Day-4

Hello World

printf를 사용한 C 코드

```
#include<stdio.h>

int main()
{
            printf("Hello World!\n");
            return 0;
}
```

- printf()
  - 표준 출력
  - 변환 명세를 사용하여 형식에 맞춰 출력해주는 함수
  - 변환 명세 예시
    - %d, %f, %s 등

streams를 사용한 C++ 코드

```
#include<iostream>

int main()
{
        std::cout << "Hello World!\n";
        return 0;
}
```

- cout

- 표준 출력 스트림을 나타냄

global std namespace의 사용

cout 예제는 다음과 같이 global std namespace를 사용하여 작성 할 수 있음

```
#include<iostream>

using namespace std;

int main()
{
        cout << "Hello World!\n";
        return 0;
}
```

- 공간 확보와 가독성 향상을 위해 std namespace를 생략할 수 있음

I/O Stream

- std::cout
  - std::cout은 출력 스트림의 예시
  - 데이터가 대상으로 리디렉션됨
  - 이 경우 대상은 표준 출력
- I/O stream을 선호해야 하는 이유
  - Type-safe(타입 안전)
    - I/O stream에 의해 제공되는 객체 유형은 컴파일러에 의해 정적으로 알려져 있음
    - 반면 printf는 %필드를 사용하여 동적으로 유형을 파악
  - 적은 오류 발생
    - I/O stream을 사용하면 I/O stream에 전달된 실제 객체와 일치해야 하는 중복된 % 토큰이 없음
    - 중복성을 제거하면 오류 클래스가 제거됨

    - 확장 가능
        - I/O stream 메커니즘을 통해 기존 코드를 손상시키지 않고 새로운 사용자 정의 유형을 I/O stream에 전달 가능
    - 성능
        - 올바르게 사용한다면 C I/O(printf, scanf 등)보다 빠를 수 있음
- 일반적인 C 오류
    - 매개변수 수를 제대로 입력하지 않은 경우

```
printf("long phrase %d long phrase %d",3);
// %d 변환 명세가 문자열 내에 2개 존재
// 매개변수를 한개만 입력했기 때문에 error 발생
```

    - 잘못된 형식 사용

```
int a = 3;
printf(" %f",a);
// %f 변환 명세는 float형 변수를 출력하기 위함
// int형 변수 a를 매개변수로 했기 때문에 error 발생
```

    - %c 변환 명세는 선행 공백을 자동으로 건너뛰지 않음

```
scanf("%d", &var1);
scanf(" %c", &var2);
```

std::print

- C++ 23은 포매터 문자열을 기반으로 하는 std::print 함수의 개선된 버전을 도입
- C++ stream의 모든 이점을 제공하고 코드의 복잡함이 줄어듬

```
#include<print>

int main()
{
        std::print("Hello World! {}, {}, {}\n", 3, 4ll, "aa")
```

```
        // "Hello World! 3 4 aa" 출력
}
```

- C++ 23 표준이 넓게 채택되면 기본 출력 방식이 될 것임