МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Практична робота №1**

**з дисципліни «Сучасні освітні технології »   
на тему:**

**«**Сучасне системне програмне забезпечення»

Виконав:

студент гр. ПЗ2421

Кулик С. В.

Прийняв:

Жеваго О. О.

Дніпро, 2025

Тема. Структурна обробка виключень.

Мета. Набути практичних навичок обробки виключень за допомогою structured exception handling (SEH)

**Постановка задачі**

Написати програму для обробки виключень за індивідуальним завданням, виконати її налагодження. Передбачити для програми вхідні дані, які сприяють генерації виключення, та такі, при яких це не можливо. Передбачити фільтр EXCEPTION\_CONTINUE\_SEARCH та обробку завершення для всіх інших виключень.

Варіант 8: EXCEPTION\_INVALID\_HANDLE

**Текст програми**

#include <windows.h>

#include <excpt.h>

#include <iostream>

#include <iomanip>

using namespace std;

// Функція для демонстрації обробки завершення

void DemoTerminationHandling() {

cout << "\n=== ДЕМОНСТРАЦІЯ ОБРОБКИ ЗАВЕРШЕННЯ ===" << endl;

\_\_try {

cout << "Початок захищеного блоку (try)" << endl;

// Створюємо недійсний дескriptor

HANDLE invalidHandle = (HANDLE)0x12345678;

cout << "Спроба роботи з недійсним дескриптором..." << endl;

// Викликаємо операцію, що може призвести до виключення

DWORD result = WaitForSingleObject(invalidHandle, 1000);

cout << "Операція завершена успішно" << endl;

}

\_\_finally {

cout << "Блок завершення (finally) виконується" << endl;

// Перевіряємо, чи було аномальне завершення

if (AbnormalTermination()) {

cout << "Аномальне завершення блоку try!" << endl;

}

else {

cout << "Нормальне завершення блоку try" << endl;

}

}

}

// Функція для демонстрації обробки виключень

void DemoExceptionHandling() {

cout << "\n=== ДЕМОНСТРАЦІЯ ОБРОБКИ ВИКЛЮЧЕНЬ ===" << endl;

\_\_try {

cout << "Початок захищеного блоку для обробки виключень" << endl;

// Створюємо заведомо недійсний дескриптор

HANDLE invalidHandle = (HANDLE)0xFFFFFFFF;

cout << "Спроба закрити недійсний дескриптор..." << endl;

// Викликаємо операцію, що призведе до EXCEPTION\_INVALID\_HANDLE

if (!CloseHandle(invalidHandle)) {

cout << "CloseHandle повернув помилку, але виключення не згенеровано" << endl;

// Генеруємо виключення вручну для демонстрації

cout << "Генеруємо EXCEPTION\_INVALID\_HANDLE вручну..." << endl;

RaiseException(EXCEPTION\_INVALID\_HANDLE, 0, 0, NULL);

}

cout << "Цей рядок не повинен виконуватися після виключення" << endl;

}

\_\_except (GetExceptionCode() == EXCEPTION\_INVALID\_HANDLE ?

EXCEPTION\_EXECUTE\_HANDLER : EXCEPTION\_CONTINUE\_SEARCH) {

DWORD exceptionCode = GetExceptionCode();

cout << "Обробка виключення!" << endl;

cout << "Код виключення: 0x" << hex << uppercase << exceptionCode << dec << endl;

cout << "Тип: EXCEPTION\_INVALID\_HANDLE" << endl;

cout << "Виключення успішно оброблено" << endl;

}

cout << "Продовження виконання після обробки виключення" << endl;

}

// Функція для демонстрації вкладених обробників

void DemoNestedHandlers() {

cout << "\n=== ДЕМОНСТРАЦІЯ ВКЛАДЕНИХ ОБРОБНИКІВ ===" << endl;

\_\_try {

cout << "Зовнішній try блок" << endl;

\_\_try {

cout << "Внутрішній try блок" << endl;

// Генеруємо виключення

cout << "Генерація EXCEPTION\_INVALID\_HANDLE у вкладеному блоці..." << endl;

RaiseException(EXCEPTION\_INVALID\_HANDLE, 0, 0, NULL);

}

\_\_finally {

cout << "Внутрішній finally блок виконується" << endl;

if (AbnormalTermination()) {

cout << "Аномальне завершення внутрішнього блоку" << endl;

}

}

cout << "Цей рядок не повинен виконуватися" << endl;

}

\_\_except (GetExceptionCode() == EXCEPTION\_INVALID\_HANDLE ?

EXCEPTION\_EXECUTE\_HANDLER : EXCEPTION\_CONTINUE\_SEARCH) {

cout << "Обробка виключення у зовнішньому блоці" << endl;

cout << "Код виключення: 0x" << hex << uppercase << GetExceptionCode() << dec << endl;

}

}

// Функція для демонстрації фільтру EXCEPTION\_CONTINUE\_SEARCH

void DemoContinueSearch() {

cout << "\n=== ДЕМОНСТРАЦІЯ EXCEPTION\_CONTINUE\_SEARCH ===" << endl;

\_\_try {

\_\_try {

cout << "Генерація іншого типу виключення (не INVALID\_HANDLE)..." << endl;

// Генеруємо інший тип виключення

RaiseException(EXCEPTION\_ACCESS\_VIOLATION, 0, 0, NULL);

}

\_\_except (GetExceptionCode() == EXCEPTION\_INVALID\_HANDLE ?

EXCEPTION\_EXECUTE\_HANDLER : EXCEPTION\_CONTINUE\_SEARCH) {

cout << "Цей обробник не повинен виконуватися для ACCESS\_VIOLATION" << endl;

}

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER) {

cout << "Загальний обробник спрацював" << endl;

cout << "Код виключення: 0x" << hex << uppercase << GetExceptionCode() << dec << endl;

cout << "Це демонструє роботу EXCEPTION\_CONTINUE\_SEARCH" << endl;

}

}

// Функція для тестування з коректними даними

void TestWithValidData() {

cout << "\n=== ТЕСТУВАННЯ З КОРЕКТНИМИ ДАНИМИ ===" << endl;

\_\_try {

cout << "Створення коректного дескриптора події..." << endl;

// Створюємо коректний дескриптор

HANDLE validHandle = CreateEvent(NULL, TRUE, FALSE, L"TestEvent");

if (validHandle != NULL) {

cout << "Дескриптор створено успішно: 0x" << hex << uppercase << validHandle << dec << endl;

// Виконуємо коректні операції

cout << "Очікування на подію (таймаут 100мс)..." << endl;

DWORD result = WaitForSingleObject(validHandle, 100);

cout << "Результат очікування: " << result << endl;

cout << "Закриття дескриптора..." << endl;

if (CloseHandle(validHandle)) {

cout << "Дескриптор закрито успішно" << endl;

}

}

else {

cout << "Помилка створення дескриптора" << endl;

}

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER) {

cout << "Несподіване виключення при роботі з коректними даними!" << endl;

cout << "Код: 0x" << hex << uppercase << GetExceptionCode() << dec << endl;

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

try {

// 1. Демонстрація обробки завершення

DemoTerminationHandling();

// 2. Демонстрація обробки виключень

DemoExceptionHandling();

// 3. Демонстрація вкладених обробників

DemoNestedHandlers();

// 4. Демонстрація EXCEPTION\_CONTINUE\_SEARCH

DemoContinueSearch();

// 5. Тестування з коректними даними

TestWithValidData();

}

catch (...) {

cout << "Перехоплено C++ виключення" << endl;

}

cout << "\n=================================================" << endl;

cout << "Програма завершена" << endl;

system("pause");

return 0;

}

**Результат роботи**

**=== ДЕМОНСТРАЦІЯ ОБРОБКИ ЗАВЕРШЕННЯ ===**

**Початок захищеного блоку (try)**

**Спроба роботи з недійсним дескриптором...**

**Операція завершена успішно**

**Блок завершення (finally) виконується**

**Нормальне завершення блоку try**

**=== ДЕМОНСТРАЦІЯ ОБРОБКИ ВИКЛЮЧЕНЬ ===**

**Початок захищеного блоку для обробки виключень**

**Спроба закрити недійсний дескриптор...**

**CloseHandle повернув помилку, але виключення не згенеровано**

**Генеруємо EXCEPTION\_INVALID\_HANDLE вручну...**

**Обробка виключення!**

**Код виключення: 0xC0000008**

**Тип: EXCEPTION\_INVALID\_HANDLE**

**Виключення успішно оброблено**

**Продовження виконання після обробки виключення**

**=== ДЕМОНСТРАЦІЯ ВКЛАДЕНИХ ОБРОБНИКІВ ===**

**Зовнішній try блок**

**Внутрішній try блок**

**Генерація EXCEPTION\_INVALID\_HANDLE у вкладеному блоці...**

**Внутрішній finally блок виконується**

**Аномальне завершення внутрішнього блоку**

**Обробка виключення у зовнішньому блоці**

**Код виключення: 0xC0000008**

**=== ДЕМОНСТРАЦІЯ EXCEPTION\_CONTINUE\_SEARCH ===**

**Генерація іншого типу виключення (не INVALID\_HANDLE)...**

**Загальний обробник спрацював**

**Код виключення: 0xC0000005**

**Це демонструє роботу EXCEPTION\_CONTINUE\_SEARCH**

**=== ТЕСТУВАННЯ З КОРЕКТНИМИ ДАНИМИ ===**

**Створення коректного дескриптора події...**

**Дескриптор створено успішно: 0x00000000000000B4**

**Очікування на подію (таймаут 100мс)...**

**Результат очікування: 258**

**Закриття дескриптора...**

**Дескриптор закрито успішно**

**=================================================**

**Програма завершена**

**Висновки щодо основних особливостей структурної обробки**

1. **Гарантоване виконання коду**: Механізм \_\_try/\_\_finally забезпечує обов'язкове виконання блоку завершення незалежно від способу виходу з захищеного блоку.
2. **Гнучка фільтрація виключень**: SEH дозволяє селективно обробляти різні типи виключень за допомогою фільтрів, що повертають:
   * EXCEPTION\_EXECUTE\_HANDLER - для обробки виключення
   * EXCEPTION\_CONTINUE\_SEARCH - для передачі обробки на вищий рівень
   * EXCEPTION\_CONTINUE\_EXECUTION - для продовження з точки виключення
3. **Ієрархічна обробка**: Можливість створення вкладених обробників дозволяє реалізувати складну логіку обробки виключень з різними рівнями деталізації.
4. **Інформативність**: Функції GetExceptionCode() та GetExceptionInformation() надають детальну інформацію про виключення для прийняття рішень щодо обробки.
5. **Інтеграція з Windows API**: SEH тісно інтегрований з Windows API та дозволяє обробляти як апаратні, так і програмні виключення.
6. **Управління ресурсами**: Комбінація \_\_try/\_\_finally забезпечує надійне звільнення ресурсів навіть при виникненні виключень.
7. **Продуктивність**: SEH має мінімальні накладні витрати при нормальному виконанні програми, оскільки обробка виключень виконується безпосередньо операційною системою.

Структурна обробка виключень є потужним механізмом для створення надійних і стійких до помилок програм під Windows, що особливо важливо для системного програмування.