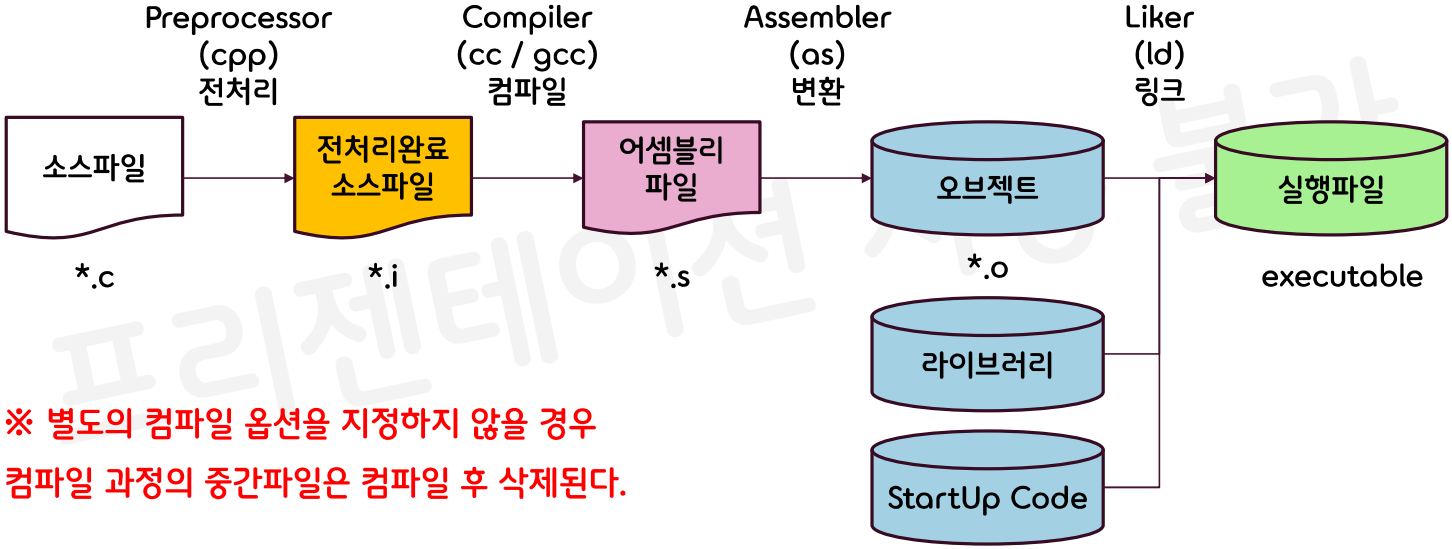
작성자 : 김기희 (cupid072@naver.com)

선행처리자(Preprocessor)

- 학습목표
  - 선행처리기와 선행처리 지시자에 대해 설명할 수 있다.
  - 선행처리 지시자의 종류를 기술할 수 있다.
  - #include 지시자의 기능을 설명할 수 있다.
  - #define 지시자의 기능을 설명할 수 있다.
  - 중복 포함 방지를 위한 include guard를 사용할 수 있다.
  - 매크로 상수(Object-liked Macro)에 대해 설명할 수 있다.
  - 매크로 상수(Function-liked Macro)에 대해 설명할 수 있다.
  - 매크로 연산자 #과 ##의 기능에 대해 설명할 수 있다.
  - 조건부 컴파일 지시자를 사용할 수 있다.

선행처리자(Preprocessor)

- 선행처리자(Preprocessor)는 소스파일을 컴파일(Compile)하기 전에 컴파일 환경에 맞게 소스코드를 편집하는 기능을 수행한다.



※ 별도의 컴파일 옵션을 지정하지 않을 경우
컴파일 과정의 중간파일은 컴파일 후 삭제된다.

선행처리 지시자

- 선행처리 지시자
  - 컴파일러에 의해 소스 코드를 컴파일 하기 전에 이뤄지는 선행처리자에게 특정 작업을 지시하기 위한 명령
  - 반드시 #으로 시작하여야 한다.
  - 하나의 지시자는 하나의 라인으로 작성한다.
  - 명령문의 끝에는 세미 콜론을 붙이지 않는다.

- 선행 처리 지시자의 종류

기능	설명
매크로 확장 지시자	매크로 또는 매크로 함수를 정의하거나 정의를 해제하기 위한 지시자
조건부 컴파일 지시자	주어진 조건에 따라 소스 코드를 변경하거나 컴파일 여부를 제어하기 위한 지시자
외부 파일 지시자	특정 파일 병합 지시자로 지정된 파일 내용을 현재 파일에 병합하기 위한 지시자
기타 지시자	컴파일 과정을 지원하기 위한 지시자

Macro와 Macro 함수

- macro(매크로)
  - 컴퓨터 프로그래밍에서의 매크로는 자주 사용하는 여러 개의 명령어를 묶어서 하나의 키 입력 동작으로 만든 것을 의미한다.
  - 매크로는 복잡하거나 반복되는 작업을 단순화하기 위한 목적으로 사용된다.
  - C/C++에서의 매크로는 어떤 키워드를 코드로 치환하는 것을 목적으로 한다.
  - 위에서 기술한 "어떤 키워드"는 C/C++ 소스코드에서 기술되는 모든 코드가 대상이 된다.

- macro의 C언어 표준에 의거한 이름은 다음과 같이 구분된다.
  - 유사 개체 매크로 (object-like macro) : 일반적으로 단순 매크로라고 칭함
  - 유사 함수 매크로(function-like macro) : 일반적으로 매개변수 매크로라고 칭함

유사 개체 매크로 (object-like macro)

- 유사 개체 매크로(object-like macro)는 단순 매크로(simple macro)라고도 하며 소스 코드의 특정 키워드를 대체하기 위한 매크로이다.

- 매크로에 의해 대체 가능한 항목은 식별자, 키워드, 숫자상수, 문자상수, 문자열 리터럴, 연산자, 제어문 등 모든 것이 대상이 될 수 있다.

- 유사 개체 매크로 정의 형식

  #define 식별자 대체목록

```
#define MAX 100
...
int ary[MAX] = { 0 };
int ary[100] = { 0 };
```

```
#define MAX = 100
...
int ary[MAX] = { 0 };
int ary[= 100] = { 0 };    // ERROR
```

유사 개체 매크로 (object-like macro)

- 매크로 사용의 장점
  - 프로그램의 가독성을 높여준다.
  - 프로그램의 유지 보수성을 높여준다.
  - 일관성 없는 코드나 오타를 방지할 수 있다.

- 다양한 매크로의 예)

```
#define BOOL int
#define BEGIN {
#define END }
.....
while (BOOL)
BEGEN
    .....
END
```

매크로의 대체 목록은 비어 있을 수 있다.

```
#define DEBUG
```

대체목록이 비어 있는 매크로는 매크로 정의 여부를 판단한다.

```
#ifdef
    printf("디버깅 정보 출력\n");
#endif
```

유사 함수 매크로(function-like macro)

- 유사함수 매크로(function-like macro)는 매개변수 매크로(Parameterized macro)라고도 하며 소스 코드 내의 함수를 대체하기 위한 매크로이다.

- 유사 함수 매크로 정의 형식

  #define 식별자(인수목록...) 대체목록

  #define MAX(x, y) ((x) > (y) ? (x) : (y))
  .....
  int max_value = MAX(a, b);

- 선행처리기에 의해 아래와 같이 치환된다.

  int max_value = ((x) > (y) ? (x) : (y));

유사 함수 매크로(function-like macro)

- 유사 함수 매크로 사용의 장점

  - 실제 함수 호출이 아닌 함수 수행 부분을 코드로 대체함으로써 수행 속도를 높일 수 있다.
  - 매크로 함수의 매개변수는 일반적으로 generic이다. "generic"이라는 것은 함수 매개변수와는 달리 매크로 함수의 매개변수는 자료형이 없다는 것이다. 이것은 매크로를 통해 서로 다른 자료형의 데이터 간에도 최대 값을 찾을 수 있음을 의미한다.

- 유사함수 매크로 사용의 단점

  - 함수 호출 대신 코드로 대체 되므로 실행파일의 크기가 증가한다.
  - 매크로 함수의 매개 변수에 대한 형식 체크가 지원되지 않는다.
  - 실제 함수가 아니므로 포인터를 가지지 않는다.
  - 매크로는 매개변수의 값을 여러 번 평가할 수 있다.

유사 함수 매크로(function-like macro)

- 유사 함수 매크로 정의 시 주의사항

① 치환 목록에서 사용되는 매개변수는 괄호를 사용하여 기술한다.

```
#define CUBE(x) (x * x * x)
int a = CUBE(2 + 1);          // 3 * 3 * 3 값을 원하는가?
```

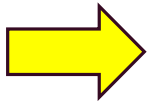
② 치환 목록은 되도록이면 괄호를 사용하여 기술한다.

```
#define CUBE(x) (x) * (x) * (x)
int a = 81 / CUBE(3)          // 81 / 27 값을 원하는가?
```

유사 함수 매크로(function-like macro)

③ 복수 구문의 매크로는 do ~ while 루프로 감싼다.

```
#define SWAP(x, y) \
    tmp = x; \
    x = y; \
    y = tmp
.....
```



```
#define SWAP(x, y) \
do { \
      tmp = x; \
      x = y; \
      y = tmp; \
} while (0)
```

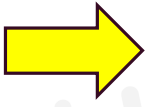
```
int x = 10;
int y = 20;
int tmp = 0;
if (1)  SWAP(x, y);          // if (1) tmp = 1; x = y; y = tmp;
printf("x : %d\n", x);       // 조건이 참 일 때 수행할 문장이 복수의 명령으로 구성되었으므로 ERROR
printf("y : %d\n", y);
```

유사 함수 매크로(function-like macro)

④ 매크로 함수의 매개변수는 여러 번 평가 될 수 있다.

매크로 함수의 매개변수나 치환목록에서의 매개변수에 대한 부작용(side effect)는 예상치 못한 결과를 초래한다.

```
#define CUBE(x) ((x) * (x) * (x))
.....
int i = 2;
int a = CUBE(++i);
```



```
int cube(int x) {
    return x * x * x;
}
int i = 2;
int a = cube(++i);
```

※ 위의 예제는 하나의 Sequance(실행명령) 내에서 하나의 변수에 대하여 한 번 이하의 부작용(side effect) 만을 가지도록 하여야 한다는 규칙에 위배된다.

외부 파일 포함 지시자

- 외부 파일 포함 지시자는 외부의 파일 내용을 현재 파일에 포함시키기 위해 사용되는 지시자이다.
- #include 지시자에 의해 지정된 파일의 데이터는 #include 지시자가 기술된 위치에 포함된다.
- #include 지시자는 확장자가 .h인 헤더 파일 만이 아니라 모든 파일에 사용할 수 있으나 일반적으로 헤더 파일이나 C 소스 파일을 대상으로 한다.
- #include 지시자에 지정되는 파일의 형식은 C 소스파일과 동일한 형식인 텍스트 파일 이어야 한다.

- 외부 파일 포함 지시자 형식

#include <외부파일경로>	표준 라이브러리 헤더 디렉토리에서 탐색
#include "외부파일경로"	현재 디렉토리에서 우선 검색, 없을 경우 표준 헤더 디렉토리에서 탐색

외부 파일 포함 지시자

- 외부 파일 포함 지시자 사용시 주의사항

- #include 를 뒤 따르는 파일경로는 <> 또는 ""를 사용하여 기술한다.
  - <> : 표준 헤더 디렉토리에서 탐색하여 포함
  - "" : 현재 디렉토리를 우선 탐색, 없으면 표준 헤더 디렉토리를 탐색하여 포함

- #include 를 뒤 따르는 <> 또는 "" 내에는 파일 경로를 기술한다.
  예를 들어 dir 디렉토리 하위의 file.c 파일을 포함하고자 하는 경우

```
#include "dir/file.c"
```

- 호환성을 위해 Windows 디렉토리 구분자가 아닌 Posix 디렉토리 구분자를 사용한다.

```
#include "dir\file.c"          // NG
#include "dir/file.c"          // OK
```

조건부 컴파일

- 조건부 컴파일이란? 매크로 정의를 사용하여 지정한 조건에 따라 코드의 일정 부분을 컴파일 할 것인가를 결정하는 기법을 말한다.

- 조건부 컴파일 지시자

  #if

  #ifdef

  #ifndef

조건부 컴파일

- #if 지시자
  - #if지시자에 뒤 따르는 조건식을 평가하여 결과가 참이면 if 지시자 내의 문장을 수행한다.
  - #else 지시자는 기술할 수 도 있고 기술하지 않을 수 도 있으며 기술하는 경우 마지막에 기술되어야 한다.
  - C 언어의 if 문과는 달리 반드시 #endif 지시자를 사용하여 #if 조건부 컴파일의 끝을 명시하여야 한다.

형식 1)	형식 2)	형식 3)	형식 4)
#if 조건식 ..... #endif	#if 조건식 ..... #else ..... #endif	#if 조건식1 ..... #elif 조건식2 ..... #elif 조건식n ..... #endif	#if 조건식1 ..... #elif 조건식2 ..... #elif 조건식n ..... #else ..... #endif

조건부 컴파일

- #ifdef 지시자
  - #ifdef 지시자에 뒤 따르는 **매크로가 정의되어 있는가?** 를 판단한다.
  - #else 지시자는 기술할 수 도 있고 기술하지 않을 수 도 있으나 일반적으로는 기술하지 않는다.
  - #endif 지시자를 사용하여 #if 조건부 컴파일의 끝을 명시하여야 한다.

형식 1)	형식 2)
#ifdef 매크로 ..... #endif	#ifdef 매크로 ..... #else ..... #endif

조건부 컴파일

- #ifndef 지시자
  - #ifndef 지시자에 뒤 따르는 **매크로가 정의되어 있지 않은가?** 여부를 판단한다.
  - #else 지시자는 기술할 수 도 있고 기술하지 않을 수 도 있으나 일반적으로는 기술하지 않는다.
  - #endif 지시자를 사용하여 #if 조건부 컴파일의 끝을 명시하여야 한다.
  - 주로 중복 선언 방지를 위한 inclusion guard에 사용된다.

형식 1)	형식 2)
#ifndef 매크로 ..... #endif	#ifndef 매크로 ..... #else ..... #endif

조건부 컴파일

- inclusion guard(중복 포함 방지)

  inclusion guard(중복 포함 방지)란? include에 의한 외부 파일 포함으로 인해 발생하는 중복 선언을 방지하기 위한 기법을 말한다.

- 헤더 파일에 포함되는 내용은 다음과 같다.
  - 외부에서 선언 및 정의된 전역변수
  - 외부에서 선언 및 정의된 함수의 원형(prototype)
  - 구조체나 공용체 자료형 선언
  - 매크로 지시자

- 구조체나 공용체 자료형은 헤더 파일에서 선언하는 것이 일반 적이며 동일한 헤더 파일을 여러 파일에서 include 하는 경우 중복 선언 오류가 발생한다.

조건부 컴파일

- inclusion guard(중복 포함 방지) 방법

```
file.h 헤더 파일에 내용 ----------------------------------------------------------
#ifndef FILE_H_              // FILE_H_ 매크로가 정의되어 있지 않다면
#define FILE_H_              // FILE_H_ 매크로를 정의한다.
#include "file.h"            // 외부 파일을 포함하거나 매크로를 정의한다.
.....
#endif
-------------------------------------------------------------------------
```

- C 소스 파일에서 한 번이라도 file.h 헤더 파일을 include 하면 FILE_H_ 매크로가 정의된다.
- 최초의 include 에 의해 정의된 FILE_H_ 매크로에 의해 이후 include 작업이 수행되지 않게 된다.

매크로 연산자

- 매크로 함수에서 사용가능한 연산자는 다음과 같다.
- # # 연산자를 뒤 따르는 토큰(Token)을 문자열로 치환
- ## ## 앞과 뒤의 토큰을 합쳐 하나의 토큰(Token)을 생성한다.

```
#define PRINT_VAR(x) printf(#x " = %d\n", x)
#define CONCAT(x, y) (x ## y)
.....
int a1;
int a2;
CONCAT(a, 1) = 10;                  // (a1) = 1;
CONCAT(a, 2) = 20;                  // (a2) = 2;
PRINT_VAR(a1 + a2);          // printf("a1 + a2" " = %d\n", a1 + a2);
PRINT_VAR(a2 - a1);          // printf("a2 - a1" " = %d\n", a2 - a1);
```

기타 선행 처리 지시자

- #undef
  - #define 지시자에 의해 정의된 매크로를 무효화 한다.
  - #define 매크로는 정의된 위치에서 #undef 지시자에 취소되는 위치까지 유효하다.
  - #define 매크로가 #undef 지시자에 의해 취소되지 않는다면 파일의 끝 까지 유효하다.

- #pragma
  - 컴파일러의 컴파일 방법을 세부적으로 제어하고자 할 때 사용한다.
  - #pragma 지시자는 directive-name을 사용하여 컴파일러에게 어떤 기능을 제어할 것인가를 알린다.

```
#pragma once                    // 오직 한 번만 include 하도록 한다. (Visual Studio 에서만 유효)
#pragma pack(1)                 // 바이트 정렬 단위를 1바이트로 설정
#pragma warning(disable:4996)   // 4996번 경고 메시지를 무시한다. (Visual Studio 에서만 유효)
```

※ #pragma 지시자는 비정규 지시자로 컴파일러에 따라 지원 여부가 다르다.

미리 정의 된 매크로 (Built-in macro)

- 미리 정의된 매크로 (Built-in macro)는 컴파일러에 의해 미리 정의되어 사용이 약속되어 있는 매크로를 말한다.
- 미리 정의된 매크로를 재정의(redefine) 하거나 해제(undefine)를 하는 경우 예상치 못한 오류가 발생할 수 있으므로 임의로 재정의 하거나 해제하지 않도록 하여야 한다.
- 컴파일과 관련된 미리 정의된 매크로는 다음과 같다.

매크로	설명
__FILE__	파일명
__FUNCTION__	함수명
__LINE__	라인번호
__DATE__	컴파일 날짜
__TIME__	컴파일 시간

- __FILE__과 __LINE__ 매크로는 #line 지시자를 사용하여 제어가 가능하다.

```
#line 100              // #line 지시자에 의해 #line 지시자 다음의 라인 값이 100으로 설정된다.
#line 100 "main"       // #line 지시자에 의해 라인 값과 함께 __FILE__ 매크로의 값을 "main"으로 설정
```

작성자 : 김기희 (cupid072@naver.com)

분할 컴파일

분할 컴파일

- 하나의 프로그램을 여러 개의 소스 파일로 나누어 독립적으로 컴파일 하면 디버깅이 쉽고 유지 보수와 코드 재활용에 유리하다.

  - 분할된 파일은 독립적으로 컴파일과 디버깅이 가능해야 한다.
  - 분할 컴파일 된 개체 파일은 링크되어 하나의 프로그램이 된다.
  - 파일 간에 전역 변수를 공유할 때는 extern 지정자를 사용한다.
  - 파일 간의 전연 변수 공유를 제한할 때는 static 지정자를 사용한다.
  - 헤더 파일의 중복 포함 문제는 조건부 컴파일 지시자로 해결한다.

헤더 파일의 구성

- 헤더 파일에는 다음과 같은 내용을 포함한다.

  - 함수의 선언부
  - 외부 변수의 선언
  - 구조체 자료형의 선언
  - 열거형의 선언
  - typedef에 의한 자료형의 재정의
  - 전처리지시자