# 6-3. C++ 프로그래밍 시작

창의적소프트웨어프로그래밍
2022년도 여름학기
Racin

# 이번 시간에는

- 오늘 마지막 세트에서는 C++에 발을 살짝 담가 보려고 해요.
  - 처음 보는 단어들, **연산자**들이 약간 나오기는 하는데
    뭐 지금은 일단 복붙메타 충실히 따르면 될 듯 해요

- 우선 CSP_6_3_yeshi_base.cpp를 VS에 탑재해 봅시다.

# C++ 버전 Hello, World! 프로그램

- 옹... 당연히 나와야 할 코드가 적혀 있습니다.
  - 실행해 보면 당연히 나와야 할 메시지가 출력될 거예요!

```cpp
#include <iostream>
#include <string>

int main()
{
    std::string str{ "Hello, World!" };
    std::cout << str << std::endl;

    return 0;
}
```

# C++ 버전 Hello, World! 프로그램

- 살짝 들여다 보면...

```cpp
#include <iostream>
#include <string>

int main()
{
    std::string str{ "Hello, World!" };
    std::cout << str << std::endl;

    return 0;
}
```

표준 C++ 라이브러리 중
기본 입력 / 출력 기능을 쓰고 싶을 때
include하는 헤더 파일

# C++ 버전 Hello, World! 프로그램

- 살짝 들여다 보면...

```
#inclu
#inclu

int ma
{
    std::string str = "Hello, World!";
    std::cout << str << std::endl; // :은 콜론, endl 끝은 소문자 L

    return 0;
}
```

이전에 살짝 본 적 있는 :: **연산자**는
Python에서 random.randint() 썼을 때의 .이랑
대충 유사한 의미를 가져요.
('그럼 C의 . **연산자**는?' 싶으면 정상이에요)

표준 C++ 라이브러리에서 정해 둔 **이름**들은
대부분 namespace std에 소속되어 있어요!
(namespace는 지금은 그러려니 해도 좋을 듯?)
('소속'도 그냥 그러려니 해도 좋을 듯?)

# C++ 버전 Hello, World! 프로그램

- 살짝 들여다 보면...

```
#inclu
#inclu

int ma
{
    std::string str = "Hello, World!";
    std::cout << str << std::endl; // :은 콜론, endl 끝은 소문자 L

    return 0;
}
```

이전에 살짝 본 적 있는 :: **연산자**는
Python에서 random.randint() 썼을 때의 .이랑
대충 유사한 의미를 가져요.
('그럼 C의 . **연산자**는?' 싶으면 정상이에요)

여기는 **선언**의 specifier 자리라서 **수식**을 적기는 애매한 곳이에요.
일단은, :: '**연산자**'는 조금 특별한 친구라고 두고 넘어갑시다.
실제로도 얘는 (그 구조상 이항 **연산자**임에도) 붙는 우선순위 0순위예요!

# C++ 버전 Hello, World! 프로그램

- 살짝 들여다 보면...

```
#incl
#incl

int m
{
    std::str
    std::cout << str << std::endl; // :은 콜론, endl 끝은 소문자 L

    return 0;
}
```

cout은 "see-out"이라 발음하며,
여기서의 c는 character를 의미해요.
화면에 글자를 출력하기 때문에 이렇게 이름지었대요.

# C++ 버전 Hello, World! 프로그램

- 살짝 들여다 보면...

```
#include <iostream>
#include <string>

int main()
{
    std::string str = "Hello, W                ";
    std::cout << str << std::endl; // :은 콜론, endl 끝은 소문자 L

    return 0;
}
```

endl은 end-line을 의미해요.
이걸 std::cout, << **연산자**랑 섞어 **수식**을 구성하면
적절히 엔터 키를 출력해 줘요.

# C++ 버전 Hello, World! 프로그램

- 살짝 들여다 보면...

```
#inc
#inc

int
{
    std::string str = "Hello, World!";
    std::cout << str << std::endl; // :은 콜론, endl 끝은 소문자 L

    return 0;
}
```

std::string **형식**은 Python의 **str 형식**과 비슷하게 쓸 수 있어요.
당분간은 큰 고민 없이 ㄱㄱ 가능할 듯!

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 이제 보통 이거 바로 다음에 하는,
  int double 이런거 쓰는 시간을 가져 봅시다.

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 이제 보통 이거 바로 다음에 하는,
  int double 이런거 쓰는 시간을 가져 봅시다.
  - main() **정의** 내용물을 아래와 같이 바꾸고 실행해 봅시다:

```
int main()
{
    std::string str = "Hello, World!";

    std::cout << 3 << std::endl;
    std::cout << 4.5 << std::endl;
    std::cout << 'a' << std::endl;
    std::cout << "Hello" << std::endl;
    std::cout << str << std::endl;
    return 0;
}
```

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 오...맙소사...
  이렇게 쉬울 수가 없습니다.

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 오...맙소사...
  이렇게 쉬울 수가 없습니다.
  - C였다면 우리가 %d %f %c로 직접 '출력할 **값**의 **형식**'에 맞게 양식을 구성해야 했는데,
    C++에선 << **연산자** 하나만 외우면 다 되는 것 같습니다

- 이게 왜 되는 건지 궁금한 친구들은...

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 오...맙소사...
  이렇게 쉬울 수가 없습니다.
  - C였다면 우리가 %d %f %c로 직접 '출력할 **값**의 **형식**'에 맞게 양식을 구성해야 했는데,
    C++에선 << **연산자** 하나만 외우면 다 되는 것 같습니다

- 이게 왜 되는 건지 궁금한 친구들은...
  지금 VS에 적어 둔 << **연산자**에 마우스 포인터를 갖다 대 보세요.

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 오...맙소사...
  이렇게 쉬울 수가 없습니다.
  - C였다면 우리가 %d %f %c로 직접 '출력할 **값**의 **형식**'에 맞게 양식을 구성해야 했는데,
    C++에선 << **연산자** 하나만 외우면 다 되는 것 같습니다

- 이게 왜 되는 건지 궁금한 친구들은...
  지금 VS에 적어 둔 << **연산자**에 마우스 포인터를 갖다 대 보세요.
  - 뭔가 엄청 긴 게 튀어 나오는데,
    **수식** 3 왼쪽 << **연산자**를 보면 툴팁 끝에 ::operator<<(int _Val) 이라고 나오는 것을 대충 목격할 수 있을 것입니다
    - 다른 수식 왼쪽 << **연산자**는 뭐라고 나오는 지 확인해 보세요!

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 오...맙소사...
  이렇게 쉬울 수가 없습니다.
  - C였다면 우리가 %d %f %c로 직접 '출력할 **값**의 **형식**'에 맞게 양식을 구성해야 했는데,
    C++에선 << **연산자** 하나만 외우면 다 되는 것 같습니다

- 이게 왜 되는 건지 궁금한 친구들은...
  지금 VS에 적어 둔 << **연산자**에 마우스 포인터를 갖다 대 보세요.
  - 뭔가 엄청 긴 게 튀어 나오는데,
    **수식** 3 왼쪽 << **연산자**를 보면 툴팁 끝에 ::operator<<(int _Val) 이라고 나오는 것을 대충 목격할 수 있을 것입니다
    - 다른 수식 왼쪽 << **연산자**는 뭐라고 나오는 지 확인해 보세요!

  - 오... 잘은 모르겠지만, 이 많은 형식들을 커버하는 **함수**들을 미리 정의해 놨나 봅니다

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 이번에는 사용자로부터 키보드 입력을 받아 봅시다.
  main() **정의** 내용물을 아래와 같이 바꾸고 실행해 보세요:

```
int main()
{
    int number;

    std::cout << "숫자를 입력하세요>";
    std::cin >> number;

    std::cout << "입력한 숫자는 ";
    std::cout << number;
    std::cout << "입니다." << std::endl;

    return 0;
}
```

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 한국인을 위한 메시지 출력 부분을 뺀다면,
  이 코드는 대충 이런 구조를 가집니다: (여러분 코드는 그냥 둬요!)

```
int main()
{
    int number;


    std::cin >> number;


    std::cout << number;


    return 0;
}
```

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 한국인을 위한 메시지 출력 부분을 뺀다면,
  이 코드는 대충 이런 구조를 가집니다: (여러분 코드는 그냥 둬요!)

```
int main()
{
    int number;

    std::cin >> number;

    std::cout << number;

    return 0;
}
```

입력과 출력은 대충 대칭이니까
scanf()와 printf() 호출식이 대충 비슷하게 생겼듯,
C++에서도 이런 식으로 대칭되어 보이게 만들어 놓았나 봐요!

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 한국인을 위한 메시지 출력 부분을 뺀다면,
  이 코드는 대충 이런 구조를 가집니다: (여러분 코드는 그냥 둬요!)

```
int main()
{
    int number;

    std::cin >> number;

    std::cout << number;

    return 0;
}
```

입력과 출력은 대충 대칭이니까
scanf()와 printf() 호출식이 대충 비슷하게 생겼듯,
C++에서도 이런 식으로 대칭되어 보이게 만들어 놓았나 봐요!

어? 그러고 보니 scanf() 쓸 때 붙이던 &는 어디로 갔나요?

# 기본 형식 값을 기본 방법으로 입력받기 / 출력하기

- 한국인을 위한 메시지 출력 부분을 뺀다면,
  이 코드는 대충 이런 구조를 가집니다: (여러분 코드는 그냥 둬요!)

```
int main()
{
    int number;

    std::cin >> number;

    std::cout << number;

    return 0;
}
```

입력과 출력은 대충 대칭이니까
scanf()와 printf() 호출식이 대충 비슷하게 생겼듯,
C++에서도 이런 식으로 대칭되어 보이게 만들어 놓았나 봐요!

어? 그러고 보니 scanf() 쓸 때 붙이던 &는 어디로 갔나요?
...이 물음에 대한 답은 >> **연산자**에 마우스 대면 나오고,
다음 시간에 좀 더 본격적으로 다루어 보도록 할께요.

# using 지시자

- 아마 이 쯤 오면 슬슬 매 번 std:: 적는 것이 귀찮아질텐데,
그럴 땐 아래와 같이 'using 지시자'를 적음으로써 편의를 도모할 수 있어요:

```
using namespace std; // 이걸 미리 적어 두면...

int main()
{
    int number;

    cin >> number; // std::를 생략해도 되고

    std::cout << number; // 평소처럼 '풀 네임'을 직접 적어도 돼요!

    return 0;
}
```

# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 이번에는, 검은 창에 출력하는 게 아니라
  뭔가 다른 곳에 출력하는 방법을 살펴 봅시다.

# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 이번에는, 검은 창에 출력하는 게 아니라
  뭔가 다른 곳에 출력하는 방법을 살펴 봅시다.
  - 방금 전에 마우스 갖다 대면서 느낀 점,
    우항 **수식**의 **형식**에 따라 실제로 **실행**할 Code가 자동으로 선택된다는 점을 생각하면,
    이번에는 << **연산자** 왼쪽을 std::cout이 아닌 다른 것으로 바꾼다고 보면 되겠지요?

# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 이번에는, 검은 창에 출력하는 게 아니라
  뭔가 다른 곳에 출력하는 방법을 살펴 봅시다.
  - 방금 전에 마우스 갖다 대면서 느낀 점,
    우항 **수식**의 **형식**에 따라 실제로 **실행**할 Code가 자동으로 선택된다는 점을 생각하면,
    이번에는 << **연산자** 왼쪽을 `std::cout`이 아닌 다른 것으로 바꾼다고 보면 되겠지요?

  - 오늘은 두 가지 케이스를 살펴 봅니다:
    - 파일에
    - `std::string` **object**에

# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 먼저, 파일입니다. input.txt를 예로 들어 볼께요.
  방금 전 코드를 아래와 같이 수정해 봅시다:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main()
{
    int number;

    cout << "숫자를 입력하세요>";
    cin >> number;

    // 오른쪽 상자에서 계속...
```

```
    // ...왼쪽 상자 다음에 이어짐

    ofstream os{"input.txt"}; // 열기
    os << number;
    os.close(); // 저장 후 닫기

    cout << "입력한 숫자는 ";
    cout << number;
    cout << "입니다." << endl;

    return 0;
}
```

# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 먼저, 파일입니다. input.txt를 예로 들어 볼께요.
  방금 전 코드를 아래와 같이 수정해 봅시다:
  - 뭔가 생긴 것만 봐서는,
    실행하면 input.txt에 내가 적은 숫자(에 대한 10진수 표현)가 들어 있을 것 같습니다

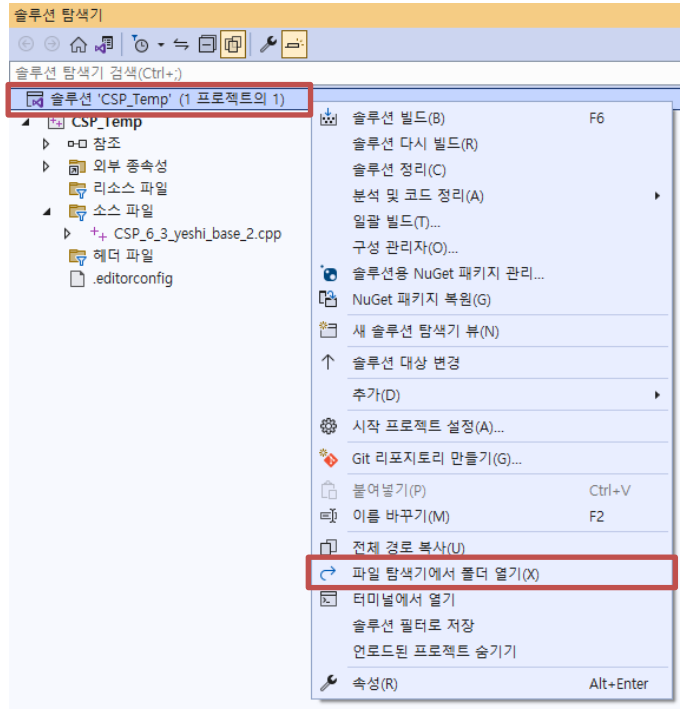
# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 먼저, 파일입니다. input.txt를 예로 들어 볼께요.
  방금 전 코드를 아래와 같이 수정해 봅시다:
  - 뭔가 생긴 것만 봐서는,
    실행하면 input.txt에 내가 적은 숫자(에 대한 10진수 표현)가 들어 있을 것 같습니다

  - 확인해 보려면...
    - 솔루션 탐색기의 내 솔루션 폴더를 우클릭

    - '파일 탐색기에서 폴더 열기(X)' 클릭

      ...하면 열리는 폴더에 들어 있을 거예요!



# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 이번에도 잠깐 << **연산자**를 괴롭혀 볼까요?
  아래의 두 부분에 마우스 포인터를 대 보고, 차이점을 확인해 봅시다.

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main()
{
    int number;

    cout << "숫자를 입력하세요>";
    cin >> number;

    // 오른쪽 상자에서 계속...
```

```
 // ...왼쪽 상자 다음에 이어짐

    ofstream os{"input.txt"}; // 열기
    os << number;
    os.close(); // 저장 후 닫기

    cout << "입력한 숫자는 ";
    cout << number;
    cout << "입니다." << endl;

    return 0;
}
```

# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 네, 지금의 경우 우항의 **형식**이 동일해서 그런지
  두 팝업이 완벽하게 똑같이 생긴 것을 구경할 수 있었습니다.
  - 그래서 검은 창에 출력하는 스타일이랑 동일한 형태로 숫자가 파일에 적혀 있었지요

# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 파일 입력은 어떨까요? 이번에는 여러분이 직접 시도해 봅시다.
  주석으로 표시한 부분에 해당 작업을 수행하는 **문장**들을 적어 보세요:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main()
{
    int number;

    cout << "숫자를 입력하세요>";
    cin >> number;

    // 오른쪽 상자에서 계속...
```

```
    // ...왼쪽 상자 다음에 이어짐

    ofstream os{"input.txt"}; // 열기
    os << number;
    os.close(); // 저장 후 닫기

    // TODO: input.txt에서 숫자 읽어서
    //       변수 number 자리에 담기

    cout << "입력한 숫자는 ";
    cout << number;
    cout << "입니다." << endl;

    return 0;
}
```

# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

여백의 아름다움

# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 동일한 '파일' 개념을 쓰니까 두 줄이 비슷하고,
  동일한 '입력' 개념을 쓰니까 한 줄이 비슷합니다.

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main()
{
    int number;

    cout << "숫자를 입력하세요>";
    cin >> number;
```

```
    ofstream os{"input.txt"}; // 열기
    os << number;
    os.close(); // 저장 후 닫기

    ifstream is{"input.txt"}; // 열기
    is >> number;
    is.close(); // 닫기

    cout << "입력한 숫자는 ";
    cout << number;
    cout << "입니다." << endl;

    return 0;
}
```

# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 마지막으로, 편하게 쓸 수 있던 std::string 입니다.
  대충 개념은 알았으니, 이번엔 강사만 해 볼께요:

```
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

int main()
{
    int number;

    cout << "숫자를 입력하세요>";
    cin >> number;

    ostringstream os;
    os << number;

    string result = os.str();

    istringstream is(result);
    is >> number;

    cout << "입력한 숫자는 ";
    cout << number;
    cout << "입니다." << endl;

    return 0;
}
```

# 기본 형식 값을 다른 방법으로 입력받기 / 출력하기

- 요약하면...
  - '어디에' 출력하는지가 달라지면서
    출력을 준비하거나 마무리하는 방법만 조금씩 차이가 생겼지
    출력용 Code 자체는 언제나 동일한 것을 사용하는 것을 관찰할 수 있었습니다

  - 결론부터 말해서, 우리가 배웠던 옛날 C에선 이런 건 못 해요

  - 왜 C++는 이게 되는가에 대해서는 후반부에 천천히 짚어 보고,
    일단은 방금 느꼈던 점만 적당히 간직해 두도록 합시다

# 마무리

- 요약하면...
  - '어디에' 출력하는지가 달라지면서
    출력을 준비하거나 마무리하는 방법만 조금씩 차이가 생겼지
    출력용 Code 자체는 언제나 동일한 것을 사용하는 것을 관찰할 수 있었습니다

  - 결론부터 말해서, 우리가 배웠던 옛날 C에선 이런 건 못 해요

  - 왜 C++는 이게 되는가에 대해서는 후반부에 천천히 짚어 보고,
    일단은 방금 느꼈던 점만 적당히 간직해 두도록 합시다

- '쉬워지면서 동시에 어려워졌'지요?
  다음 시간부터 본격적으로 다양한 C++ 코드들을 작성해 볼께요.
  - 오늘은 느낀 점 적고 집에 갑시다