

# C++ Programming

## 13<sup>th</sup> Study: Exception Handling

- Exception handling in C++
- try ~ catch ~ throw
- Standard exception classes
- Stack unwinding

C++ Korea 옥찬호 (utilForever@gmail.com)



# Exception handling in C++

The process of responding to the occurrence, during computation, of exceptions often changing the normal flow of program execution

# Exception handling in C++

- 프로그램은 다양한 외부 환경과 함께 공존함
  - 운영체제, 네트워크, 파일 시스템, 여러 라이브러리, 사용자 입력 등
  - 외부 환경과의 연동하는 동안 무언가 문제가 생겼을 때 대응이 필요함
- 완벽한 프로그램이라도 예외적인 상황이 발생할 수 있음
  - 프로그램을 작성할 때는 항상 에러 처리 기능을 준비해서 포함해야 함
  - C는 특별히 에러 처리를 위한 도구들을 제공해주지 않음
  - Java는 익셉션을 에러 처리 메커니즘으로 사용하도록 강제함
  - C++는 두 언어의 중간 정도에 위치
    - 익셉션을 지원하지만, 사용 여부는 프로그래머의 선택사항

# Exception handling in C++

- 익셉션은 어떤 코드 영역에서 발생한 예외적인 상황이나 에러 조건을 일반적인 코드 처리 흐름을 따르지 않고도 다른 영역에 알려주기 위한 메커니즘
- 문제 상황을 맞이한 코드에서 던지면(throw)  
다른 코드에서 그 익셉션을 받아서(catch) 처리

던지면(throw)



익셉션(exception)

받아서(catch)



The background is a solid blue color. It is decorated with several squares of varying sizes and opacities. Some squares are solid blue, while others are white outlines. They are scattered across the slide, with a higher concentration in the top-left and bottom-right areas.

# try ~ catch ~ throw

Three keywords in C++ exception handling

# try ~ catch ~ throw

- 익셉션은 두 부분으로 구성

- 익셉션을 처리하기 위한 try/catch 블록

```
try {  
    // 익셉션이 발생할 코드  
} catch (exception-type1 exception-name) {  
    // type1 익셉션을 받아서 처리할 코드  
} catch (exception-type2 exception-name) {  
    // type2 익셉션을 받아서 처리할 코드  
}  
// 나머지 코드
```

- 익셉션을 발생시키기 위한 throw 구문

- 많은 경우 throw 구문은 라이브러리의 깊은 곳에 숨어 있어 프로그래머가 볼 수 없음
- 그렇게 숨어 있는 throw 구문도 try/catch 블록과 연동됨

# try ~ catch ~ throw

- 익셉션을 처리하기 위한 try/catch 블록

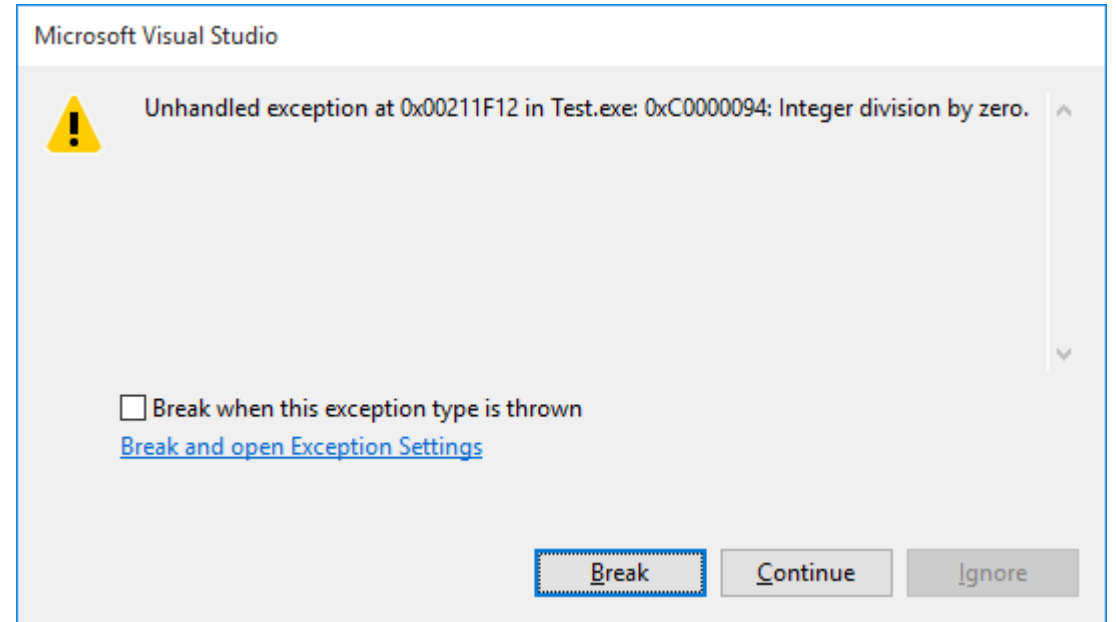
- 익셉션이 발생할 코드는 throw 구문을 직접 포함할 수도 있고, 코드에서 이용되는 함수 안에서 여러 차례의 호출 단계를 거쳐 간접적으로 익셉션이 발생할 수도 있음
- 익셉션이 발생하지 않으면 catch 블록의 코드가 실행되지 않고 try 블록 다음에 오는 나머지 코드 부분만 실행됨
- 만약 throw 구문이 실행되어 익셉션이 발생하면, throw 구문 다음에 오는 코드는 실행되지 않고 발생한 익셉션 타입에 합치하는 catch 블록으로 실행 흐름이 분기됨
- catch 블록에서 리턴이나 또 다른 익셉션을 발생시키는 등의 실행 흐름 분기를 하지 않으면 catch 블록이 실행된 다음 마지막 catch 블록 다음에 오는 나머지 코드가 실행됨

# try ~ catch ~ throw

- 예제 : 나눗셈의 분모가 0인 예외 상황 (Divided by zero)

```
int Divide(int num, int den)
{
    return num / den;
}

int main() {
    cout << Divide(5, 2) << endl;
    cout << Divide(10, 0) << endl;
    cout << Divide(3, 3) << endl;
}
```



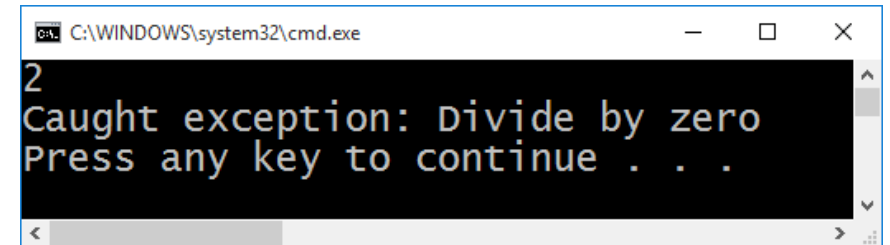


# try ~ catch ~ throw

- 예제 : 나눗셈의 분모가 0인 예외 상황 (Divided by zero)

```
int Divide(int num, int den)
{
    if (den == 0)
        throw invalid_argument("Divide by zero");
    return num / den;
}

int main() {
    try {
        cout << Divide(5, 2) << endl;
        cout << Divide(10, 0) << endl;
        cout << Divide(3, 3) << endl;
    } catch (const invalid_argument& e) {
        cout << "Caught exception: " << e.what() << endl;
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
2
Caught exception: Divide by zero
Press any key to continue . . .
```

# try ~ catch ~ throw

- 예제 : 파일을 열 수 없는 예외 상황 (Unable to open file)

```
void readFile(const string& fileName,
              vector<int>& dest)
{
    ifstream istr;
    int temp;
    istr.open(fileName.c_str());
    // 값을 하나씩 읽어서 vector에 저장
    while (istr >> temp)
        dest.push_back(temp);
}
```

```
int main()
{
    vector<int> myInts;
    const string fileName = "numbers.txt";
    readFile(fileName, myInts);
    for (size_t i = 0; i < myInts.size(); i++)
        cout << myInts[i] << " ";
    cout << endl;
}
```

# try ~ catch ~ throw

- 예제 : 파일을 열 수 없는 예외 상황 (Unable to open file)

```
void readFile(const string& fileName,
              vector<int>& dest)
{
    ifstream istr;
    int temp;
    istr.open(fileName.c_str());
    if (istr.fail())
        throw exception();
    // 값을 하나씩 읽어서 vector에 저장
    while (istr >> temp)
        dest.push_back(temp);
}
```

```
int main()
{
    vector<int> myInts;
    const string fileName = "numbers.txt";
    try {
        readFile(fileName, myInts);
    } catch (const exception& e) {
        cerr << "Unable to open file "
              << fileName << endl;
        return 1;
    }
    for (size_t i = 0; i < myInts.size(); i++)
        cout << myInts[i] << " ";
    cout << endl;
}
```

# try ~ catch ~ throw

- 익셉션으로 던질 데이터는 어떤 타입이든 사용할 수 있음

```
int Divide(int num, int den)
{
    if (den == 0)
        throw "Divide by zero";
    return num / den;
}
```

```
int main() {
    try {
        cout << Divide(5, 2) << endl;
        cout << Divide(10, 0) << endl;
        cout << Divide(3, 3) << endl;
    } catch (const string& s) {
        cout << s.c_str() << endl;
    }
}
```

익셉션으로 던질 데이터의 타입이 변경되면  
catch 구문도 수정해야 함

# try ~ catch ~ throw

- 여러 상황에 따라 서로 다른 익셉션 타입을 사용할 수도 있음

```
void readFile(const string& fileName, vector<int>& dest) {  
    ifstream istr;  
    int temp;  
    istr.open(fileName.c_str());  
    if (istr.fail())  
        // 파일 열기 실패에 따른 익셉션  
        throw invalid_argument("");  
    // 값을 하나씩 읽어서 vector에 저장  
    while (istr >> temp)  
        dest.push_back(temp);  
    if (istr.eof())  
        istr.close();  
    else {  
        // 알 수 없는 오류에 의한 익셉션  
        istr.close();  
        throw runtime_error("");  
    }  
}  
}
```

```
try  
{  
    readFile(fileName, myInts);  
}  
catch (const invalid_argument& e)  
{  
    cerr << "Unable to open file " << fileName << endl;  
    return 1;  
}  
catch (const runtime_error& e)  
{  
    cerr << "Error reading file " << fileName << endl;  
    return 1;  
}
```

# try ~ catch ~ throw

- 함수나 멤버 함수에서 던질 수 있는 익셉션 타입의 종류를 지정할 수 있음  
→ throw 리스트 또는 익셉션 명세(Exception specification)

```
void readFile(const string& fileName, vector<int>& dest)
    throw (invalid_argument, runtime_error) { ... }
```

- throw 리스트는 함수 정의부분만 아니라 함수 선언부에도 반드시 같이 지정되어야 함

```
void readFile(const string& fileName, vector<int>& dest)
    throw (invalid_argument, runtime_error);
```

- throw 리스트를 달리하는 것만으로는 함수 오버로딩을 할 수 없음
- throw 리스트가 명시되어 있지 않으면 어떤 익셉션이든 던질 수 있음
- 함수나 멤버 함수가 아무런 익셉션도 발생시키지 않는다면 공백 throw 리스트를 사용
  - C++11에서는 noexcept 키워드를 사용

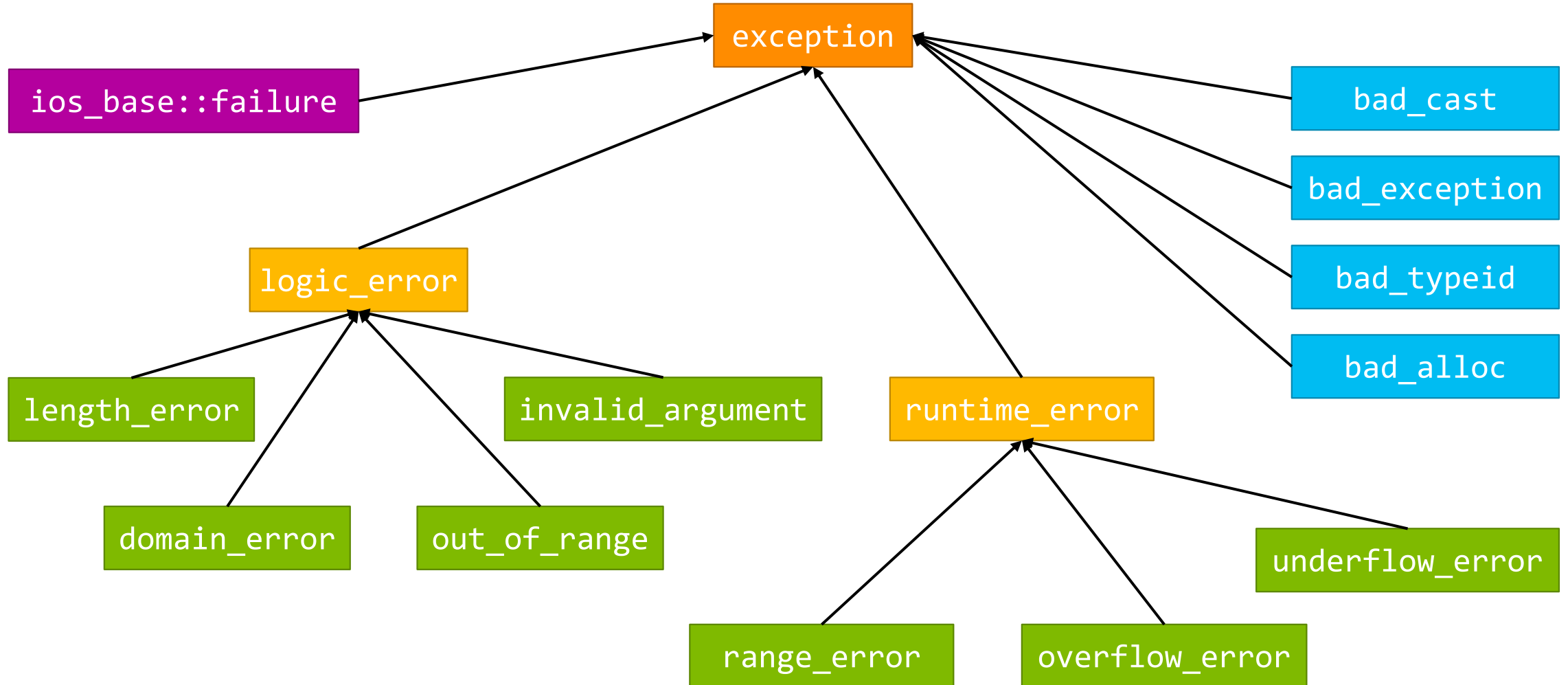
```
void readFile(const string& fileName, vector<int>& dest) throw();           // C++98
```

```
void readFile(const string& fileName, vector<int>& dest) noexcept;         // C++11
```

# Standard exception classes

Base class for standard exceptions in C++

# Standard exception classes





# Standard exception classes

- `bad_alloc` : 메모리 할당 오류로 new 연산에서 발생
- `bad_cast` : 형변환 오류로 `dynamic_cast`에서 발생
- `bad_typeid` : typeid에 대한 피연산자가 nullptr인 경우 발생
- `bad_exception` : 예기치 못한 예외로 함수 발생 목록에 없는 예외
- `logic_error` : 클래스와 관련된 논리 오류
  - `out_of_range`, `invalid_argument`, `length_error` 등
- `runtime_error` : 런타임에 발생하는 오류
  - `range_error`, `overflow_error`, `underflow_error` 등

# Stack unwinding

The general act of popping one or more frames off the stack to resume execution elsewhere in the program

# Stack unwinding

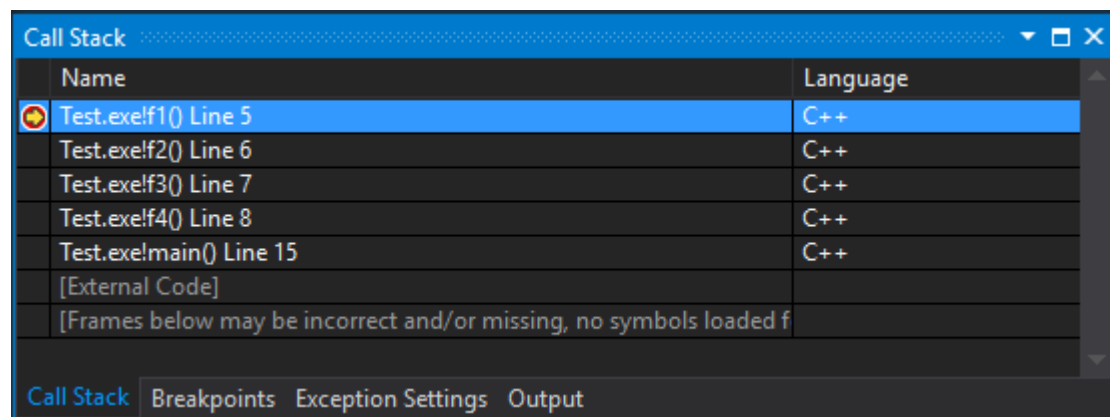
- 코드에서 익셉션이 발생하면 스택에서 캐치 핸들러를 찾음
  - 캐치 핸들러는 바로 현재의 스택 프레임에 존재할 수도 있고 몇 단계의 함수 호출 스택을 거슬러 올라가야 있을 수도 있음
  - 캐치 핸들러를 찾으면 해당 스택 위치로 스택 프레임을 모두 되돌리게 됨
  - 스택 풀기(Stack unwinding)란 스택 프레임마다 지역 변수를 모두 삭제하고 남아있는 코드의 실행을 생략한 채 그 함수가 호출된 지점으로 실행 코드를 되돌리는 것을 말함

# Stack unwinding

```
void f1() { throw 0; }  
void f2() { f1(); }  
void f3() { f2(); }  
void f4() { f3(); }
```

```
int main()  
{  
    try  
    {  
        f4();  
    }  
    catch (int e)  
    {  
        cout << e << endl;  
    }  
}
```

함수 호출



Name	Language
Test.exelf1() Line 5	C++
Test.exelf2() Line 6	C++
Test.exelf3() Line 7	C++
Test.exelf4() Line 8	C++
Test.exelmain() Line 15	C++
[External Code]	
[Frames below may be incorrect and/or missing, no symbols loaded f...	

Call Stack Breakpoints Exception Settings Output

스택 풀기

# Stack unwinding

- 하지만 스택 풀기가 일어날 때 포인터 변수에 할당된 메모리는 해제 작업이 이루어지지 않기 때문에 메모리 릭 문제가 발생할 수 있음

```
void funcOne() throw(exception) {  
    string str1;  
    string* str2 = new string();  
    funcTwo();  
    delete str2;  
}
```

실행 X → 포인터 변수는 메모리 릭 발생

```
void funcTwo() throw(exception) {  
    ifstream istr;  
    istr.open("fileName");  
    throw exception();  
    istr.close();  
}
```

실행 X → 지역 변수는 소멸자 실행

```
int main()  
{  
    try  
    {  
        funcOne();  
    }  
    catch (const exception& e)  
    {  
        cerr << "Exception caught!" << endl;  
        return 1;  
    }  
}
```

예외 발생

# Stack unwinding

- 해결 방법 1 : 스마트 포인터를 이용 (shared\_ptr, unique\_ptr 등)
  - 스마트 포인터는 스택에 저장, 소멸할 때마다 연관된 리소스를 해제
    - shared\_ptr : [http://en.cppreference.com/w/cpp/memory/shared\\_ptr](http://en.cppreference.com/w/cpp/memory/shared_ptr)
    - unique\_ptr : [http://en.cppreference.com/w/cpp/memory/unique\\_ptr](http://en.cppreference.com/w/cpp/memory/unique_ptr)
- #include <memory>

```
void funcOne() throw(exception)
{
    string str1;
    // string* str2 = new string();
    unique_ptr<string> str2(new string("Hello"));
    funcTwo();
    // delete str2;
}
```

# Stack unwinding

- 해결 방법 2 : 익셉션 받기, 리소스 정리, 재전송
- 함수 호출마다 발생 가능한 모든 익셉션을 받아 필요한 리소스 해제 작업을 완료한 후, 상위 익셉션 핸들러에 재전송

```
void funcOne() throw(exception)
```

```
{
```

```
    string str1;
```

```
    string* str2 = new string();
```

```
    try {
```

```
        funcTwo();
```

```
    } catch (...) {
```

```
        delete str2;
```

```
        throw; // 익셉션 재전송
```

```
    }
```

```
    delete str2;
```

```
}
```

“...”은 모든 타입의 익셉션을 매칭할 수 있음