add(int num) {

// Проверяем, есть ли уже такой элемент

bool alreadyExists = false;

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

if (elements[i] == num) {

alreadyExists = true;

break;

}

}

// Если элемента нет - добавляем

if (!alreadyExists) {

elements.push\_back(num);

}

}

// Оператор [] - доступ по индексу

int& operator[](int index) {

if (index < 0 || index >= elements.size()) {

throw "Error: Index out of range!"; // Просто бросаем строку

}

return elements[index];

}

// Оператор () - получить размер множества

int operator()() {

return elements.size();

}

// Оператор + - объединение двух множеств

MySet operator+(MySet other) {

MySet result = \*this; // Копируем текущее множество

// Добавляем все элементы из второго множества

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

result.add(other.elements[i]);

}

return result;

}

// Префиксный ++ (добавляем максимальный элемент + 1)

MySet& operator++() {

if (elements.empty()) {

add(0); // Если множество пустое, добавляем 0

}

else {

// Находим максимальный элемент

int max = elements[0];

for (int i = 1; i < elements.size(); i++) {

if (elements[i] > max) {

max = elements[i];

}

}

add(max + 1); // Добавляем max + 1

}

return \*this;

}

// Постфиксный ++

MySet operator++(int) {

MySet temp = \*this;

++(\*this); // Используем префиксную версию

return temp;

}

// Вывод множества на экран

void print() {

cout << "{ ";

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

cout << elements[i] << " ";

}

cout << "}" << endl;

}

};

int main() {

try {

// Создаем первое множество

MySet set1;

set1.add(1);

set1.add(2);

set1.add(3);

cout << "Set1: ";

set1.print();

// Создаем второе множество

MySet set2;

set2.add(3);

set2.add(4);

set2.add(5);

cout << "Set2: ";

set2.print();

// Объединение множеств

MySet set3 = set1 + set2;

cout << "Union (set1 + set2): ";

set3.print();

// Размер множества

cout << "Size of set1: " << set1() << endl;

// Доступ по индексу

cout << "set1[1] = " << set1[1] << endl;

set1[1] = 10;

cout << "After set1[1] = 10: ";

set1.print();

// Префиксный ++

++set1;

cout << "After ++set1: ";

set1.print();

// Постфиксный ++

set2++;

cout << "After set2++: ";

set2.print();

// Проверка ошибки

cout << "Trying to access set1[100]..." << endl;

cout << set1[100] << endl;

}

catch (const char\* error) {

cout << error << endl;

}

return 0;

}

**Main(второй способ):**

#include <iostream>

#include <vector>

using namespace std;

// Простой класс для ошибки выхода за границы

class OutOfRangeError {

public:

string message;

OutOfRangeError() : message("Index is out of range!") {}

};

// Простой класс для ошибки пустого множества

class EmptySetError {

public:

string message;

EmptySetError() : message("Set is empty!") {}

};

class MySet {

private:

vector<int> elements;

public:

void add(int num) {

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

if (elements[i] == num) {

return; // Элемент уже есть - ничего не делаем

}

}

elements.push\_back(num);

}

int& operator[](int index) {

if (index < 0 || index >= elements.size()) {

throw OutOfRangeError(); // Бросаем объект ошибки

}

return elements[index];

}

int operator()() {

return elements.size();

}

MySet operator+(MySet other) {

MySet result = \*this;

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

result.add(other.elements[i]);

}

return result;

}

MySet& operator++() {

if (elements.empty()) {

throw EmptySetError();

}

int max = elements[0];

for (int i = 1; i < elements.size(); i++) {

if (elements[i] > max) {

max = elements[i];

}

}

add(max + 1);

return \*this;

}

MySet operator++(int) {

MySet temp = \*this;

++(\*this);

return temp;

}

void print() {

cout << "{ ";

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

cout << elements[i] << " ";

}

cout << "}" << endl;

}

};

int main() {

try {

MySet set1;

set1.add(1);

set1.add(2);

set1.add(3);

cout << "Set1: ";

set1.print();

MySet set2;

set2.add(3);

set2.add(4);

set2.add(5);

cout << "Set2: ";

set2.print();

MySet set3 = set1 + set2;

cout << "Union (set1 + set2): ";

set3.print();

cout << "Size of set1: " << set1() << endl;

cout << "set1[1] = " << set1[1] << endl;

set1[1] = 10;

cout << "After set1[1] = 10: ";

set1.print();

++set1;

cout << "After ++set1: ";

set1.print();

set2++;

cout << "After set2++: ";

set2.print();

// Проверка ошибки с пустым множеством

MySet emptySet;

cout << "Trying ++ on empty set..." << endl;

++emptySet;

}

catch (OutOfRangeError e) {

cout << "ERROR: " << e.message << endl;

}

catch (EmptySetError e) {

cout << "ERROR: " << e.message << endl;

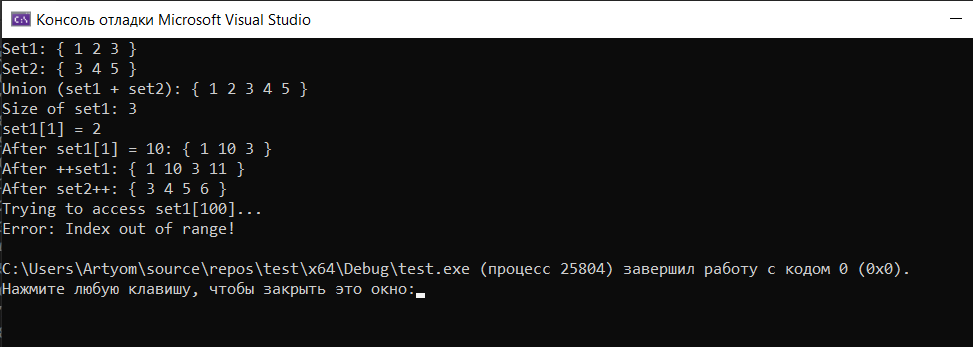
}

return 0;

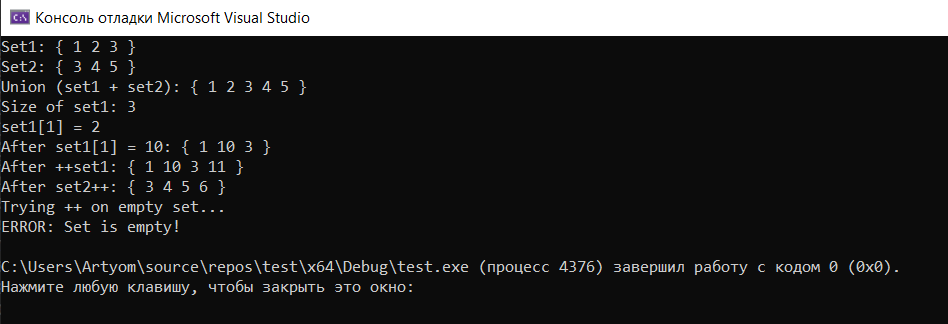
}

**Результат работы кода:**

**1 способ:**

****

**2 способ:**

****

**Контрольные вопросы:**

1. Что представляет собой исключение в C++?

Исключение в C++ - это способ обработки ошибок или нештатных ситуаций, возникающих во время выполнения программы. Это механизм для передачи управления от места возникновения ошибки к обработчику этой ошибки.

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

Исключения позволяют разделить вычислительный процесс на:

• Обнаружение ошибки (выброс исключения): Код, который обнаруживает нештатную ситуацию и сигнализирует о ней.

• Обработку ошибки (перехват и обработка исключения): Код, который реагирует на возникшую ошибку, пытаясь исправить ситуацию или корректно завершить программу.

• Преимущества:

\* Улучшение читаемости кода: Отделение логики обработки ошибок от основной логики программы делает код более понятным и простым в сопровождении.

\* Централизованная обработка ошибок: Позволяет обрабатывать ошибки в одном месте (блоке catch), что облегчает управление и изменение стратегии обработки ошибок.

\* Устойчивость программы: Предотвращает аварийное завершение программы в случае возникновения ошибки, позволяя выполнить корректное завершение или восстановить работоспособность.

3. Какой оператор используется для генерации исключительной ситуации?

Оператор throw используется для генерации исключительной ситуации.

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

Контролируемый блок (блок try) – это блок кода, в котором могут возникнуть исключения. Он нужен для того, чтобы указать компилятору, в каких частях программы нужно отслеживать возможные ошибки, которые могут привести к исключениям.

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

Секция-ловушка (блок catch) - это блок кода, который следует сразу за блоком try и предназначен для перехвата и обработки исключений, выброшенных в блоке try. Она нужна для обработки возникших исключительных ситуаций и предотвращения аварийного завершения программы.

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

Секция-ловушка (catch) может иметь разные формы в зависимости от типа исключения, которое нужно перехватить:

• catch (тип\_исключения имя\_переменной): Перехватывает исключение указанного типа. Используется, когда нужно обработать конкретный тип исключения. Например, catch (std::runtime\_error& err) перехватит исключение типа std::runtime\_error.

• catch (...): Перехватывает любое исключение, независимо от его типа. Используется как последняя ловушка, чтобы гарантированно обработать любое необработанное исключение. Важно: её следует ставить в конце списка catch блоков.

• Спецификация исключения в секции ловушке указывает, какой тип исключения будет обработан этим catch-блоком. Это позволяет организовать иерархическую обработку исключений.

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

Стандартный класс std::exception можно использовать в качестве базового класса для создания собственной иерархии исключений. Также можно использовать другие стандартные классы исключений, такие как std::runtime\_error, std::logic\_error и другие, в зависимости от типа ошибки.

8. Каким образом можно создать собственную иерархию исключений?

Собственная иерархия исключений создается путем наследования от базового класса исключений (обычно std::exception) и определения собственных классов исключений для конкретных ситуаций. Например:

#include <exception>

#include <string>

class MyException : public std::exception {

public:

MyException(const std::string& message) : message\_(message) {}

const char\* what() const noexcept override { return message\_.c\_str(); }

private:

std::string message\_;

};

class MySpecificException : public MyException {

public:

MySpecificException(const std::string& message) : MyException(message) {}

};

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

Функция f1 с такой спецификацией исключений может выбрасывать исключения типа int и double. Все другие исключения, если они возникнут, могут привести к вызову std::unexpected, что обычно завершает программу. Однако, начиная со стандарта C++17, спецификации throw() стали deprecated и в большинстве случаев игнорируются компилятором.

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

Функция f1 с такой спецификацией исключений не должна выбрасывать никаких исключений. Если исключение возникнет внутри f1, то будет вызвана функция std::unexpected, которая по умолчанию завершает программу. Эта спецификация также стала deprecated в C++17. Начиная с C++11 для указания, что функция не выбрасывает исключений, следует использовать noexcept.

11. В какой части программы может генерироваться исключение?

Исключение может генерироваться в любой части программы, где может возникнуть нештатная ситуация, например, при делении на ноль, при попытке доступа к недопустимой памяти, при ошибке ввода-вывода и т.п. Обычно исключения генерируются в функциях, которые выполняют операции, подверженные ошибкам.

**Ссылка на GitHub:**

<https://github.com/Drazda1488/oop_9lab>