그런즉 너희가 먹든지 마시든지 무엇을 하든지 다 하나님의 영광을 위하여 하라 (고전 10:31)

---


**NOTE:** The following materials have been compiled and adapted from the numerous sources including my own. Please help me to keep this tutorial up-to-date by reporting any issues or questions. Send any comments or criticisms to idebtor@gmail.com Your assistances and comments will be appreciated.

---

# Lab 4a- C 전처리기

## 단원

1. 전처리기의 소개
2. Header Files
3. Macros
4. Conditionals
5. 그 외

- `lab` 자료의 설명과 함께 참조하면 효과적이다.

## Step 1. 전처리기의 소개

C 전처리기(Preprocessor)는 C 컴파일러가 컴파일 전에 프로그램을 변환하기 위해 자동으로 사용하는 매크로 프로세서(Macro Processor)이다. 매크로 프로세서라고 불리는 이유는 매크로를 정의할 수 있기 때문이다. 매크로를 통해 코드를 간략하게 표현할 수 있다.

아래 모든 자료는 https://gcc.gnu.org/onlinedocs/cpp/ 공식 문서에서 참조 가능하다.

## Step 2. Header Files

#include 를 통해 사용이 가능하다. 사용 방법은 총 두가지로:

```
#include <filename.h>
#include "filename.h"
```

위 방법은 시스템 디렉토리에 존재하는 헤더 파일을 접근하고, 아래 방법은 사용자가 생성한 파일을 접근한다. 컴파일 할 때 -I 옵션을 추가하여 디렉토리의 경로를 알려준다. <> 꺾쇠를 사용할 경우, 디폴트(default)로 시스템 디렉토리(/usr/include/sys)를 탐색하고, "" 따옴표의 경우, 따옴표에 포함된 경로를 먼저 탐색한다.

```
gcc sourcefile -I filepath
```

#include 는 헤더 파일의 내용을 소스 코드 파일에 그대로 가져온다. 그렇기에, 중복되어 #include 되는 경우를 방지하기 위해 #ifndef 로 파일을 감싼다.

```
/* File: foo.cpp */
#ifndef FILE_FOO_SEEN     // 파일의 접근 여부
#define FILE_FOO_SEEN

contents

#endif                    // 파일의 끝, !FILE_FOO_SEEN
```

해당 헤더 파일이 접근이 되면 FILE_FOO_SEEN 이 정의(define)되어 다시 접근되었을 때 넘기게 된다.

#ifndef 외에 다른 방법으로 #pragma once 가 존재한다.

## Step 3. Macros

#define 을 사용한다.

```
#define macroname tokensequence
```

매크로 이름을 정의하고, 대체할 토큰을 정의한다. 일반적으로 매크로 이름은 대문자로 작성한다.

아래와 같이 여러 값을 매크로로 정의하는 것도 가능하다.

```
#define NUMBERS 1, \
                2, \
                3
```

이외에 함수(Function), 인자(Argument)를 매크로로 정의할 수 있다. `#define` 도 `#include` 와 비슷하게, 컴파일되기 전에 `macroname` 을 모두 `tokensequence` 로 대체하는 작업을 한다.

## Step 4. Conditionals

조건부(Conditionals)는 컴파일러로 전달되는 코드 내용을 포함할지 여부를 선택하도록 전처리기에 지시하는 명령어이다.

C 전처리기의 조건부는 어떤 면에서 C 의 if 조건문과 유사하지만, 이들 사이의 차이를 이해하는 것이 중요하다. if 조건문은 프로그램을 실행하는 동안 조건이 테스트됩니다. 이는 프로그램이 실행되는 데이터에 따라 실행 간에 다르게 동작하도록 허용한다. 반면에, 전처리기에 조건부 지시문의 조건은 프로그램이 컴파일 될 때 테스트된다. 컴파일 당시 상황에 따라 다른 코드가 프로그램에 포함될 수 있도록 하는 것이 목적이다.

조건부를 사용하는 대표적인 이유 3 가지:

- 운영체제에 따라 다른 코드를 필요로 하는 경우.
- 동일한 소스 파일을 다른 방법으로 컴파일하기 원하는 경우.
- 조건을 항상 거짓으로 설정하여, 참조를 할 수 있도록 주석과 같은 역할을 수행하는 경우.

가장 간단하고 대표적인 예시:

```
#ifdef MACRO

controlled text

#endif /* !MACRO */
```

이 `MACRO` 들은 gcc 컴파일러의 `-D` 옵션으로 접근 가능하다. 하지만, 본 수업과정에서는 주로 `0` 과 `1` 을 (조건) 사용한다. `0`(거짓, false)는 `#else`, 또는 `#endif` 로 조건부가 끝날 때 까지의 코드를 가린다.
`1` 은 그 반대의 작업을 수행한다.

```
#if condition

controlled text

#else

controlled text

#endif
```

`#else` 를 사용해 `#if` 의 조건이 거짓인 경우를 다른 코드를 실행하도록 할 수 있다. `else if` 의 경우 `#elif` 를 사용한다.

위 3 번째 이유로 주석을 유지하기 원하는 경우(주석은 전처리 과정에서 사라진다), `#if 0` 로 정의하여 유지할 수 있다.

## Step 5 그 외

이 단원에서는 위에서 배운 preprocessor 들의 기능들을 혼합하여 사용하는 방법을 다룬다.

강의 자료나 제공되는 스켈레톤 코드에서 자주 등장하는 예제 몇개를 설명한다:

- Conditionals

```
#if 0

controlled text

#else

controlled text   // your code here

#endif
```

뼈대로 `#if ~ #else` 에 코드가 주어지고, `#else ~ #endif` 까지 코드를 직접 작성하면서 위 `#if ~ #else` 코드를 참조 할 수 있도록 되어있다.

- Conditionals & Macros

```
#ifdef DEBUG
#define DPRINT(func) func;
#else
#define DPRINT(func) ;
#endif

context

DPRINT(cout << “DEBUG - …
```

gcc 로 컴파일을 할 때 `-D` 옵션의 인자로 `DEBUG` 를 지정할 경우, `#ifdef` 는 참이 된다.

```
gcc sourcefile -DDEBUG
```

참이 되면, 매크로로 `DPRINT()`로 감싸여진 코드가 덮어쓰면서 실행이 된다. 거짓인 경우, 빈칸이 된다.

- 그리고, 전처리가 된(실행 파일이 되기 전) 상태를 확인하기 위해서는 `-E` 옵션을 사용한다.

```
gcc -E main.cpp
```

**축하합니다! Congratulations!**

- 과제 진행은 `main.cpp` 상단에 주석을 따라 진행한다.
- 과제 설명에 따라 `main.i` 파일의 설명을 캡처하여 제출한다.

---

**제출파일 목록**

- `main.cpp`
- 이미지 2장

---

*One thing I know, I was blind but now I see. John 9:25*