Лекція 4

ОБРОБКА ВИНЯТКОВИХ СИТУАЦІЙ

Лекція 4. Обробка виняткових ситуацій

План

1. Загальне поняття про виняткові ситуації.
2. Перехоплення та обробка виняткових ситуацій у C++.
3. Створення винятків користувача.
4. Стандартні винятки.

1. Загальне поняття про виняткові ситуації

Виняткова ситуація (ВС) або **виняток** – це певна подія, що призвела до збою у роботі програми. Внаслідок виникнення ВС програма не може коректно продовжити своє виконання. Інакше кажучи, ВС – це порушення природного (нормального) ходу виконання алгоритму.

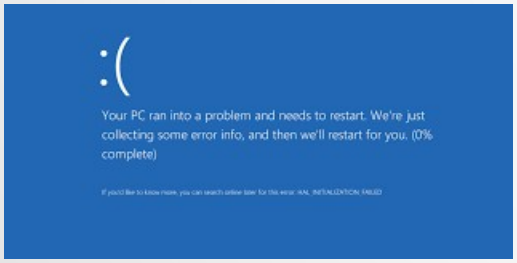
Існують два основних типи ВС:

- **апаратні** (генеруються процесором);
- **програмні** (генеруються операційною системою чи прикладними програмами).

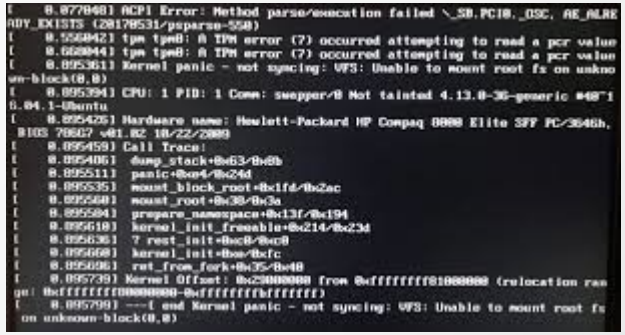
1. Загальне поняття про виняткові ситуації

Апаратні ВС виникають, як правило, при різних збоях обладнання, таких, наприклад, як:

- ділення на нуль;
- переповнення;
- звернення до некоректної адреси пам'яті;
- розрив мережевого з'єднання;
- відключення живлення тощо.



BSOD – «синій екран смерті» у Windows

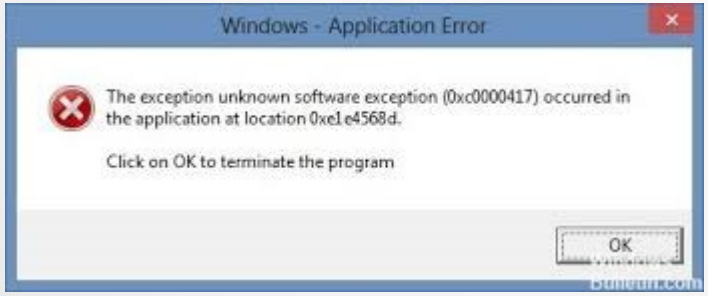


Kernel panic у Linux

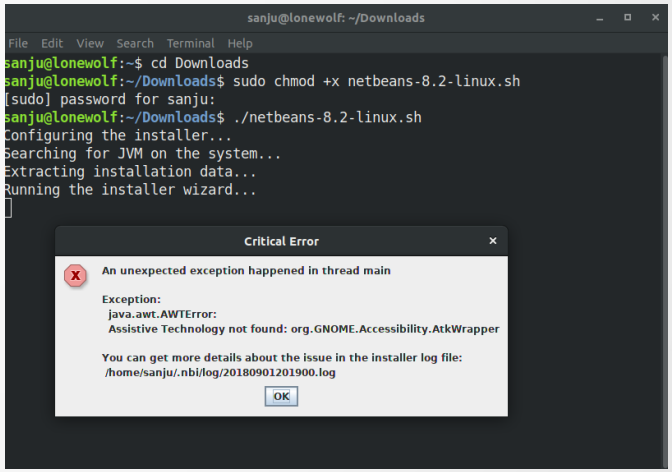
1. Загальне поняття про виняткові ситуації

Програмні ВС явно генеруються ОС або прикладною програмою, коли вони виявляють аномальну ситуацію, що виникла в процесі їхньої роботи. Наприклад:

- нестача оперативної пам'яті;
- вихід за межі масиву;
- спроба обчислення кореня з негативного числа;
- некоректні вихідні дані і т. п.



ВС, що ініціюється Windows під час некоректної роботи програми з пам'яттю



ВС, що генерується Linux при помилці встановлення програми

1. Загальне поняття про виняткові ситуації

Розглянемо наступний приклад.

```
// Функція, що реалізує ділення двох чисел
double div(double top, double bot)
{
    return top / bot;
}
```

Очевидно, що у такому вигляді функція div() є небезпечною, тому що при нульовому значенні параметра bot відбудеться ділення на нуль, що викличе відповідне апаратне переривання з подальшим **аварійним завершенням програми**.

Програміст, який реалізує, наприклад, математичну бібліотеку, знає про «вузькі місця», де можливі виникнення помилок, але поняття не має, як їх обробляти (це повинен робити користувач бібліотеки).

Примітка. Наявність у програмі необроблених ВС є не просто поганим тоном, а й погано характеризує розробника в очах замовника!

1. Загальне поняття про виняткові ситуації

Стандартним підходом до вирішення цієї проблеми є використання деякої глобальної змінної, якій присвоюється код помилки, у разі її виникнення. Наприклад.

```
// Коди помилок
enum { NO_ERR = 0, DIV_ZERO = 1, … };

// Глобальна змінна, що містить код останньої помилки
int errno = NO_ERR;

// ...
double div(double top, double bot)
{
	errno = NO_ERR;
	if (bot == 0)
	{
		errno = DIV_ZERO;
		return 0;
	}
	return top / bot;
}
```

1. Загальне поняття про виняткові ситуації

Тоді порядок використання функції div() має бути таким.

```
// ...
double a, b, c;

cin >> a >> b;
c = div(a, b);
if (errno == DIV_ZERO)
{
      // Обробка помилки поділу на нуль
      cerr << "Devide by zero!" << endl;
      // ...
}
// …
```

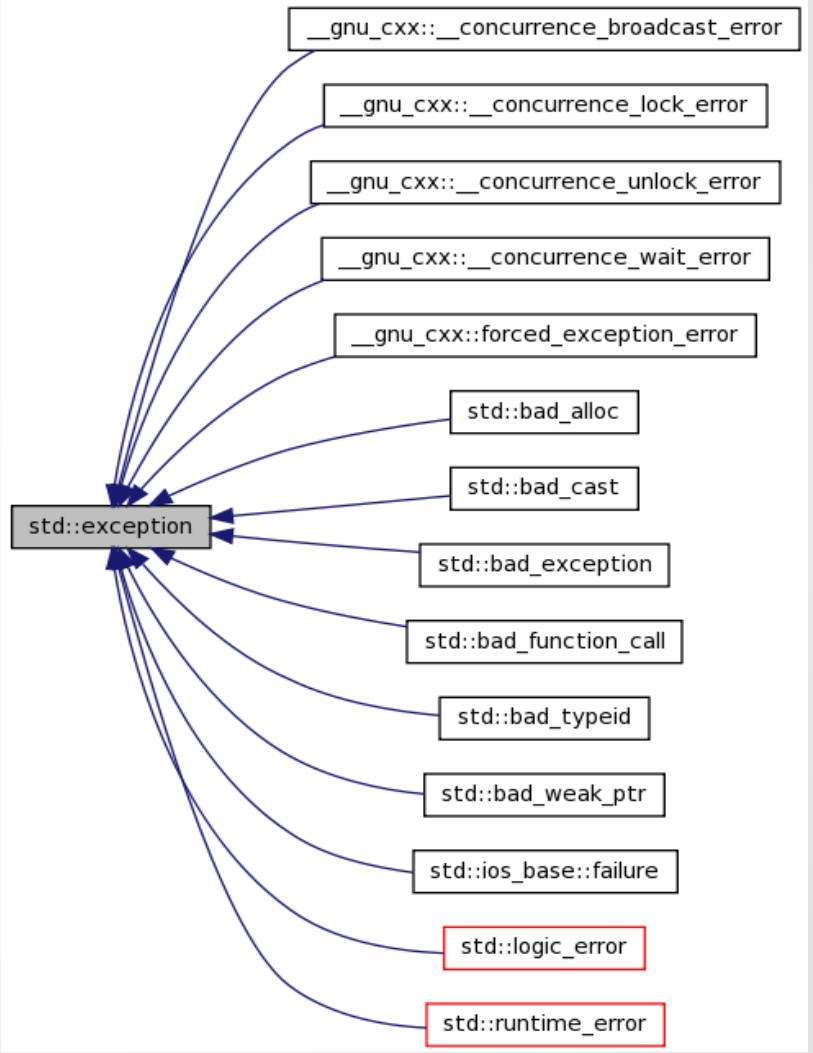
Такий підхід до обробки помилок реалізований, наприклад, у мові С. Очевидно, що він є незручним, тому що велика кількість перевірок на помилки істотно збільшує розмір програмного коду і робить його менш читальним.

2. Перехоплення та обробка виняткових ситуацій в C++

У мові програмування C++ для роботи з ВС використовується поняття винятку.

Виняток – це змінна (об'єкт деякого класу чи значення базового типу), яка описує конкретну ВС та відповідним чином обробляється.

Програміст може створити свою власну систему опису ВС (наприклад, кваліфікувати різні типи помилок цілими значеннями), так і використовувати спеціалізовані класи стандартної бібліотеки C++ (похідні від класу **std::exception**).



2. Перехоплення та обробка виняткових ситуацій в C++

C++ підтримує механізми генерації, перехоплення та обробки ВС. Для цього у C++ використовується конструкція **try…catch**, що має таку форму:

```
try
{
   // Код, де можливе виникнення винятку
   // ...
}
catch (type1 arg1)
{
  // Код, що обробляє ВС типу type1 (дані про ВС знаходиться в arg1)
  // ...
}
...
catch (typeN argN)
{
  // Код, що обробляє ВС типу typeN
  // ...
}
```

2. Перехоплення та обробка виняткових ситуацій в C++

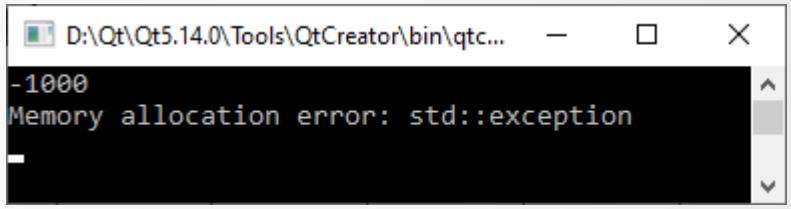
Розглянемо приклад обробки ВС засобами стандартної бібліотеки C++.

```
#include <iostream>
#include <exception>

int main()
{
    int size, *array = nullptr;

    std::cin >> size; // Запит розміру масиву
    try
    {
        array = new int[size]; // Спроба виділення пам'яті
    }
    catch (std::exception e)
    {
        // Обробка ВС
        std::cerr << "Memory allocation error: " << e.what() << std::endl;
        return 1;
    }
    delete [] array; // Звільнення пам'яті
    return 0;
}
```

Приклад роботи програми



2. Перехоплення та обробка виняткових ситуацій в C++

Слід зазначити, що у наведеному вище прикладі буде здійснено перехоплення тільки ВС, тип яких відповідає std::exception. Всі інші помилки будуть проігноровані. Якщо, наприклад, у блоці try {} відбудеться поділ на нуль, такий тип ВС оброблюватися не буде.
Для виправлення цієї ситуації використовується варіант catch(…) {}, у якому перехоплюються всі типи ВС, щоправда, без можливості ідентифікувати причину виникнення.

```
try
{
   // Код, де можливе виникнення ВС
   // ...
}
catch (std::exception e)
{
   // Стандартний обробник
   // ...
}
catch (...)
{
   // Обробник ВС за замовчуванням
   // ...
}
```

2. Перехоплення та обробка виняткових ситуацій в C++

Для **генерації** (**збудження**) ВС у C++ використовується оператор **throw**. Загалом він має таку форму:

throw [<виняток>];

Тут необов'язковий параметр “виняток” містить певний вираз або змінну (у т.ч. це може бути об'єкт класу), що ідентифікують ВС. В результаті виконання throw генерується ВС, тип якого визначається значенням параметра "виняток" і яка має бути оброблена в блоці catch {}.

```
// …
try
{
	// …
	throw 1; // Генерація ВС (у catch буде передано ціле значення 1)
}
catch (int err)
{
	// Обробка...
}
```

Примітка. Виклик throw без параметрів поза блоком catch {} аварійно завершує роботу програми.

2. Перехоплення та обробка виняткових ситуацій в C++

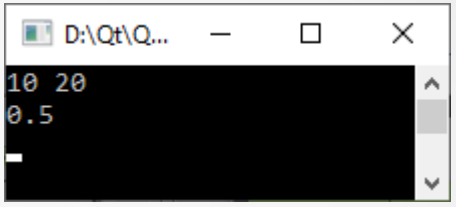
Розглянемо ще один простий приклад.

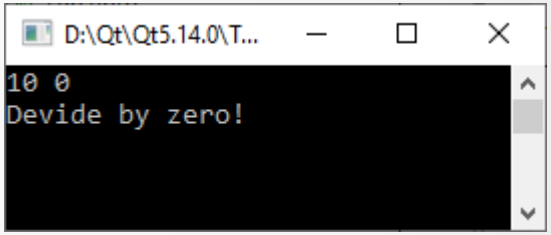
```
#include <iostream>

int main()
{
    double a, b, c;

    std::cin >> a >> b;
    try
    {
        if (b == 0)
            throw 1; // Генерація ВС
        c = a / b;
    }
    catch (...)
    {
        std::cerr << "Devide by zero!" << std::endl;
        return 1;
    }
    std::cout << c << std::endl;
    return 0;
}
```

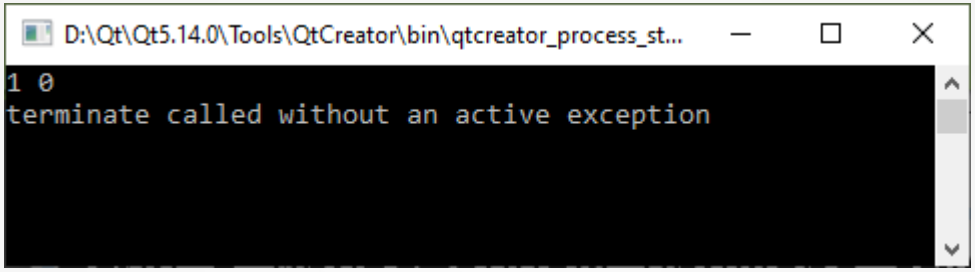
Результати роботи програми





2. Перехоплення та обробка виняткових ситуацій в C++

Якщо в наведеному прикладі замінити “throw 1;” на “throw;”, то вивід програми у разі помилки зміниться.



Таким чином, виклик throw без параметрів у блоці try {} призводить до аварійного завершення програми.

Як зазначалося, throw без параметрів можна викликати у блоці catch {}. У цьому випадку ВС буде передано для обробки вищому обробнику.

3. Створення винятків користувача

Розглянемо приклад створення класу, що реалізує винятки користувача.

```
// Коди помилок
enum { NO_ERR = 0, DIV_ZERO, NEG_ROOT };

// Реалізація винятків користувача
class Error
{
private:
    int err_code = NO_ERR; // Код помилки
public:
    Error(int e) : err_code(e) {} // Конструктор
    ~Error(void) {} // Деструктор
    void setError(int e) // Присвоєння коду помилки
    {
        err_code = e;
    }
    int getError(void) const // Повернення коду помилки
    {
        return err_code;
    }
```

```
    // Повернення помилки
    std::string sayError(void)
    {
        std::string ret;

        switch (err_code)
        {
            case NO_ERR:
                ret = "OK";
                break;
            case DIV_ZERO:
                ret = "Divide by zero";
                break;
            case NEG_ROOT:
                ret = "Negative root";
                break;
            default:
                ret = "Unknown error";
        }
        return ret;
    }
};
```

3. Створення винятків користувача

Використовувати вищенаведений клас можна, наприклад, таким чином.

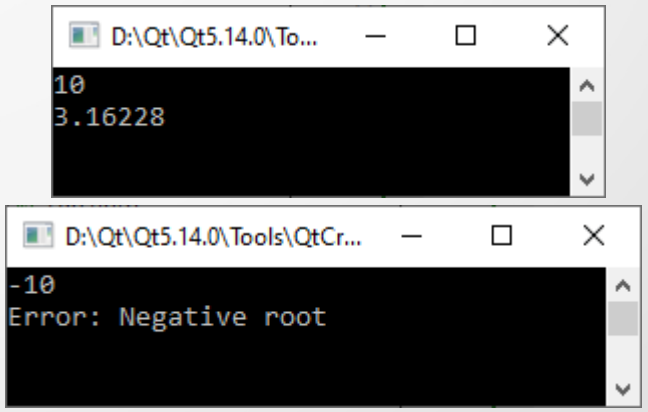
```
#include <iostream>
#include <cmath>

// Заголовний файл,
// у якому описано клас Error
#include "error.h"

int main()
{
    double val,
           res;

    std::cin >> val;
    try
    {
        if (val < 0)
            // Генерація користувацького винятку
            throw Error(NEG_ROOT);
        res = sqrt(val);
    }
    catch (Error err)
    {
        // Виведення даних про помилку
        std::cerr << "Error: " <<
                     err.sayError() << std::endl;
        return 1;
    }
    std::cout << res << std::endl;
    return 0;
}
```

Результати роботи програми



4. Стандартні винятки

У стандартній бібліотеці мови C++ реалізовано безліч класів обробки винятків. Всі вони є похідними від класу **std::exception**, підключити опис якого до програми можна так:

```
#include <exception>
```

Клас std::exception крім конструктора, деструктора і оператора присвоєння містить віртуальний метод **what()**, що повертає рядок-пояснення про помилку.

Клас std::exception надає єдиний інтерфейс обробки помилок за допомогою оператора throw. Усі винятки, що генеруються стандартною бібліотекою, успадковуються від std:: exception.

Приклад його використання наведено на Слайді №11.

4. Стандартні винятки

Спадкоємцями exception є безліч класів, серед яких можна виділити такі:

- **logic_error** – повідомляє про помилки, які є наслідком неправильної логіки у межах програми;
- **runtime_error** – повідомляє про помилки, що виходять за рамки програми та важко передбачуваних (переповнення, зникнення точності тощо);
- **bad_function_call** – інформує про некоректні виклики функцій (наприклад, при неправильній адресації до них тощо);
- **bad_alloc** – генерується у разі, якщо сталася помилка виділення пам'яті;
- **bad_cast** – повідомляє про помилки, що виникають під час перетворення типів даних;
- **failure** – генерується при помилках вводу-виводу і т. п.

4. Стандартні винятки

Приклад обробки ВС, що виникають при файловому вводі-виводі.

```
#include <iostream>
#include <fstream>

int main ()
{
    std::ifstream file;

    // Налаштування параметрів ВС, що перехрґоплюються
    file.exceptions(std::ifstream::failbit | std::ifstream::badbit);
    try
    {
        file.open("test.txt");
        while (!file.eof())
            file.get();
        file.close();
    }
    catch (std::ifstream::failure e)
    {
        std::cerr << "Exception opening/reading/closing file: " << e.what() << std::endl;
        return 1;
    }
    return 0;
}
```

Виведення програми при спробі відкриття неіснуючого файлу