

C++ Programming

13th Study: C++ exception handling

- try ... catch ...
- standard exception classes
- stack unwinding

C++ Korea 윤석준 (icysword77@gmail.com)



The background is a solid blue color with several white-outlined squares of various sizes scattered across it. Some squares are solid white, while others are just outlines. They are positioned in the corners and along the edges, creating a modern, geometric aesthetic.

try ... catch ...

C++ exception handling 사용법

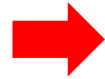
Exception

- Exception (예외) : 비정상적인 상황이 발생
- Exception handling (예외 처리) : 그 상황을 처리

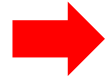
```
try
{
    if (/* Exception Condition */)
        throw new std::exception("Error Description");
}
catch (std::exception e)
{
    std::cout << "Exception : " << e.what() << std::endl;
}
```

try ... catch 사용법

예외가 발생할 수 있는 부분을 정의
`try { }`와 `catch { }`는 한쌍



예외를 발생시킴



`try` 안에서 발생한 예외 중 `e`를 `catch`



예외 처리

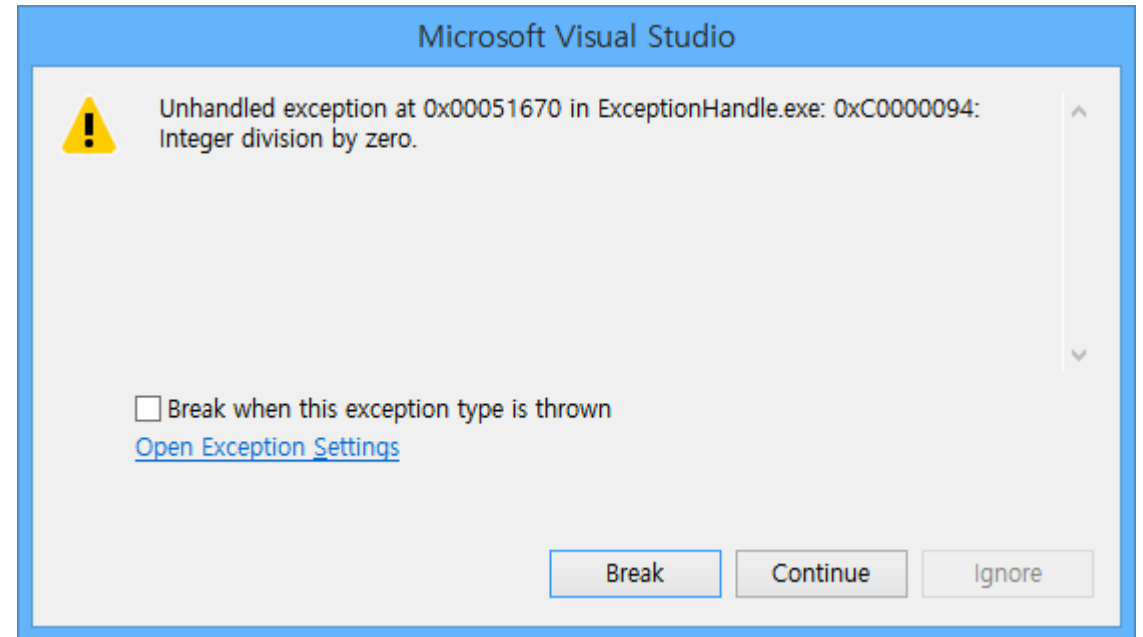


```
try
{
    if (/* Exception Condition */)
        throw new std::exception("Error Description");
}
catch (std::exception e)
{
    cout << "Exception : " << e.what() << endl;
}
```

예외 예제 division by zero

```
#include <iostream>

void main()
{
    int a = 9;
    int b = 0;
    int c = a / b;
    std::cout << c << std::endl;
}
```



Exception handling by program logic

```
void main()
```

```
{
```

```
int a = 9;  
int b = 0;  
int c = 0;
```

```
if (b != 0)  
{
```

```
    c = a / b;  
    std::cout << c << std::endl;
```

```
}
```

```
else
```

```
    std::cout << "Exception : Divide by zero" << std::endl;
```

```
}
```

원래 Code와 예외처리 부분이 뒤죽박죽

Exception handling by try ... catch

```
void main()
```

```
{
```

```
int a = 9;
```

```
int b = 0;
```

```
int c = 0;
```

좀 더 명확하게 구분가능

(이라고 주장하기엔 앞 Page 예제랑 별 차이가...)

```
try
```

```
{
```

```
if (b == 0) throw b;
```

```
c = a / b;
```

```
std::cout << c << std::endl;
```

```
}
```

```
catch (int divided)
```

```
{
```

```
std::cout << "Exception : Divide by " << divided << std::endl;
```

```
}
```

```
}
```

Exception handling by using function

```
void main()
{
```

```
    int a = 9;
    int b = 0;
    int c = 0;
```

```
template <typename T>
T Divide(T a, T b)
{
    if (b == 0) throw std::exception("Divide by zero");
    return a / b;
}
```

```
try
```

```
{
    c = Divide<int>(a, b);
    std::cout << c << std::endl;
}
```






```
catch (int divided)
```

```
{
    std::cout << "Exception : Divide by " << divided << std::endl;
}
```

```
}
```

함수를 사용하면서 예외 처리 할려면...

Declare Function w/t Exception List

- `void func(int a)`  모든 타입의 예외가 발생 가능하다.
- `void func(int a) throw(int);`  `int` 타입 예외를 던질 수 있다.
- `void func(int a) throw(char *, int);`  타입이 2가지 이상일 경우는 , 로 나열
- `void func(int a) throw();`  예외를 발생하지 않는다.
- `void func(int a) noexcept;`  C++11 Style

Standard exception classes

표준 예외 클래스

Standard exception classes

- 자주 발생하는 예외에 대해서 미리 정의한 예외 클래스가 있음
- 각각의 예외별로 다른 header 파일에 정의되어 있지만,
- `#include <exception>`하면 대부분의 경우 사용 가능

예제 : bad allocation

```
#include <iostream>
#include <new>

void main()
{
    char* ptr;
    try
    {
        ptr = new char[(~unsigned int((int)0) / 2) - 1];
        delete[] ptr;
    }
    catch (std::bad_alloc &ba)
    {
        std::cout << ba.what() << std::endl;
    }
}
```

대표적인 Standard exception classes

- `bad_alloc`: 메모리 할당 오류로서 `new` 연산에서 발생 `<new>`
- `bad_cast`: 형변환 오류로서 `dynamic_cast` 에서 발생 `<typeinfo.h>`
- `bad_type_id`: typeid에 대한 피 연산자가 널 포인터인 경우 발생
- `bad_exception`: 예기치 못한 예외로서 함수 발생 목록에 있지 않는 예외
- `bad_logic_error`: 클래스 논리 오류
- `invalid_argument`, `length_error`, `out_of_range` 의 기본 `<stdexcept>`
- `runtime_error`: 실행 오류로 `overflow_error`,
`underflow_error`의 기본 `<stdexcept>`

Stack Unwinding

스택 풀기

Stack Unwinding

- exception이 발생했는데, catch를 안해주면 ???
- 해당 함수를 호출한 곳으로 exception을 전달
- 언제까지 ? catch 를 만날때까지
- 함수 호출시 그 정보를 Stack에 저장하므로,
- 호출한 함수를 찾아 갈때 Stack에서 꺼내면서 찾아감

Stack Unwinding

- 예외 처리를 하기 위해 발생 시점부터 처리하는 위치까지 Stack에서 함수를 소멸시키면서 이동

```
void f1() { throw 0; }  
void f2() { f1(); }  
void f3() { f2(); }  
void f4() { f3(); }  
void main()  
{  
    try  
    {  
        f4();  
    }  
    catch (int e)  
    {  
        std::cout << e << std::endl;  
    }  
}
```

함수 호출



Call Stack	
Name	Language
ExceptionHandler.exe!f1() Line 3	C++
ExceptionHandler.exe!f2() Line 4	C++
ExceptionHandler.exe!f3() Line 5	C++
ExceptionHandler.exe!f4() Line 6	C++
ExceptionHandler.exe!main() Line 13	C++
[External Code]	
[Frames below may be incorrect and/or missing, no sym	

스택 풀기

