

Лекція 7 (2023)

Об'єктно-орієнтоване програмування

Лектор: *Розова Людмила Вікторівна*

План лекції 7

1

Типи помилок в програмах

2

Виключна ситуація та її обробка

3

Специфікації виключень

4

Виключення в конструкторах і деструкторах

5

Класи виключень

Матеріали курсу:

https://github.com/LRozova/OOP_ukr_2023

Типи помилок в програмах

У програмах на С++ можуть виникати такі типи помилок:

- **Синтаксичні** – це помилки в синтаксисі мови С++, виявляються компілятором;
- **Логічні** – це помилки побудови алгоритму, які важко виявити на етапі розробки програми. Ці помилки виявляються на етапі виконання під час тестування роботи програми;
- помилки **часу виконання**. Такі помилки виникають під час роботи програми. Помилки часу виконання можуть бути:
 - логічними помилками програміста;
 - помилками зовнішніх подій (наприклад, нестача оперативної пам'яті);
 - невірним введенням даних користувачем тощо.

У результаті виникнення помилки часу виконання програма призупиняє свою роботу.

Виключна ситуація

Для обробки помилки часу виконання застосовується механізм **обробки виключних ситуацій**.

Виключна ситуація – це подія, що призвела до збою в роботі програми. У результаті виникнення виключної ситуації програма не може коректно продовжити своє виконання. Можливі дії при цьому:

- ✓ перервати виконання програми;
- ✓ повернути значення, що означає «помилка»;
- ✓ вивести повідомлення про помилку в потік ***cerr*** і повернути програмі деяке прийнятне значення, яке дозволить їй продовжувати роботу;
- ✓ **згенерувати (викинути) виключення.**

Приклади дій у програмі, що можуть призвести до виникнення виключних ситуацій:

- ділення на нуль;
- нестача оперативної пам'яті при застосуванні оператора *new* для її виділення (або іншої функції);
- доступ до елемента масиву за його межами (помилковий індекс);
- переповнення значення для деякого типу;
- добування кореня з від'ємного числа;
- неможливість відкриття файлу;
- інші ситуації...

Обробка виключної ситуації

Механізм обробки виключних ситуацій:

- Функція, в якій виникла помилка, **генерує виключення** (використовується ключове слово ***throw*** з параметром - константою, змінною або об'єктом) .
- Відшукується відповідний **обробник** і йому передається управління
- Якщо обробник не знайдений, викликається стандартна функція **terminate()**, яка викликає функцію **abort()**

```
try {  
    // try-блок (блок коду, що підлягає перевірці на наявність помилок)  
    throw виключення; // генерування виключення  
}  
  
// блоки для обробки виключень різного типу  
catch (type1 argument1)  
{  
    // catch-блок (обробник виключення типа type1)  
}  
  
catch (type2 argument2)  
{  
    // catch-блок (обробник виключення типа type2)  
}  
  
...
```

Перехоплення виключень:

Коли за допомогою ***throw*** генерується виключення, то функції виконавчої бібліотеки:

1. створюють копію параметра ***throw*** у вигляді статичного об'єкта, який існує до тих пір, поки виключення не буде оброблено;
2. в пошуках підходящого обробника ***розкручують стек*** (пошук відповідного обробника, в зовнішніх частинах програми). При цьому викликаються деструктори локальних об'єктів, що виходять з області дії;
3. передають об'єкт і управління ***відповідному обробнику catch***, який має параметр, сумісний з типом з цим об'єктом.

Приклад 1.

```
#include <iostream>
using namespace std;
int main()
{ try
    {
        throw 1; // генерація виключення
        //throw "ERROR";
        //throw 1.1;
    }
    catch (int) // перехоплення виключення типа int
    {
        cerr << "We caught an int exception" <<endl;
        //cerr – включено в iostream, використовується для виведення
        //повідомлень про помилки в консоль (як cout).
        // це стандартний потік для помилок    }
```

```
catch (const char* str)
```

```
// перехопления исключения по константной ссылке
```

```
{ cerr << "We caught an exception of type  
    string with value: " << str<<endl;  
}
```

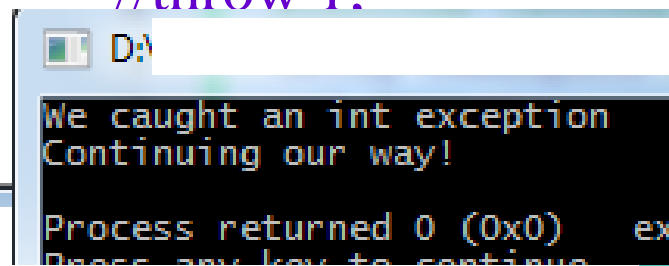
```
catch (...)
```

```
{ cerr << "All Errors" << endl;  
}
```

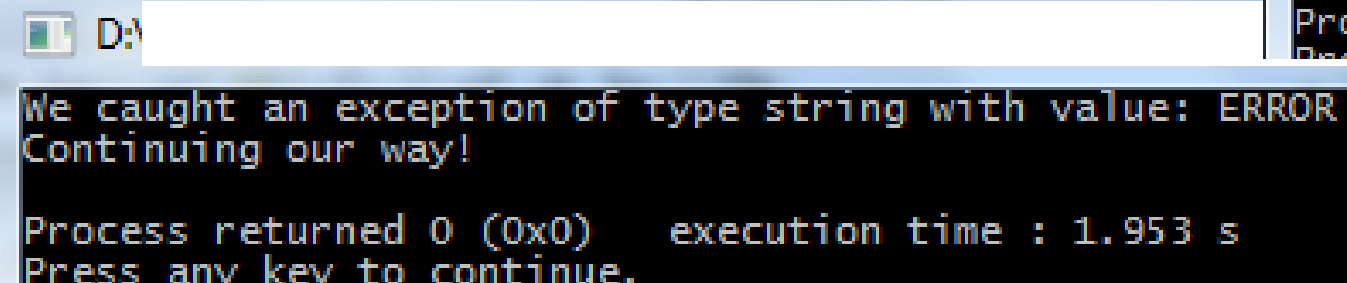
```
cout << "Continuing our way!\n";  
return 0;
```

```
//throw 1;
```

```
}  
//throw "ERROR";
```

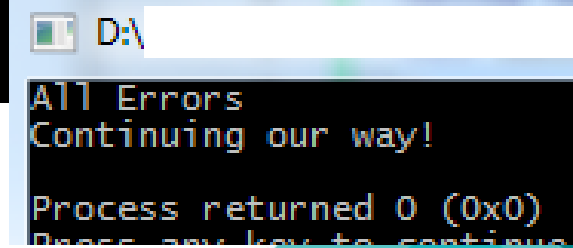


```
D:\>  
We caught an int exception  
Continuing our way!  
  
Process returned 0 (0x0)   ex  
Press any key to continue
```



```
D:\>  
We caught an exception of type string with value: ERROR  
Continuing our way!  
  
Process returned 0 (0x0)   execution time : 1.953 s  
Press any key to continue.
```

```
//throw 1.1;
```



```
D:\>  
All Errors  
Continuing our way!  
  
Process returned 0 (0x0)  
Press any key to continue
```

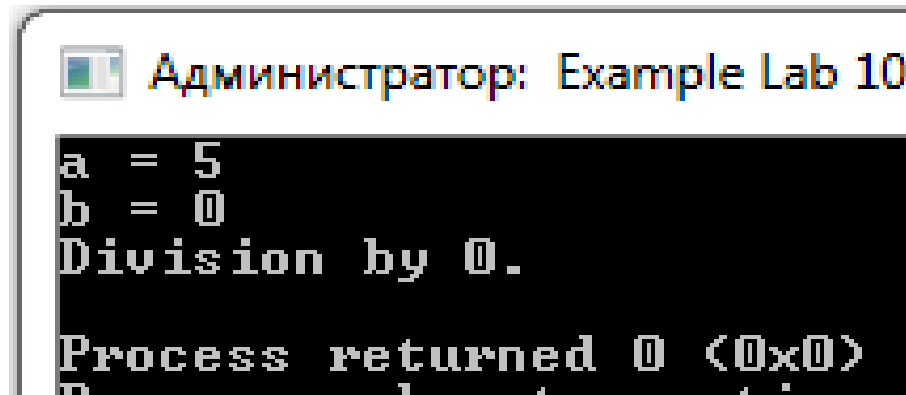
Приклад 2

```
#include <iostream>
using namespace std;

int main()
{   // обробка виразу  $\sqrt{a}/\sqrt{b}$ 
    double a, b;
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    double c;

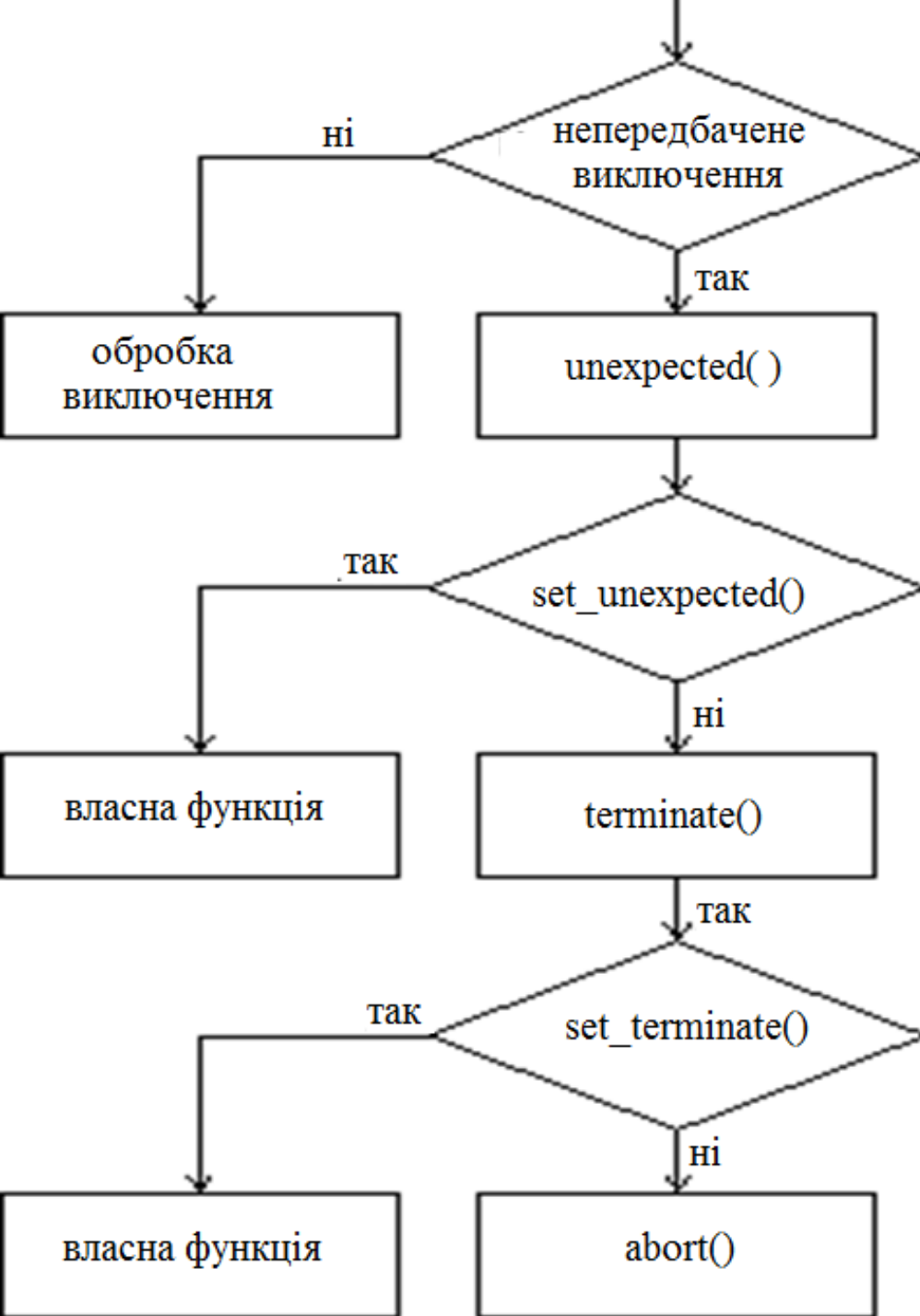
    try {
        // початок блока try
        if (b == 0)
            throw 1;
        if (b < 0)
            throw 2;
```

```
if (a < 0)
    throw 2;
c = sqrt(a) / sqrt(b);
cout << "c = " << c << endl;
}
catch (int e) // перехопления помилки
{
    if (e == 1)
        cout << "Division by 0." << endl;
    if (e == 2)
        cout << "Negative root." << endl;
}
}
```



```
Администратор: Example Lab 10
a = 5
b = 0
Division by 0.
Process returned 0 (0x0)
```

Алгоритм обробки виключень, у тому числі не перехоплених

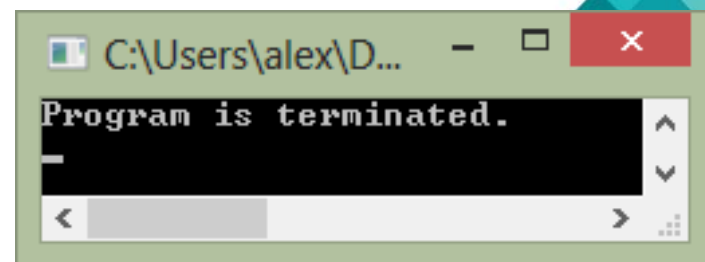


Заміна стандартного обробника

```

void SoftAbort() {
    cerr << "Program is terminated."
    << endl;
    exit(1);
}

int main() {
    set_terminate(SoftAbort);
    throw 5;
    return 0;
}
  
```



Специфікації виключень. Список виключень функцій¹⁴

Специфікації виключень або список виключень функцій - це механізм оголошення функцій із зазначенням того, чи буде функція генерувати виключення (і які саме) чи ні.

Це може бути корисно при визначенні необхідності переміщення виклику функції в блок try.

```
void f1() throw (int, const char*) { ... }
```

// можуть генеруватися виключення типу int, char*

```
void f2() throw (...) { ... }
```

// можуть генеруватися будь-які виключення

```
void f() throw() { ... } // не генеруються виключення
```

Але функція може породити виключення, яке не зазначено. Це призводить до виклику **unexpected()**, яка за замовчуванням викликає **terminate()**

Виключення в конструкторах

Виключення в конструкторах надають можливість передавати інформацію про проблеми, що виникли при створенні об'єкта

```
#include <iostream>
using namespace std;
```

```
class Vector
```

```
{ char name;
```

```
  int *p;
```

```
  int size;
```

```
public:
```

```
  Vector(char n,int s ):name(n),size(s)
```

```
  {if(size<0) throw "Size Error in Constructor";
```

```
    p=new int[size];
```

```
    if (!p) throw "Memory Error in Constructor";
```

```
    for(int i=0; i<size; i++)
```

```
      p[i]=i;
```

```
}
```

Приклад 3

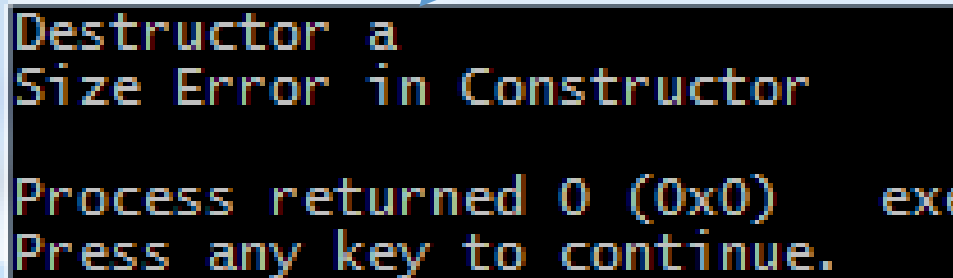
```

~Vector() {cout<<"Destructor " <<name<<endl; }
};

int main()
{ try
    {Vector a('a',5);
      Vector b('b',-1);
    }
  catch (const char *msg)
  {
      cerr<<msg<<endl;
  }
  return 0;
}

```

Спочатку спрацює
деструктор
для об'єкта **a**



```

Destructor a
Size Error in Constructor

Process returned 0 (0x0)   ex
Press any key to continue.

```


Виключення в деструкторах

Правило: Жодне з виключень не повинно покинути межі деструктора, бо це викличе функцію *terminate* ().

Тому:

- В найкращому випадку не треба генерувати виключення в деструкторах взагалі;
- Або інкапсулювати всі ці дії в деякому методі і викликати його з використанням *try / catch*:

```
T::Destroy() {  
    // метод, який може генерувати виключення  
}  
  
T::~~T() // деструктор  
{  
    try { Destroy(); }  
    catch (...) { /* ... */ }  
}
```

Класи виключень

В реальних програмах частіше використовуються **класи виключень** - це звичайний клас, який викидається в якості виключення.

При генерації виключень в операторі *throw* використовують або об'єкти цих класів, або анонімні екземпляри.

Можна передавати через параметри конструкторам цих класів і зберігати для подальшої обробки будь-яку інформацію про стан програми в момент генерації виключення.

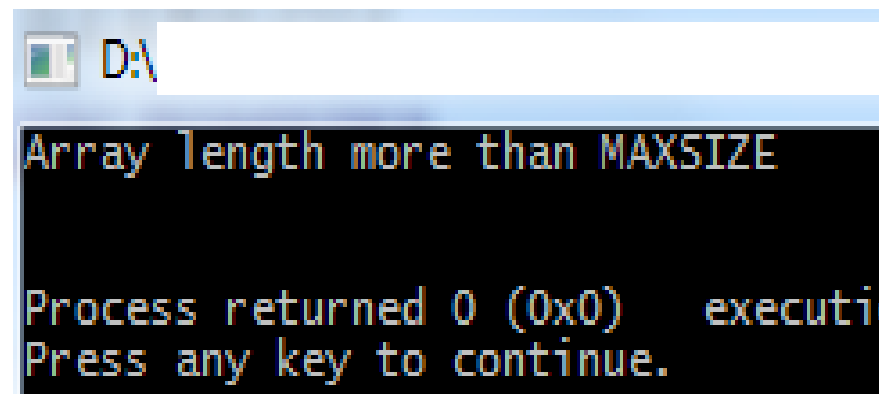
Приклад 4

```
const int MAX_SIZE=20;
class Error //клас помилка
{
    string str;
public:
    //конструктор, ініціює атрибут str повідомленням про помилку
    Error(string s)
    {
        str=s;
    }
    void what()
    {
        cout<<str<<endl; //виводить значення атрибута str
    }
};
```

```

class Array
{
    int size;
    int *a;
public:
    Array(int s)
    {
        if(s>MAX_SIZE)
            throw Error("Array length more than MAXSIZE\n");
        size=s;    a=new int [s];
        for(int i=0; i<size; i++) a[i]=i;
    };
int main()
{
    try
    {
        Array x(25);
    }
    catch(Error &e)
    {
        //об'єкти класа помилок приймають за посиланням
        e.what();
    }
    return 0;
}

```



A screenshot of a Windows command prompt window. The title bar shows the drive 'D:\'. The command prompt displays the following text: 'Array length more than MAXSIZE' on the first line, 'Process returned 0 (0x0) executi' on the second line, and 'Press any key to continue.' on the third line. The text is in a monospaced font with a color scheme of yellow and green on a black background.

Ієрархія класів виключень

Перевага використання класів виключень — це можливість створювати **ієрархію класів виключень**, для організування єдиного підходу обробки виключень програми.

Наприклад:

//Базовий клас обробки помилок

```
class MathError  
{...}
```

//Клас помилки переповнення

```
class Overflow : public MathError  
{...}
```

//Клас помилок ділення на нуль

```
class ZeroDivide: public Overflow  
{...}
```

Якщо додати деякий віртуальний метод в базовий клас, і додати його визначення в похідні класи, то це дозволить створити після блоку ***try*** єдиний обробник ***catch***, що приймає посилання на об'єкт базового класу

```
try
{ //генерація будь-яких виключень MathError, Overflow
  //або ZeroDivide
}
catch (MathError &er)
{ //обробка будь-якого виключення,
  //виклик віртуального методу або його визначень
}
```

Клас **exception** – клас виключень стандартної бібліотеки, є батьківським класом стандартних виключень.

`bad_alloc` - помилка при динамічному розподіленні пам'яті за допомогою `new`;

`bad_cast` - неправильне використання оператора `dynamic_cast`

`bad_typeid` - операція `typeid` не може визначити тип операнда;

`bad_exception` - при виконанні функції сталося неочікуване виключення;

exception

`bad_alloc`

`bad_cast`

`bad_typeid`

`bad_exception`

`logic_error`

`length_error`

`domain_error`

`out_of_range`

`invalid_argument`

`runtime_error`

`range_error`

`overflow_error`

`underflow_error`

`length_error` - спроба створення об'єкта, більшого, ніж максимальний розмір для даного типу;
`domain_error` - порушення внутрішніх умов перед виконанням дії;
`out_of_range` - спроба виклику функції з параметром, що не входять в допустимі значення;
`invalid_argument` - спроба виклику функції з невірним параметром;
`range_error` - неправильний результат обчислень при виконанні;
`overflow_error` - арифметичне переповнення;
`underflow_error` - зникнення порядку

Можна налаштувати обробник виключень `exception`, який буде ловити та обробляти виключення необхідним чином

Також можна створити свої власні класи-виключення, дочірні класу **exception**, і перевизначити віртуальний константний метод **what ()** (метод класу exception), додати інформацію про конкретну виключну ситуацію

Приклад 5

```
#include <iostream>
#include <string>
#include <exception>
using namespace std;

class ExcDiv: public exception
{private:
    string error;
public:
    ExcDiv(string msg) { error=msg; }
    //визначаємо метод what()
    const char* what() { return error.c_str(); }
    //можна додати в клас інформацію стосовно конкретного виключення
};
```

```
int div_ab(int a, int b)
{
    if(a<=0&&b<=0) throw ExcDiv("a<=0 and b<=0!");
    if (b==0) throw ExcDiv("b=0!");
    return a/b;
}
```

```
int main()
```

```
{ try
```

```
{ div_ab(0,0);
```

```
}
```

//спочатку треба ставити блоки catch для дочірніх класів

```
catch(ExcDiv& e)
```

```
{ cout<<e.what()<<endl;
```

```
}
```

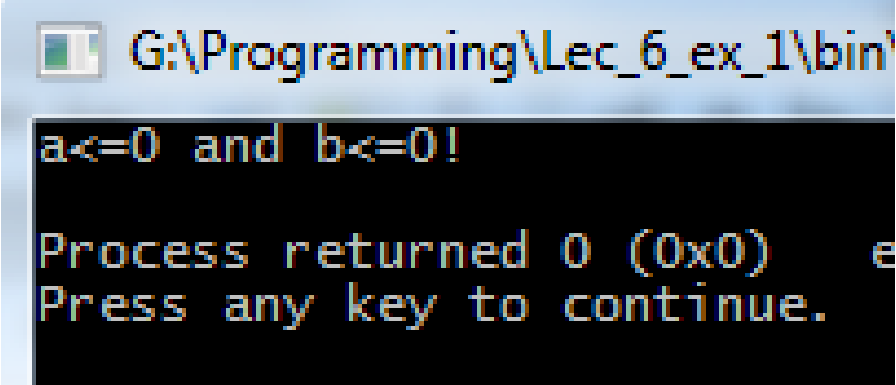
```
catch(exception & e)
```

```
{ cerr<<e.what()<<endl;
```

```
}
```

```
return 0;
```

```
}
```



```
G:\Programming\Lec_6_ex_1\bin'
a<=0 and b<=0!
Process returned 0 (0x0)
Press any key to continue.
```

Дякую за увагу!