# Chapter 05 - 예외 처리하기

| 생성일 | @2025년 9월 21일 오후 8:22 |
| --- | --- |
| 태그 | |

- C++에서는 예외가 발생했음을 명시적으로 구분하고, 예외를 더 간단하게 처리할 수 있는 문법을 제공함

→ try, catch, throw문

try, catch, throw 문으로 예외 처리

```
try {
    // 예외를 던질 수 있는 코드 -> 예외 발생 가능한 코드 블록
} catch ( 예외_형식 예외_이름 ) {
    // 예외를 처리하는 코드
}
```

- C++에서 예외 처리는 try 키워드로 시작함
  - 예외가 발생할 수 있는 코드를 try와 함께 중괄호를 묶음
  - 그리고 조건문 등으로 예외인지 판단한 후, throw 명령으로 예외를 던짐
  - 예외를 던질 때는 예외를 설명하는 값이나 객체를 함께 전달함

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

int main()
{
    try
    {
        int input;
        cout << "정수 중 하나를 입력하세요 : ";
        cin >> input;
```

```
        if (input > 0)  // 입력받은 숫자가 양수이면
        {
            cout << "throw 1" << endl;
            throw 1;    // 예외 1 발생 ( 정수 형식 예외)
            cout << "after throw 1" << endl;
        }

        if (input < 0) // 임력받은 숫자가 음수이면
        {
            cout << "throw -1.0f" << endl;
            throw - 1.0f;   // 예외 1.0f 발생 (부동소수점 형식
예외)
            cout << "after throw -1.0f" << endl;
        }

        if (input == 0) // 입력받은 숫자가 0이면
        {
            cout << "throw Z" << endl;
            throw 'Z';  // 예외 Z 발생 (문자 형식 예외)
            cout << "after throw Z" << endl;
        }
    }

    catch (int a)   // 정수 형식 예외 받기
    {
        cout << "catch " << a << endl;
    }

    catch (float b)  // 부동 소수점 형식 예외 받기
    {
        cout << "catch " << b << endl;
    }

    catch (char c)  // 문자 형식 예외 받기
    {
        cout << "catch " << c << endl;
    }
```

```
    return 0;
}
```

- 첫 번째 실행 결과를 보면 throw 1 이전 메세지만 나오고 이후 메세지는 나오지 않았음
  → throw 이후의 코드는 무시되고 try 구문을 빠져나옴
  → try 블록에서 차례대로 실행되다가 예외가 발생하면 이후 구문은 실행되지 않는다.

## catch (...) 문으로 기타 예외 처리

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

int main()
{
    try
    {
        throw 1; // 예외 1 발생 (정수 형식의 예외)
    }
    catch (char c) // 문자 형식 예외 받기
    {
        cout << "catch " << c << endl;
    }

    return 0;
}
```

- **catch(...)문**으로 일부 예외는 뭉뚱그려 처리할 수 있음.

## 처리되지 않은 모든 예외 받기

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

int main()
```

```
{
    try
    {
        int input;
        cout << "정수 중 하나를 입력해보세요 : ";
        cin >> input;

        if (input > 0)  // 입력받은 숫자가 양수이면
        {
            cout << "throw 1" << endl;
            throw 1;    // 예외 1 발생 (정수 형식 예외)
            cout << "after throw 1" << endl;
        }

        if (input < 0)  // 임력받은 숫자가 인수이면
        {
            cout << "throw -1.0f" << endl;
            throw - 1.0f;   // 예외 1.0f 발생 (부동 소수점 형식
예외)
            cout << "after throw -1.0f" << endl;
        }

        if (input == 0) // 임력받은 숫자가 음수이면
        {
            cout << "throw Z" << endl;
            throw 'Z';  // 예외 Z 발생 (문자 형식 예외)
            cout << "after throw -1.0f" << endl;
        }
    }

    catch (int a)   // 정수 형식 예외 받기
    {
        cout << "catch " << a << endl;
    }
    catch (...) // 처리되지 않은 나머지 예외 모두 받기
    {
        cout << "catch all" << endl;
    }
```

```
    return 0;
}
```

- 예외를 한번에 catch(...) 문으로 처리하는 것은 편리하지만 좋은 코드는 아님
    - 예외는 종류마다 제대로 구분해서 처리해 줘야 더 완벽한 프로그램을 만들 수 있음

## 예외가 전달되는 동작 확인하기

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

void func_throw()
{
    cout << "func_throw()" << endl;
    cout << "throw -1" << endl;
    throw - 1;  // 정수 형식 예외 던지기
    cout << "after throw -1" << endl;
}

int main()
{
    try
    {
        func_throw();
    }
    catch (int exec)    // 정수 형식 예외 받기
    {
        cout << "catch " << exec << endl;
    }
    return 0;
}
```

- func_throw 함수에서 던진 예외가 이 함수를 호출한 main 영역의 catch문에서 정상으로 처리된 것을 확인할 수 있음
    - 예외 처리의 책임은 throw가 발생한 함수를 호출한 쪽으로 넘어감

→ **스택 풀기**

## 스택 풀기 동작 확인하기

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

void func_throw()
{
    cout << endl;
    cout << "func_throw() 함수 내부" << endl;
    cout << "throw -1" << endl;
    throw - 1;
    cout << "after throw -1" << endl;
}

void func_2() {
    cout << endl;
    cout << "func_2() 함수 내부" << endl;
    cout << "func_throw() 호출" << endl;
    func_throw();
    cout << "after func_throw()" << endl;
}

void func_1() {
    cout << endl;
    cout << "func_1() 함수 내부" << endl;
    cout << "func_2() 호출" << endl;
    func_2();
    cout << "after func_2()" << endl;
}

int main()
{
    cout << "main 내부" << endl;

    try {
```

```
        cout << "func_1 호출" << endl;
        func_1();
    }
    catch (int exec) {  // 정수 형식 예외 받기
        cout << endl;
        cout << "catch " << exec << endl;
    }

    return 0;
}
```

실행 결과

```
main 내부
func_1 호출

func_1() 함수 내부
func_2() 호출

func_2() 함수 내부
func_throw() 호출

func_throw() 함수 내부
throw -1

catch -1
```

- 함수가 호출되는 순서는 다음과 같음

**main → func_1( ) → func_2( ) → func_throw( )**

- 예외를 전달하는 순서가 스택에 쌓인 역순이므로 ***스택 풀기***라고 함.

## 어설션을 이용한 예외 처리

- 어설션 : 코드를 검증하여 예상치 못한 상황에서 프로그램 동작을 중단 시키는 도구

  → 안전성과 신뢰성을 높여줌
- C++에서는 <cassert> 헤더에 정의된 assert 매크로를 통해 예외를 비교적 간단하게 처리할 수 있음

```
#include <iostream>
#include <cassert>

using namespace std;

void print_number(int* _pt_int)
{
    assert(_pt_int != NULL);
    cout << *_pt_int << endl;
}

int main()
{
    int a = 100;
    int* b = NULL;
    int* c = NULL;

    b = &a;
    print_number(b);

    // c는 NULL인 상태로 인자 전달
    print_number(c);

    return 0;
}
```

- assert는 주로 개발 과정에서 조건을 검사하여 프로그램이 예상대로 동작하는지 확인하는 데 사용함

## 디버그 모드와 릴리즈 모드

- 디버그 모드 : 컴파일할 때 디버깅을 위한 정보들을 삽입해서 문제가 발생할 때 원인을 수월하게 찾을 수 있도록 하는 컴파일 모드
    - 디버깅 : 버그를 제거하는 것
- 릴리즈 모드 : 최종 사용자에게 배포할 코드를 만들 때 사용함
    - 릴리즈 모드에서는 최적화가 적용되어 프로그램의 실행 속도가 향상됨

→ **개발 : 디버그 모드, 배포 : 릴리즈 모드**

## assert를 사용할 때 주의할 점

- assert는 디버그 모드에서만 컴파일 되므로 ***다른 코드에 영향을 주지 않는 코드로만 작성***해야 함

05-2 ) 예외 처리 생략과 실패 대응

예외 처리 생략 - noexcept

- 함수가 예외를 던지지 않음을 나타낼 때는 다음처럼 noexcept 키워드로 명시

```
int func() noexcept
```

또는 함수를 호출할 때 noexcept 키워드를 사용 → 컴파일할 때 해당 함수가 예외를 던지는지 확인해 true나 false로 알려줌

```
bool does_not_throw = noexcept(my_function());
```

noexcept로 명시된 함수에서 예외 던지기

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

void real_noexcept() noexcept
```

```
{
    cout << "real_noexcept" << endl;
}

// noexcept로 명시된 함수 내에서 예외 발생
void fake_noexcept() noexcept
{
    cout << "fake_noexcept" << endl;
    throw 1;    // 정수 형식 예외 발생
}

int main()
{
    real_noexcept();

    try
    {
        fake_noexcept();
    }
    catch (int exec)
    {
        cout << "catch " << exec << endl;
    }

    return 0;
}
```

- 경고는 발생하지만 실행 파일은 만들어짐

## 예외 처리 실패 대응 - set_terminate

```
set_terminate(종료_처리_함수);
```

## 예외 처리 실패에 대응하기

```
#include <iostream>
#include <cstdlib>

using namespace std;

// 종료 처리 함수
void myterminate()
{
    cout << "myterminate called" << endl;
    exit(-1);   // 프로그램을 비정상으로 종료
}

int main(void)
{
    set_terminate(myterminate); -> 종료 처리 함수 지정
    throw 1;    // 예외 발생

    return 0;   // throw로 예외를 던졌음으로 실행되지 않음
}
```

- main 함수에서 throw 1 코드를 만나면 catch문을 찾을 수 없어 오류가 발생하고 프로그램이 강제로 종료됨