.h, .hpp, .cpp

Status	시작 전

`h`와 `hpp` 확장자의 차이

1. `h` 파일
    - **용도**: 일반적으로 **헤더 파일**로 사용됩니다.
    - **주로 선언**: 함수 선언, 클래스 정의, 상수 정의 등을 포함합니다.
    - **내용**: 보통 함수, 클래스, 변수의 **선언부만** 작성하고, 구현부는 `.cpp` 파일에 작성합니다.
    - **예시**:

```
// MyClass.h
#ifndef MYCLASS_H
#define MYCLASS_H

class MyClass {
public:
    void print();
};

#endif
```

2. `hpp` 파일

- **용도**: **템플릿 클래스/함수** 또는 **인라인 함수**의 구현을 포함하는 헤더 파일로 사용됩니다.
- **주로 선언 + 구현**: 선언과 구현을 **한 파일**에 작성합니다.
    - 템플릿의 경우, 컴파일러는 인스턴스화될 때 코드가 필요하므로 **모든 구현부**가 헤더에 포함되어야 합니다.
- **내용**: 일반적으로 템플릿 클래스나 인라인 함수와 같은 코드가 포함됩니다.
- **예시**:

```
// MyClass.hpp
#ifndef MYCLASS_HPP
#define MYCLASS_HPP

template <typename T>
class MyClass {
public:
    void print(T value) {
        std::cout << "Value: " << value << std::endl;
    }
};

#endif
```

왜 `hpp`가 필요한가?

템플릿이나 인라인 함수의 구현은 `cpp` 파일에 작성할 수 없습니다. 이유는 다음과 같습니다:

- **템플릿**: 템플릿은 컴파일 타임에 실제 데이터 타입에 따라 코드가 생성되므로, 모든 구현이 헤더 파일에 있어야 합니다.
- **인라인 함수**: 구현이 헤더에 있어야 **인라인**으로 처리됩니다.

빌드 시 `h`와 `hpp` 파일의 처리

- **`h`와 `hpp` 파일은 빌드할 필요가 없습니다**.
- `h`와 `hpp` 파일은 **헤더 파일**이기 때문에 실제 **컴파일 단위**가 아닙니다.
- 빌드 과정에서 헤더 파일은 **컴파일러가 포함**(`#include`)하여 `main.cpp`나 다른 소스 파일에 합쳐지게 됩니다.

빌드 과정 정리

1. **헤더 파일 (`h`, `hpp`)**:
    - 선언 및 구현을 포함하고 있으며 **별도 컴파일되지 않습니다**.
    - `#include` 지시어를 통해 필요한 소스 파일에 포함됩니다.

2. **소스 파일 (`cpp`)**:
    - **실제 컴파일되는 단위**입니다.
    - 컴파일러는 소스 파일을 입력으로 받아 기계어 코드로 변환합니다.
3. **예시 명령어**:

```
g++ main.cpp -o main
```

- `main.cpp` 내에서 `#include "Student.h"` 와 `#include "StudentManager.hpp"` 를 통해 필요한 헤더 파일이 포함되므로, 컴파일러가 모두 처리합니다.