



# 고급프로그래밍

## 고급문법 2

Professor Jeong, Mun-Ho

Robot Vision & Intelligence Laboratory  
Kwangwoon University  
(02-940-5625, mhjeong@kw.ac.kr)

# Schedule

주차	주제		과제	퀴즈
1	과목소개	교과목 소개 (1), C++ 시작 (2)		
2	C++	C++ 프로그래밍의 기본 (3), 클래스와 객체 (4)	1	1
3		객체생성과 사용 (5)	2	2
4		함수와 참조 (6, 3/26), 복사 생성자와 함수중복(7)	3	3
5		static, friend, 연산자중복 (8, 4/2), 연산자중복 상속(9)	4	4
6		상속 (10, 4/9), 가상함수와 추상클래스 (11)		5
7		템플릿과 STL (12, 4/16), 입출력(13)	5	
8		중간고사		
9		파일 입출력(14), 예외처리 및 C 사용(15)		6
10		람다식(16, 5/7) , 멀티스레딩(17, 5/9)	6	7
11		멀티스레딩(18, 5/14), 고급문법(19, 5/16)		8
12		고급문법 2(20, 5/21), 고급문법 3		
13	병렬프로그래밍	병렬프로그래밍		
14		병렬프로그래밍		
15		기말고사		

# 오늘의 학습내용

- 고급문법
  - 중괄호 초기화
  - 파일 시스템(C++17이후)
  - `Assert` + 형변환

중괄호 초기화

# 중괄호 초기화

```
class class_a {
public:
    double _double;
    string _string;
public:
    class_a() {}
    class_a(string str) : _string{ str } { } // _string(str)
    class_a(string str, double dbl) : _string{ str }, _double{ dbl } { }
                                     // _string(str), _double(dbl)
};

int main()
{
    class_a c1{ };
    class_a c1_1;
    class_a c2{ "ww" };
    class_a c2_1("xx");

    class_a c3{ "yy", 4.4 };
    class_a c3_1("zz", 5.5);
}
```

# 중괄호 초기화

```
class class_d {
public:
    float m_float;
    string m_string;
    wchar_t m_char;
};

int main()
{
    class_d d1{};
    class_d d1{ 4.5 };
    class_d d2{ 4.5, "string" };
    class_d d3{ 4.5, "string", 'c' };

    class_d d4{ "string", 'c' }; // compiler error
    class_d d5{ "string", 'c', 2.0 }; // compiler error
}
```

```
class_d* cf = new class_d{4.5};
```

# 중괄호 초기화

```
vector<int> v1{9, 10, 11 };  
map<int, string> m1{ {1, "a"}, {2, "b"} };  
string s{ 'a', 'b', 'c' };
```

# 중괄호 초기화 – thread

```
mutex mtx;


int sum = 0;
int counter = 0;

int main()
{
    cout.sync_with_stdio(true);

    auto work_func = []()
    {
        for (int i = 0; i < 10; i++)
        {
            mtx.lock();
            sum++;
            cout << this_thread::get_id() << " " << sum << endl;
            mtx.unlock();
        }
    };

    thread t1 = thread{work_func};
    thread t2 = thread{work_func};
    thread t3 = thread{work_func};

    t1.join();
    t2.join();
}
```





# 파일 시스템

# 파일 시스템

- `#include <filesystem>`
- 파일 메타데이터에 대한 접근 라이브러리
  - 파일 유무/탐색
  - 파일 삭제, 폴더 생성
  - 파일 생성 시간, 권한 등 확인

# 파일 존재성 보기

```
#include <filesystem>
#include <iostream>

using namespace std;
namespace fs = filesystem;

int main()
{
    filesystem::path p("./insa.txt");

    cout << p << " exist? " << boolalpha << fs::exists(p) << endl;
    cout << p << " file? " << fs::is_regular_file(p) << endl;
    cout << p << " directory? " << fs::is_directory(p) << endl;
}
```

# 경로 보기

```
#include <filesystem>
#include <iostream>

using namespace std;
namespace fs = filesystem;

int main() {
    fs::path p("./insa.txt");

    cout << "Current path   : " << fs::current_path() << endl;
    cout << "Relative path  : " << p.relative_path() << endl;
    cout << "Absolute path  : " << fs::absolute(p) << endl;
    cout << "Canonical path : " << fs::canonical(p) << endl;
}
```

# 실습 : 폴더 생성

- create\_directory( ) 함수 사용

- ① 현 경로에 "data"란 폴더를 생성하시오
- ② "data" 아래에 "data\_added" 폴더를 생성하시오

```
#include <filesystem>
#include <iostream>

using namespace std;
namespace fs = std::filesystem;

int main()
{
    fs::create_directory("./data");
    fs::create_directory("./data/data_added");
}
```

## 폴더 생성 2

- ③ 존재하지 않은 폴더명(data\_none) 아래에 "new"란 폴더를 생성하시오

```
#include <filesystem>
#include <iostream>

using namespace std;
namespace fs = std::filesystem;

int main() {
    fs::path p("./data/data_none/new");
    cout << p << " exist? " << boolalpha << fs::exists(p) << std::endl;

    fs::create_directories(p);

    cout << p << " exist? " << fs::exists(p) << endl;
}
```

# 파일 삭제

- ④ "insa.txt"를 ./data에 옮기시오
- ⑤ "insa.txt"를 remove( ) 함수를 사용하여 삭제하시오

```
#include <filesystem>
#include <iostream>

using namespace std;
namespace fs = std::filesystem;

int main()
{
    fs::path p("./data/insa.txt");
    cout << p << " exist? " << boolalpha << fs::exists(p) << std::endl;

    fs::remove(p);
    cout << p << " exist? " << boolalpha << fs::exists(p) << std::endl;
}
```

# 폴더 삭제

- ⑥ "./data/data\_added/"에 "insa2.txt", "insa3.txt"를 두시오.
- ⑦ remove( ) : 빈 폴더만 삭제 가능
- ⑧ remove\_all( )

```
#include <filesystem>
#include <iostream>

using namespace std;
namespace fs = std::filesystem;

int main()
{
    fs::path p("./data/data_added");

    error_code err;
    fs::remove(p, err); // 빈 폴더만 삭제 가능
    cout << err.message() << endl;

    fs::remove_all(p); // 폴더 삭제
}
```



# 폴더의 파일 순회

- ① `directory_iterator` 사용하여 파일 순회 결과를 출력하시오.

```
#include <filesystem>
#include <iostream>

using namespace std;
namespace fs = filesystem;

int main() {
    fs::directory_iterator itr(fs::current_path());

    while (itr != fs::end(itr))
    {
        const fs::directory_entry& entry = *itr;
        cout << entry.path() << endl;
        itr++;
    }
}
```

# 폴더의 파일 순회

② `directory_entry` 사용하여 파일 순회 결과를 출력하시오.

```
#include <filesystem>
#include <iostream>

using namespace std;
namespace fs = filesystem;

int main()
{
    for(const fs::directory_entry& entry : fs::directory_iterator(fs::current_path()))
    {
        cout << entry.path() << endl;
    }
}
```

# 폴더의 파일 순회

- ③ `"/data/data_added/"`에 `"insa2.txt"`, `"insa3.txt"`를 두시오.
- ④ 현 경로의 파일과 하위 폴더의 파일을 출력하시오

```
#include <filesystem>
#include <iostream>

using namespace std;
namespace fs = std::filesystem;

int main()
{
    for(const fs::directory_entry& entry :
        fs::recursive_directory_iterator(fs::current_path()))
    {
        cout << entry.path() << endl;
    }
}
```

# 폴더의 파일 순회

- ⑤ 현 경로의 "data" 폴더에 있는 파일과 그 하위 폴더의 파일을 출력하시오

```
#include <filesystem>
#include <iostream>

using namespace std;
namespace fs = std::filesystem;

int main()
{
    for(const fs::directory_entry& entry :
        fs::recursive_directory_iterator(fs::current_path()/ "data"))
    {
        cout << entry.path() << endl;
    }
}
```

assert + 형변환

# assert

- <cassert>
- 디버깅 모드에서 에러 검출용 함수
  - void assert(int expression)
  - Expression이 0이 되면 assert error 발생

```
#include<iostream>
#include<cassert>
using namespace std;
int main(void)
{
    int score;
    while (true)
    {
        cout << "Input score : ";
        cin >> score;

        //0보다 작은 score가 들어오면 assert error!!
        assert(score >= 0);
        cout << "=> score : " << score << endl;
    }
    return 0;
}
```

# 예제

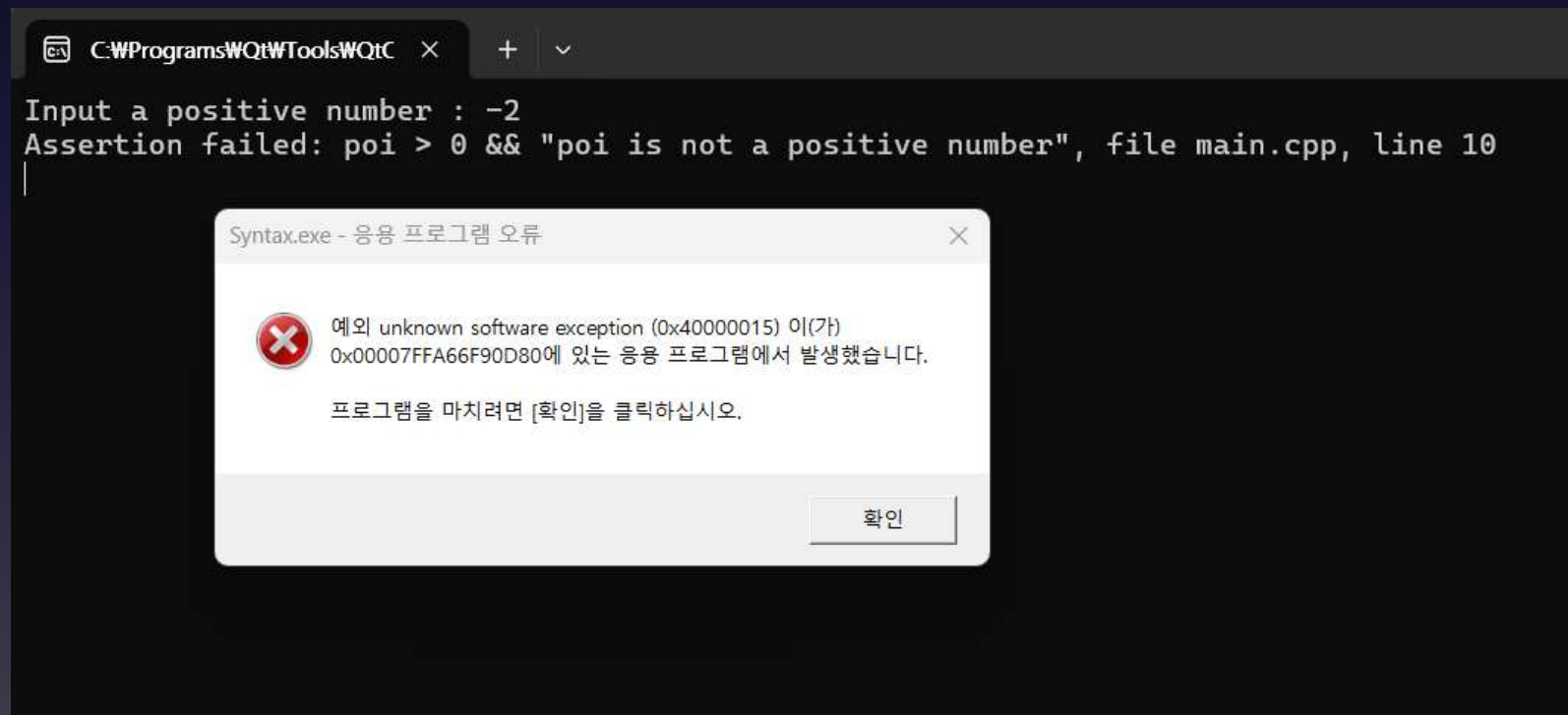
```
#include<iostream>
#include<cassert>
using namespace std;

void positive(int poi)
{
    assert(poi > 0);
    cout << poi << " 는 양의 정수" << endl;
}

int main()
{
    int num;
    cout << "Input a positive number : ";
    cin >> num;
    positive(num);
    return 0;
}
```

실습

- 앞의 예제에서 poi가 음의 정수일 경우 아래와 같이 수행되도록 수정 하시오.





# 형변환 연산자

## ■ Type Conversion Operator

- operator 변환 타입 ( )

```
class Int {  
    int data;  
public:  
    Int(int data) : data(data) { }  
    Int(const Int& i) : data(i.data) { }  
  
    operator int() { return data; }  
};  
  
int main() {  
    Int x = 3;  
  
    int a = x + 4;    //암묵적으로 int( ) 호출  
    x = a * 2 + x + 4; //암묵적으로 int( ) 호출  
  
    cout << x << endl;  
}
```

# 형변환 연산자

## ■ Type Conversion Operator

- operator 변환 타입 ( )

```
class Temp{
public:
    operator bool() { return true; }
    operator int() { cout << "operator int() Call" << endl; return 10; }
    operator float() { cout << "operator float() Call" << endl; return 20.f; }
};

int main()
{
    int i; float f; Temp t;
    i = t;    //암묵적으로 int( ) 호출
    f = t;    //암묵적으로 float( ) 호출
    if ( t )
        cout << "implicit call of bool( )" << endl
}
```

# Explicit 형변환 연산자

- Type Conversion Operator

- operator 변환 타입 ( )

```
class Temp{
public:
    explicit operator bool() { return true; }
    explicit operator int() { cout << "operator int() Call" << endl; return 10; }
};

int main()
{
    int i; Temp t;

    i = t;    //오류
    i = static_cast<int>(t);    //명시적 int( ) 호출

    if( t ) //암묵적 그러나 허용
        cout << "implicit call of bool( )" << 두이

    if( t == true ) //오류
        cout << "implicit call of bool( )" << endl

}
```

