
Preprocessor (C/C++)

ISL / 안재원

목차

1 Preprocessor

2 Preprocessor directives

- `#if #elif #else #endif`
- `#ifdef #ifndef`
- `#define #undef`
- `#error`
- `#include`
- `#line`
- `#import`
- `#using`
- `#pragma`

1. Preprocessor 란?

- 여러 파일로 구성된 소스파일을 컴파일 하기 전에 수행되는 작업이다.
- 전처리기 지시문은 #으로 시작 되며, 대표적으로 #include, #define이 있다.



1. 전처리기 지시문의 종류

- `#if #elif #else #endif`
- `#ifdef #ifndef`
- `#define #undef`
- `#error`
- `#include`
- `#line`
- `#import`
- `#using`
- `#pragma`

2. #if #elif #else #endif

- if 함수와 같은 형식으로 사용되지만, 마지막에 #endif가 반드시 필요하다.

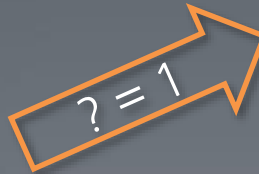
전처리 과정에서 알 수 있는 내용을 조건으로 정해야 한다.

- 조건부로 컴파일을 수행할 때 사용한다.

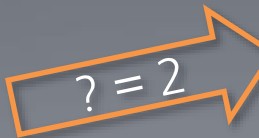
if([조건1])		#if [조건1]
{...}		...
else if ([조건2])	↔	#elif [조건2]
{...}		...
else		#else
{...}		...
		#endif

2. #if #elif #else #endif

```
#define x ?  
  
#if x == 1  
#define a 1  
#define b 2  
#define c 3  
  
#elif x == 2  
#define a 10  
#define b 20  
#define c 30  
  
#else  
#define a 100  
#define b 200  
#define c 300  
  
#endif
```



```
1  
2  
3  
계속하려면 아무 키나
```



```
10  
20  
30  
계속하려면 아무 키나
```



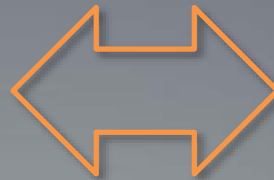
```
100  
200  
300  
계속하려면 아무 키나
```

3. #ifdef #ifndef

- #if defined를 줄인 전처리기 지시문이다.
- 뒤에 나오는 *identifier*가 #define으로 선언되어 있는지 아닌지 판단하여 조건부로 컴파일 한다.

#if defined(*identifier*)

#if !defined(*identifier*)



#ifdef *identifier*

#ifndef *identifier*

3. #ifdef #ifndef

```
#define x 2

#if x == 1
#define a 1
#endif

int main()
{
#ifdef a
#define a 100
#endif
    printf("a=%d\n", a);
    return 0;
}
```



100
계속하려면 아무 키나 누르십시오 . . .

4. #define #undef

- #define 구문

#define identifier token-string

#define identifier(identifier, ..., identifier) token-string

- #undef 구문

#undef identifier

4. #define #undef

```
#define a 10
#define b a+10
#define c (a+10)

int main()
{
    printf("%d\n",a);
    printf("%d\n",b+10);
    printf("%d\n",c+10);
    return 0;
}
```



```
#define a 10
#define b a+10
#define c (a+10)

int main()
{
    printf("%d\n",10);
    printf("%d\n",10+10+10);
    printf("%d\n",(10+10)+10);
    return 0;
}
```



```
10
110
200
계속하려면 아무 키나 누르십시오 . . .
```

4. #define #undef

```
#define CHECK(x, ...) if ((x)) { printf(__VA_ARGS__); }  
  
int main()  
{  
    CHECK(1, "define example\n");  
  
    #undef CHECK  
    #ifndef CHECK  
        printf("CHECK is undefined\n");  
    #endif  
    return 0;  
}
```

Variadic 매크로로
줄임표 위치의 모든
인수로 대체 된다.



```
define example  
CHECK is undefined  
계속하려면 아무 키나 누르십시오 . . .
```

5. #error

- 컴파일 할 때 token-string의 메시지를 출력하고, 컴파일을 종료합니다.
- #error 구문
#error token-string

```
#if !defined(_M_X64)
#error x64컴퓨터가 아닙니다.
#endif

int main()
{
    printf("Hi\n");
    return 0;
}
```

x64 프로세서에서
사용되는 매크로

```
1>----- 빌드 시작: 프로젝트: ttteesstt, 구성: Debug Win32 -----
1> ttteesstt.cpp
1>c:\users\hong\documents\code\ttteesstt\ttteesstt\ttteesstt.cpp(7): fatal error C1189: #error : x64컴퓨터가 아닙니다.
===== 빌드: 성공 0, 실패 1, 최신 0, 생략 0 =====
```


6. #include

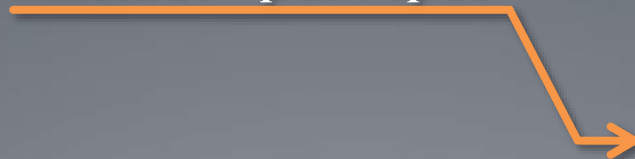
- 해당 정의를 소스 파일에 추가 할 때 사용합니다.

- #include 구문

#include "path-spec"

#include <path-spec>

- 
1. 소스파일과 동일한 디렉터리 탐색
2. include 파일이 열린 반대 순서로 탐색
3. 컴파일러 옵션의 지정된 경로를 탐색
4. INCLUDE 환경변수가 지정된 경로를 탐색

- 
1. 컴파일러 옵션의 지정된 경로를 탐색
2. INCLUDE 환경 변수가 지정한 경로를 탐색

7. #line

- 컴파일러 내부적으로 저장된 라인 수와 파일 이름을 변경할 때 사용한다.
- #line 구문


#line digit-sequence ["filename"]

7. #line




__LINE__ - 현재 소스파일의 줄 번호

```
int main()
{
    printf( "line test before #line \n line : %d, file(%s)\n", __LINE__, __FILE__ );
    #line 19 "test.cpp"
    printf( "line test after #line \n line : %d, file(%s)\n", __LINE__, __FILE__ );
    return 0;
}
```



__FILE__ - 현재 소스파일의
경로 및 이름



```
line test before #line
line : 8, file(c:\users\hong\documents\code\test\test.cpp)
line test after #line
line : 19, file(c:\users\hong\documents\code\test\test.cpp)
계속하려면 아무 키나 누르십시오 . . .
```

8. #import

- 형식 라이브러리 정보를 통합하는데 사용한다.
- #import 구문

```
#import "filename" [attributes]
#import <filename> [attributes]
```

1. 형식 라이브러리를 포함하는 파일

2. `PreID(COMMActiveX 개체형 실행할 클래스 식별자)`

3. `lib` (형식 정보를 사용할 때는 심표를 이용해 구분한다.)

4. 실행파일(.exe)

5. 라이브러리 리소스를 포함하는 라이브러리 파일(.dll)

6. 형식 라이브러리를 보유하는 복함문서

9. #using

- /clr을 사용하여 컴파일된 프로그램의 메타 데이터를 가져 올 때 사용한다.

- #using구문

#using file [as_friend]



*.dll, *.exe, *.obj 파일

10. #pragma

- 컴파일러의 기능을 지정할 때 사용한다.
- 컴파일러가 모르는 pragma를 사용하면 경고를 표시하고, 컴파일은 계속 수행한다.
- #pragma 구문
`#pragma token-string`
`__pragma(token-string)`

10. #pragma

alloc_text	auto_inline	bss_seg	check_stack	Code_seg
Comment	Component	Fenv_access	Float_control	Fp_contract
Conform	Const_seg	Data_seg	Deprecated	Detect_mismatch
Function	Hdrstop	Include_alias	Init_seg	Inline_depth
Inline_recursion	Intrinsic	Loop	make_public	Managed
Message	#pragma token_string Omp_pragma(token_string)		Optimize	Pack
Pointers_to_members	Pop_macro	Push_macro	Region	Endregion
Runtime_checks	Section	Setlocale	Strict_gs_check	Unmanaged
Vtordisp	warning			

감사합니다.