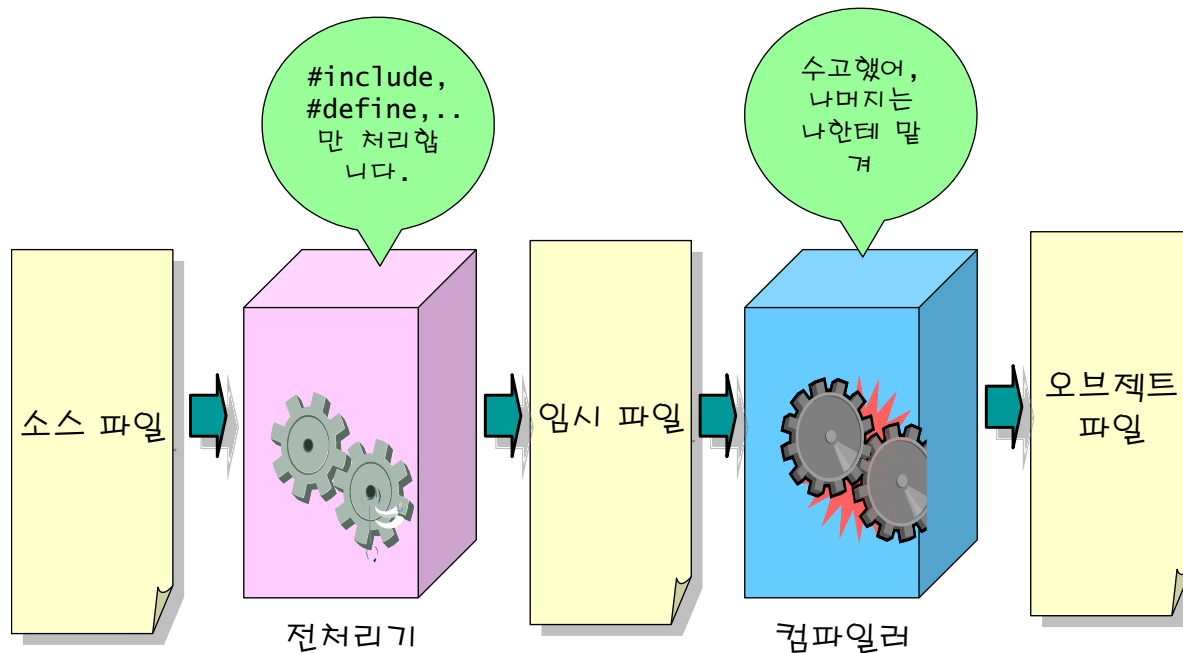


# 전처리기 (Preprocessor)

# 전처리기(Preprocessor)

- 본 작업을 시작하기 전에 정상적 작업 수행을 위해 선행 처리 되어야 할 작업을 이행(하는 모듈)
- 컴파일하기에 앞서서 소스 파일을 처리하는 컴파일러의 한 부분



```
#include <stdio.h>
#define MAX 100

int main(void) {
    int k;
    k = k+ MAX;
    printf("합은 %d입니다\n");
}
```

전처리기



```
int main(void) {
    int k;
    k = k+ 100;
    printf("합은 %d입니다\n");
}
void printf(...) {
    ...
}
```

# 전처리기의 요약

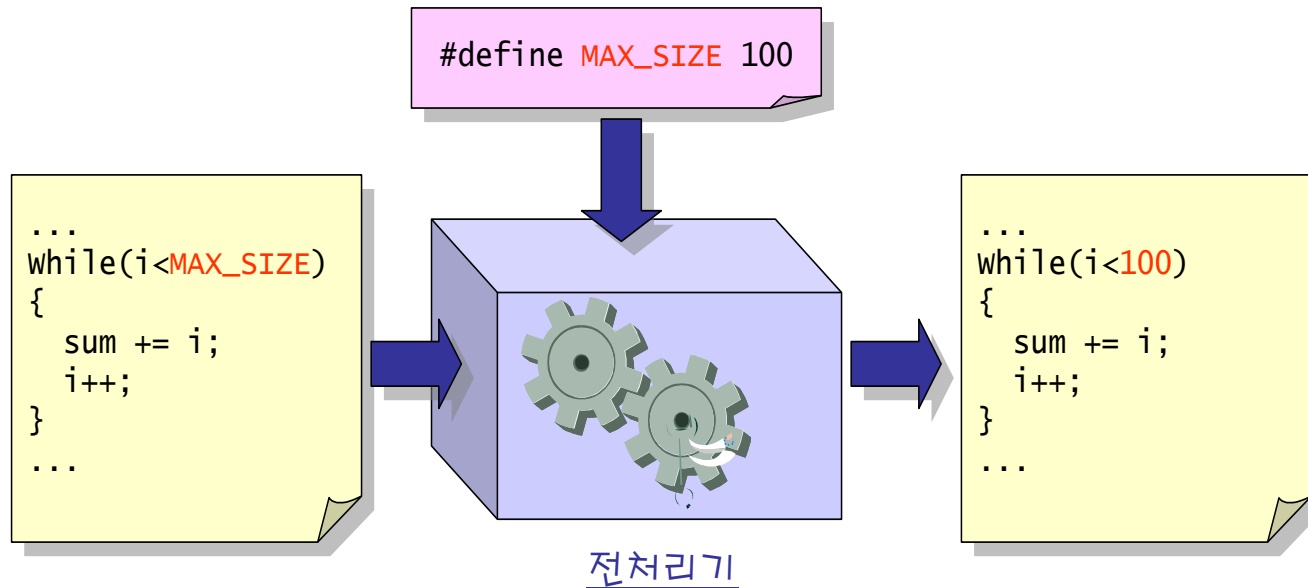
지시어	의미
#define	매크로 정의
#include	파일 포함
#undef	매크로 정의 해제
#if	조건이 참일 경우
#else	조건이 거짓일 경우
#endif	조건 처리 문장 종료
#ifdef	매크로가 정의되어 있는 경우
#ifndef	매크로가 정의되어 있지 않은 경우
#line	행번호 출력
#pragma	시스템에 따라 의미가 다름

# 단순 매크로

- 단순 매크로(macro): 숫자 상수 및 텍스트를 기호 상수로 만든 것

#define	매크로	치환텍스트
---------	-----	-------

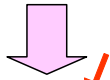
- (예) #define MAX\_SIZE 100



# 단순 매크로의 장점

- 프로그램의 가독성(readability)을 높인다.
- 상수의 변경이 용이하다.

```
#define MAX_SIZE 100
for(i=0;i<MAX_SIZE;i++)
{
    f += (float) i/MAX_SIZE;
}
```



```
#define MAX_SIZE 1000
for(i=0;i<MAX_SIZE;i++)
{
    f += (float) i/MAX_SIZE;
}
```

기호 상수를 사용하는 경우

```
for(i=0;i<100;i++)
{
    f += (float) i/100;
}
```



```
for(i=0;i<1000;i++)
{
    f += (float) i/1000;
}
```

숫자를 사용하는 경우

#define	PRINT	printf	
#define	PI	3.141592	// 원주율
#define	TWOPI	(3.141592 * 2.0)	// 원주율의 2배
#define	MAX_INT	2147483647	// 최대 정수
#define	EOF	(-1)	// 파일의 끝 표시
#define	MAX_STUDENTS	2000	// 최대 학생 수
#define	EPS	1.0e-9	// 실수의 계산 한계
#define	DIGITS	"0123456789"	// 문자 상수 정의
#define	BRACKET	"(){}[]"	// 문자 상수 정의
#define	getchar()	getc(stdin)	// stdio.h에 정의
#define	putchar()	putc(stdout)	// stdio.h에 정의



# 예제 #1

```
#include <stdio.h>
#define AND      &&
#define OR      ||
#define NOT      !
#define IS       ==
#define ISNOT    !=

int search(int list[], int n, int key)
{
    int i = 0;

    while( i < n AND list[i] != key )
        i++;
    if( i IS n )
        return -1;
    else
        return i;
}

int main(void)
{
    int m[] = { 1, 2, 3, 4, 5, 6, 7 };

    printf("%d\n", search(m, sizeof(m)/sizeof(m[0]), 5));
    return 0;
}
```



C프로그래밍  
을 다른  
언어처럼  
작성할 수  
있습니다.

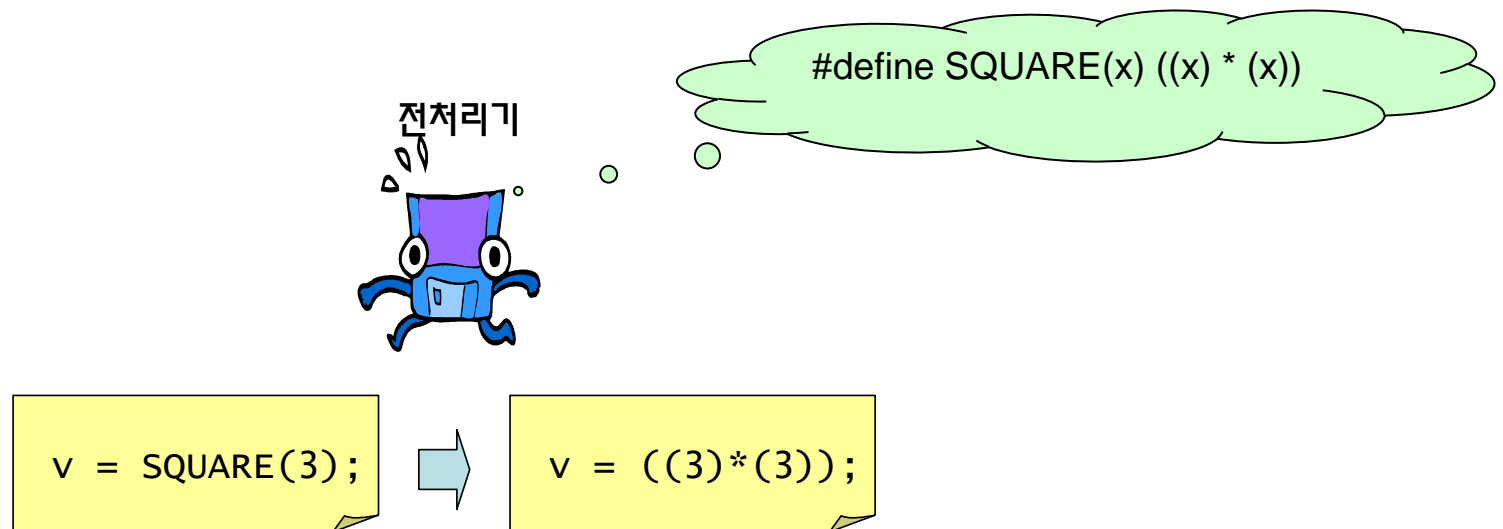


# 함수 매크로

- *함수 매크로(function-like macro)*란 매크로가 함수처럼 매개 변수를 가지는 것

**#define      매크로(매개변수1, 매개변수2,...)      치환텍스트**

- (예) #define SQUARE(x) ((x) \* (x))



```
#define SUM(x, y)          ((x) + (y))
#define AVERAGE(x, y, z) (( (x) + (y) + (z) ) / 3 )
#define MAX(x,y)          ( (x) > (y) ) ? (x) : (y)
#define MIN(x,y)          ( (x) < (y) ) ? (x) : (y)
```

```
#define SQUARE(x) x*x      // 위험 !!
```

```
v = SQUARE(a+b);
```



```
v = a + b*a + b;
```



함수  
매크로에서는  
매개 변수를  
괄호로  
둘러싸는 것이  
좋습니다.

# 예제 #1

```
1. // 매크로 예제
2. #include <stdio.h>
3. #define SQUARE(x) ((x) * (x))
4.
5. int main(void)
6. {
7.     int x = 2;
8.
9.     printf("%d\n", SQUARE(x));
10.    printf("%d\n", SQUARE(3));
11.    printf("%f\n", SQUARE(1.2));
12.    printf("%d\n", SQUARE(x+3));
13.    printf("%d\n", 100/SQUARE(x));
14.    printf("%d\n", SQUARE(++x));
15.
16.    return 0;
17. }
```

$((++x) * (++x))$

// 실수에도 적용 가능

// 논리 오류

```
4
9
1.440000
25
25
16
```

# 함수 매크로의 장단점

- 함수 매크로의 장단점
  - 함수 호출 단계가 필요없어 실행 속도가 빠르다.
  - 소스 코드의 길이가 길어진다.
- 간단한 기능은 매크로를 사용
  - `#define MIN(x, y) ((x) < (y) ? (x) : (y))`
  - `#define ABS(x) ((x) > 0 ? (x) : -(x))`

# 내장(built-in) 매크로

## ■ 내장 매크로: 미리 정의된 매크로

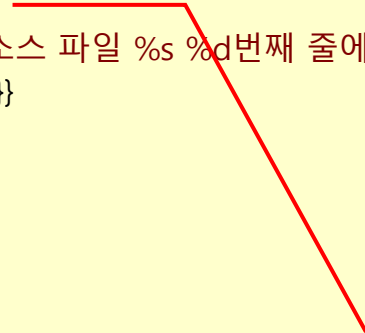
내장 매크로	설명
<code>__DATE__</code>	이 매크로를 만나면 <b>현재의 날짜</b> (월 일 년)로 치환된다.
<code>__TIME__</code>	이 매크로를 만나면 <b>현재의 시간</b> (시:분:초)으로 치환된다.
<code>__LINE__</code>	이 매크로를 만나면 소스 파일에서의 <b>현재의 라인 번호</b> 로 치환된다.
<code>__FILE__</code>	이 매크로를 만나면 <b>소스 파일 이름</b> 으로 치환된다.

📌 `printf("컴파일 날짜=%s\n", __DATE__);`

📌 `printf("치명적 에러 발생 파일 이름=%s 라인 번호= %d\n", __FILE__, __LINE__);`

# 예제: ASSERT 매크로

```
1. #include <stdio.h>
2. #define DEBUG
3. #ifdef DEBUG
4. #define ASSERT(exp) { if (!(exp)) W
5.     { printf("가정(" #exp ")이 소스 파일 %s %d번째 줄에서 실패.\n",
6.       __FILE__, __LINE__, exit(1));}
7. #else
8. #define ASSERT(exp)
9. #endif
10. int main(void)
11. {
12.     int sum;                // 지역 변수의 초기값은 0이 아님
13.     ASSERT(sum == 0);       // sum의 값은 0이 되어야 함.
14.     return 0;
15. }
```



매크로를 다음 줄로 연장  
할 때 사용

가정(sum == 0)이 소스 파일 c:\cprogram\test\test.c 17번째 줄에서 실패.

# #ifdef

- 조건부 컴파일을 지시
- 어떤 조건이 만족되었을 경우 해당 문장들을 컴파일

```
#ifdef 매크로
문장1      // 매크로가 정의되었을 경우
#else
문장2      // 매크로가 정의되지 않았을 경우
#endif
```

- (예)

```
#define LINUX    // 매크로 정의
...
...

#ifdef LINUX
...           // 리눅스 버전에 관련된 문장들
...
#else
...           // 윈도우 버전에 관련된 문장들
...
#endif

...           // 공통적으로 필요한 문장들
...
...
...
```

리눅스 버전

윈도우 버전

공통 부분

# #if

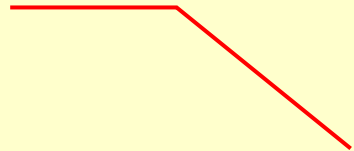
- 기호가 참으로 계산되면 컴파일
- 조건은 상수이어야 하고 논리, 관계 연산자 사용 가능

```
#if 조건  
문장들  
#endif
```

```
#define METHOD 1  
...  
#if METHOD == 1  
printf("방법 1이 선택되었습니다.\n");  
#endif
```

```
#if 조건1  
문장1  
#elif 조건2  
문장2  
#else  
문장3  
#endif
```

else if를 줄인 것





## 다양한 예

```
#if (VERSION > 3)           // 버전이 3 이상이면 컴파일
```

● ● ●

```
#endif
```

```
#if (VERSION > 3.0)           // 오류 !! 버전 번호는 300과 같은 정수로 표시
```

```
#if (AUTHOR == "CHULSOO")           // 오류 !!
```

```
#if (AUTHOR == KIM)           // 가능!! KIM은 다른 매크로
```

```
#if (VERSION*10 > 500 && LEVEL == BASIC) // 가능!!
```

```
#if (VERSION > 300 || defined(DELUXE) )
```

```
#if 0 // 소스의 일부분을 주석 처리하는 방법
```

● ● ●

```
#endif
```

# 조건부 컴파일을 이용하는 디버깅

```
#define DEBUG 1
...
#if DEBUG == 1
printf("현재 counter의 값은 %d입니다.\n", counter);
#endif

#define DEBUG
...
#ifdef DEBUG
printf("현재 counter의 값은 %d입니다.\n", counter);
#endif

...
#if defined(DEBUG)
printf("현재 counter의 값은 %d입니다.\n", counter);
#endif
```

# 헤더 파일 이중 포함 방지

prog1.h가 common.h를 include하고 있을 경우

- prog1.c 에서 prog1.h와 common.h를 include 한다면
- common.h가 이중 include된다.

```
/**
```

```
*stdio.h - definitions/declarations for standard I/O routines
```

```
****/
```

```
#ifndef _INC_STDIO
```

```
#define _INC_STDIO
```

```
....
```

```
....
```

```
#endif
```



헤더 파일이  
포함되면  
매크로가  
정의되어서  
이중 포함을  
방지합니다.

# #undef, #pragma

## ■ #undef: 매크로의 정의를 취소

```
1. #include <stdio.h>
2. #define DEBUG

3. int main(void)
4. {
5.     #ifdef DEBUG
6.         printf("DEBUG이 정의되었습니다.\n");
7.     #endif

8.     #undef DEBUG                                // DEBUG 매크로의 정의를 취소

9.     #ifdef DEBUG
10.        printf("DEBUG이 정의되었습니다.\n");      // 컴파일되지 않는다.
11.    #endif
12.        return 0;
13. }
```

# #include

- 지정된 파일을 읽어서 그 위치에 삽입
  - `#include <.....>` 표준 디렉토리에서 파일을 찾는다.
  - `#include "....."` 현재 디렉토리에서 파일을 찾는다.
- 표준 디렉토리
  - INCLUDE라는 환경 변수가 지정하는 디렉토리이다. 보통은  
“C:\Program Files\Microsoft Visual Studio\VC98\include”
- 하부 디렉토리를 지정할 수 있다.
  - `#include "graphic/point.h"`

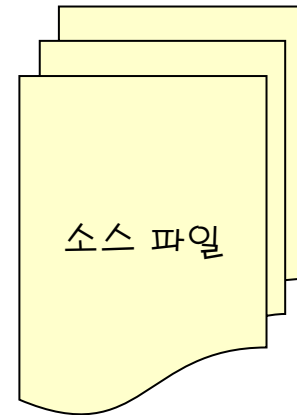
# 다중 소스 파일

## ■ 단일 소스 파일

- 파일의 크기가 너무 커진다.
- 소스 파일을 다시 사용하기가 어려움

## ■ 다중 소스 파일

- 서로 관련된 코드만을 모아서 하나의 소스 파일로 할 수 있음
- 소스 파일을 재사용하기가 간편함



# 예제 #1

## *multiple\_source.c*

```
// 다중 소스 파일
#include <stdio.h>
#include "power.h"

int main(void)
{
    int x,y;

    printf("x의 값을 입력하시오:");
    scanf("%d", &x);
    printf("y의 값을 입력하시오:");
    scanf("%d", &y);
    printf("%d의 %d 제곱값은 %f\n", x, y, power(x, y));

    return 0;
}
```

## *power.h*

```
// power.c에 대한 헤더 파일
double power(int x, int y);
```

## *power.c*

```
// 다중 소스 파일
#include "power.h"

double power(int x, int y)
{
    double result = 1.0;
    int i;

    for(i = 0; i < y; i++)
        result *= x;

    return result;
}
```