프로그래밍 기초

16장. 전처리

학습목표

- **동적 메모리 할당 방식을 이해하고 설명할 수 있다.**
  - 정적 할당과 동적 할당의 필요성
  - 함수 malloc(), calloc(), realloc(), free()의 사용
  - 동적 메모리 할당으로 구현하는 자기참조 구조체

- **연결 리스트를 이해하고 설명할 수 있다.**
  - 연결 리스트의 이해와 구현
  - 연결 리스트의 노드, 헤드, 순회, 삽입, 삭제
  - 여러 파일로 구성하는 프로젝트와 사용자 정의 헤더파일
  - 연결 리스트의 구현

- **전처리 지시자를 이해하고 설명할 수 있다.**
  - 매크로와 인자를 활용한 매크로
  - #if와 #endif
  - #ifdef와 #endif, #ifndef #endif
  - 전처리 연산자 #, #@, ##, defined

다양한 전처리 명령어

- **전처리기가 처리하는 지시자의 종류**
  - #include: 헤더파일을 삽입
  - #define: 기호상수를 정의
- **조건부 컴파일 문장**
  - 조건부로 필요한 문장을 컴파일에 참여시키는 지시자
    - ➢ 디버깅 메시지 출력 또는 개발 단계에서만 필요한 부분 추가 등
    - ➢ 버전과 플랫폼에 따라 다르게 컴파일

명령어	설명
#include	지정된 헤더 파일 내용을 현재 소스 파일에 추가
#define	기호 상수 정의
#undef	지정한 기호 상수 삭제
#if 조건식	주어진 연산식이 참(TRUE)이면 컴파일
#else	조건부 컴파일 블록의 마지막을 표시
#elif 다른_조건식	조건부 컴파일 블록 표시 (#if 조건식 이외의 다른 조건식 비교)
#endif	조건부 컴파일 블록의 종료 표시
#ifdef 매크로명	주어진 매크로명이 정의되었으면 컴파일 수행
#ifndef 매크로명	주어진 매크로명이 정의되지 않았다면 컴파일 수행

예약 매크로 예제

- **예약 매크로(predefined macro)**
  - 시스템에서 이미 정의되어 있는 매크로: ANSI 표준 매크로로 프로그램 디버깅에 활용

매크로	설명
__LINE__	현재 소스 파일에서 이 문장이 있는 행 번호를 표시
__FILE__	현재 소스 파일 이름으로 표시되는 문자열
__DATE__	소스를 컴파일한 날짜를 문자열 “mmm dd yyyy” 형식으로 표시
__TIME__	소스를 컴파일한 시간을 문자열 “hh:mm:ss” 형식으로 표시
__TIMESTAMP__	소스 파일을 마지막으로 수정한 “요일 월 날짜 시:분:초” 형식으로 표시

<07sysmacro.c>

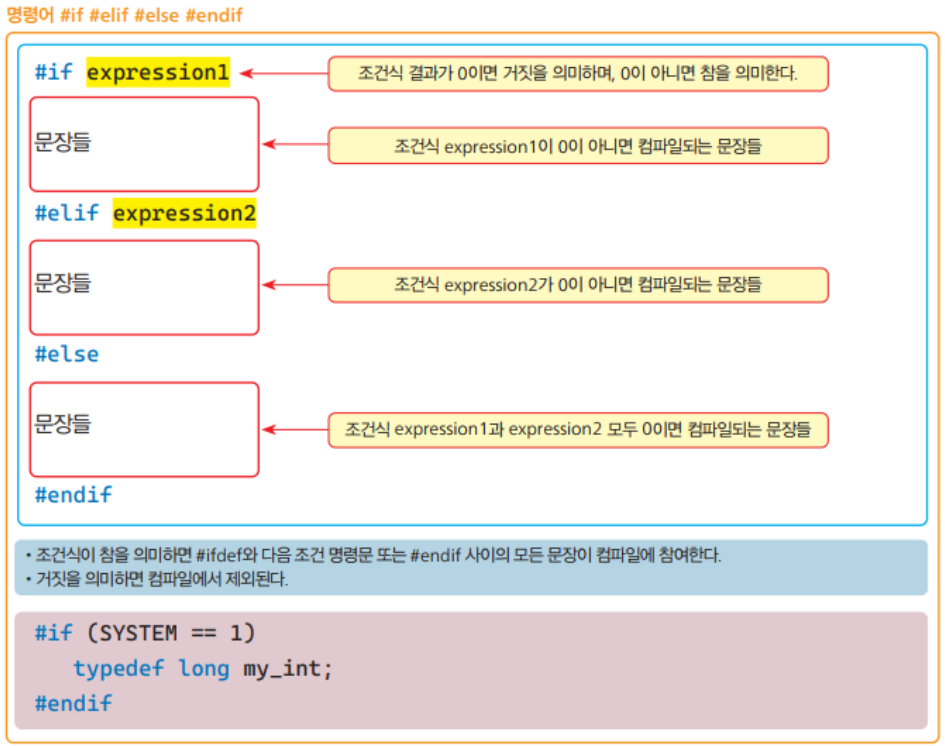
```c
#include <stdio.h>

int main(void)
{
    printf("%s\n", __DATE__);
    printf("%s\n", __FILE__);
    printf("%d\n", __LINE__);
    printf("%s\n", __TIME__);
    printf("%s\n", __TIMESTAMP__);
    return 0;
}
```

실행 결과

```
Dec  9 2025
07sysmacro.c
7
11:48:49
Mon Dec  4 14:07:27 2023
```

기본 조건부 컴파일 명령어 #if #endif

- **기본적인 조건부 컴파일 지시자**
  - 전처리 #if #endif
    - #if 다음의 조건식 expression이 0이 아니면(TRUE)
      - #if 아래의 모든 문장을 컴파일
    - 조건식 expression이 0이면 컴파일에서 제외
    - #if는 반드시 #endif로 종료
  - #if에서는 조건식에 괄호는 생략 가능
    - 괄호를 사용해도 무방
    - 조건 절의 문장이 여러 개라도 블록을 사용할 필요가 없음
  - #if와 #endif 사이
    - #define과 같은 다른 전처리기 지시자도 가능
    - 옵션으로 else if를 의미하는 #elif와 else인 #else 가능



#if 조건식

- **명령문 #if 조건식**
  - 기호 상수와 정수 상수, 문자 상수만 사용 가능
  - 실수와 문자열, 변수 등은 사용 불가능
  - 그 결과도 반드시 정수
  - 조건식에는 관계연산자와 논리연산자 그리고 사칙연산을 사용 가능
    - ➢ 조건식에 변수는 사용 불가능

문자 상수 사용

```
#define LEVEL 'A'

#if LEVEL == 'A'
  . . .
#elif LEVEL == 'B'
  . . .
#else
  . . .
#endif
```

- **잘못된 #if 조건식**

```
#if SYSTEM < 2.0
#define TEST 100
#endif
```

실수 상수 사용

```
#define PL "Java"
#if PL == "Java"
#define TEST 100
#endif
```

문자열 상수 사용

```
int a = 10;
#if a == 10
typedef long my_int;
#endif
```

변수 사용

전처리기 지시자 #if를 이용

<08preif.c>

```c
#include <stdio.h>

//상수 정의
#define WINDOWS 1
#define MAC 2
#define UNIX 3
//#define SYSTEM WINDOWS      // 주석 해제 후 사용 가능

//전처리 #if #elif #else #endif
#if (SYSTEM == WINDOWS)
typedef int my_int;
#elif SYSTEM == MAC
typedef long my_int;
#elif SYSTEM == LINUX
typedef long long my_int;
#else
typedef short my_int;
#endif

int main(void)
{
    my_int n = 17;
    printf("변수크기: %zu, 저장 값: %d\n", sizeof(n), n);

    return 0;
}
```

```
% gcc 08preif.c –DSYSTEM=1 -o 08preif.exe
```

<08preif_01.c>

```c
#include <stdio.h>

// #define WINDOWS

#ifdef WINDOWS
typedef int my_int;
#elif defined MAC
typedef long my_int;
#elif defined LINUX
typedef long long my_int;
#else
typedef short my_int;
#endif

int main(void)
{
    my_int n = 17;
#ifdef WINDOWS
    printf("Windows 변수크기: %zu, 저장 값: %d\n", sizeof(n), n);
#elif defined MAC
    printf("MAC 변수크기: %zu, 저장 값: %ld\n", sizeof(n), n);
#endif
    return 0;
}
```

조건부 컴파일에 사용(-D옵션)
-DWINDOWS
-DMAC
-DLINUX

```
% gcc 08preif_01.c -DWINDOWS -o 08preif_01.exe
```

전처리 연산자 defined

- **전처리 연산자 defined (기호상수)**
  - 기호 상수가 정의되었다면 0이 아닌 값을, 정의가 되지 않았다면 0을 반환
  - 피연산자의 괄호는 생략 가능
  - 연산자 defined는 지시자 #if의 조건식에서 사용 가능

```
#if (defined WINDOWS)
    typedef long my_int;
#endif
```

- **#if defined, #ifdef, #endif**
  - 전처리 #ifdef #endif도 조건부 컴파일 지시자
  - #ifdef 다음에 나오는 기호 상수가 이미 정의되었다면
    - ➢ #endif까지 모든 문장을 컴파일
  - 정의 되어 있지 않으면, 컴파일에서 제거

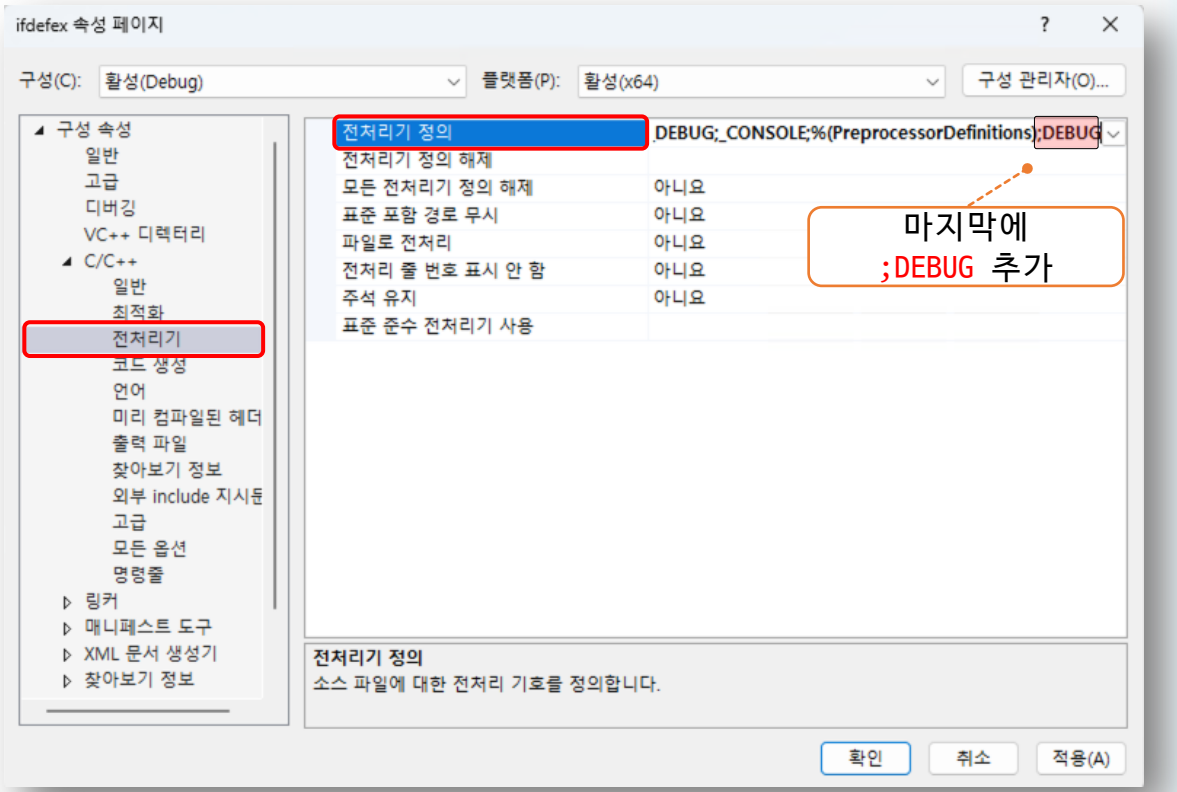
```
#ifdef 기호상수
  . . .

#endif
```

  - #ifdef는 반드시 #endif로 종료
  - #ifdef와 #endif 사이에는 C 문장 뿐 아니라 전처리 지시자도 사용 가능

```
#ifdef DEBUG
    printf("DEBUG : 1부터 %2d까지의 곱은 %d 입니다.\n", i, prod);
#endif
```

기호 상수 정의 방법

- **Visual Studio**
  - 프로젝트 속성 > C/C++ > 전처리기
- **기호상수 DEBUG가 정의되어 있으면**
  - 중간 결과를 출력하는 문장이 컴파일
- **기호상수 DEBUG가 정의되어 있지 않으면**
  - 해당 문장은 컴파일에서 제외
- **기호상수 DEBUG**
  - 지시자 #define으로 정의 가능
  - Visual Studio에서 직접 기호상수를 정의 가능
    - ➢ 프로젝트 속성 > C/C++ > 전처리기 > 전처리기 정의
      - – 세미콜론으로 기호상수 구분
      - – ;DEBUG 추가



전처리 지시자 #ifdef 이용한 디버깅 방법

<09ifdef.c>

- **기호상수 DEBUG가 정의되어 있다면**
  - printf("DEBUG : 1부터 %d까지의 곱은 %d입니다.\n", i, prod); 출력문을 컴파일
- **만일 DEBUG가 정의되어 있지 않다면**
  - #ifdef 내부의 printf()문은 컴파일 되지 않으며 그 부분은 실행되지 않음

```
% gcc 09ifdef.c -DDEBUG -o 09ifdef

% ./09ifdef
DEBUG : 1부터  5까지의 곱은 120 입니다.
DEBUG : 1부터 10까지의 곱은 3628800 입니다.
DEBUG : 1부터 15까지의 곱은 2004310016 입니다.
DEBUG : 1부터 20까지의 곱은 -2102132736 입니다.
1부터 20까지의 곱은 2432902008176640000 입니다.
```

```c
#include <stdio.h>

// #define DEBUG
#define LIMIT 20

int main(void)
{
    long long prod = 1;
    for (int i = 1; i <= LIMIT; i++)
    {
        prod *= i;

#ifdef DEBUG
        if (i % 5 == 0)
            printf("DEBUG : 1부터 %2d까지의 곱은 %d 입니다.\n",
                                    i, prod);
#endif

    }
    printf("1부터 %d까지의 곱은 %ld 입니다.\n", LIMIT, prod);

    return 0;
}
```

_DDEBUG 대신에 소스 코드 상단에 #define DEBUG로 정의할 수도 있음

명령어 #ifndef

- **전처리 지시자 #ifndef**
  - #ifdef와 반대로 기호 상수가 정의되지 않았다면
    - ➢ #ifndef와 #endif 사의의 모든 문장이 컴파일에 참여
  - 정의되었다면: 컴파일에서 제외
- **기호 상수 NAME_SIZE가 정의되어 있지 않다면**
  - NAME_SIZE를 30으로 정의하는 문장
  - 실제로 헤더 파일을 여러 개 사용하다 보면 기호 상수 정의가 중복될 수 있는데
    - ➢ 컴파일 시 경고가 발생
  - #ifndef 를 사용한 기호 상수 정의
    - ➢ 단순히 #define NAME_SIZE 30이 아니라, 헤더파일에서 기호 상수를 중복 정의하는 것을 막아주는 역할

```
#ifndef NAME_SIZE
    #define NAME_SIZE 30
#endif
```

기호상수 삭제 #undef

- #undef는 이미 정의된 기호 상수를 해지하는 지시자

- 다음 구문
  - 20으로 정의된 기호상수 SIZE
    - ➢ 전처리기 지시자 #undef SIZE로 그 효력을 상실하게 함

  - 일반적으로 기호 상수를 삭제하기 전
    - ➢ #ifdef 기호상수 문장으로 이전에 정의됨을 확인한 후 삭제하는 것을 추천

```
#define SIZE 20

#ifdef SIZE
    #undef SIZE
#endif
```

전처리 연산자 종류

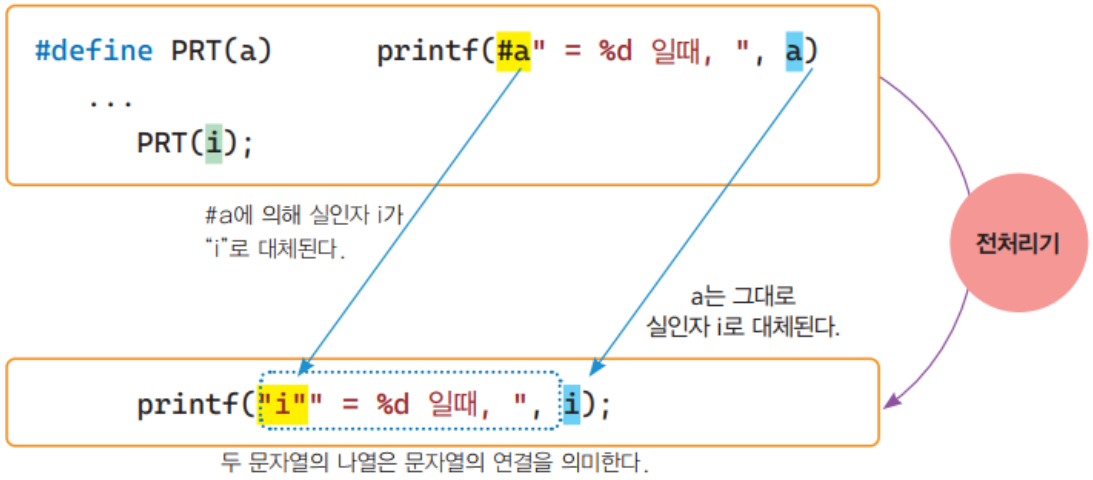
- **전처리 연산자**
  - #, #@, ##, defined 모두 4개
- **연산자 #, #@, ##**
  - 매크로 정의 지시자 #define에서 사용
- **연산자 defined**
  - 이미 보았듯이 조건부 컴파일 지시자 #if와 #elif 등에서 사용

연산자	이름	사용 예	기능
#	문자열 만들기 연산자	#인자	인자 앞 뒤에 큰 따옴표를 붙여 인자를 문자열로 만드는 연산자
#@	문자 만들기 연산자	#@인자	인자 앞 뒤에 작은 따옴표를 붙여 인자를 문자로 만드는 연산자
##	토큰 붙이기 연산자	인자##인자	인자를 다른 토큰들과 연결해주는 연산자
defined	정의 검사 연산자	defined DEBUG	상수로 정의되어 있는지 검사하는 연산자

문자열 만들기 연산자 #(1)

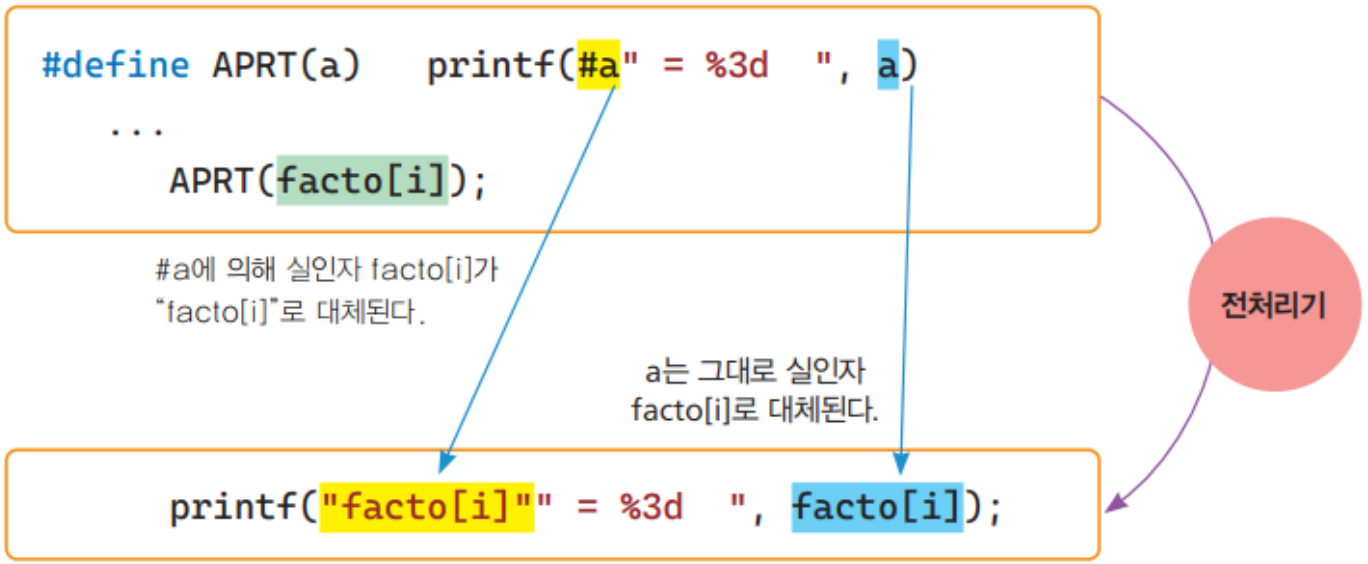
- **문자열 만들기 연산자 #**
  - 매크로 정의 시에 #뒤에 나오는 인자를 문자열로 만듦
    - ➢ 매크로에서 인자의 앞 뒤에 인용 부호 "인자"를 넣어 문자열로 만듬
  - 매크로 PRT(a) 정의
    - ➢ 인자 a의 사용을 #a로 하는 경우
      - – 이 매크로 호출 시에 인자의 앞 뒤에 인용 부호 "a"를 넣어 문자열로 만듬

- **매크로 PRT(i)의 호출**
  - 실인자 #i는 문자열 "i"로 대체되며 실인자 i는 그대로 i로 대체
  - 문자열 "i" 와 " = %2d 일때," 연결
    - ➢ 두 문자열이 연결된 문자열 "i = %2d 일때," 생성



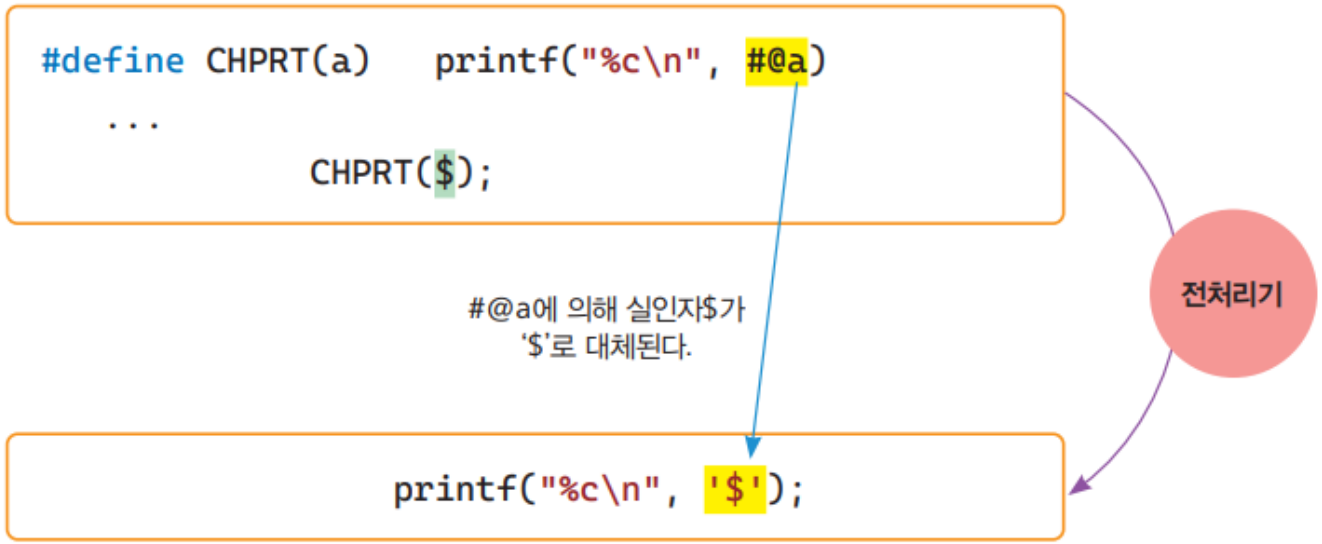
문자열 만들기 연산자 #(2)

- **매크로 정의 APRT(a)**
  - APRT(fact[i])의 호출은 어떤 문장이 실행될까?
  - printf("facto[i]"" = %3d\t", facto[i]); 문장으로 실행



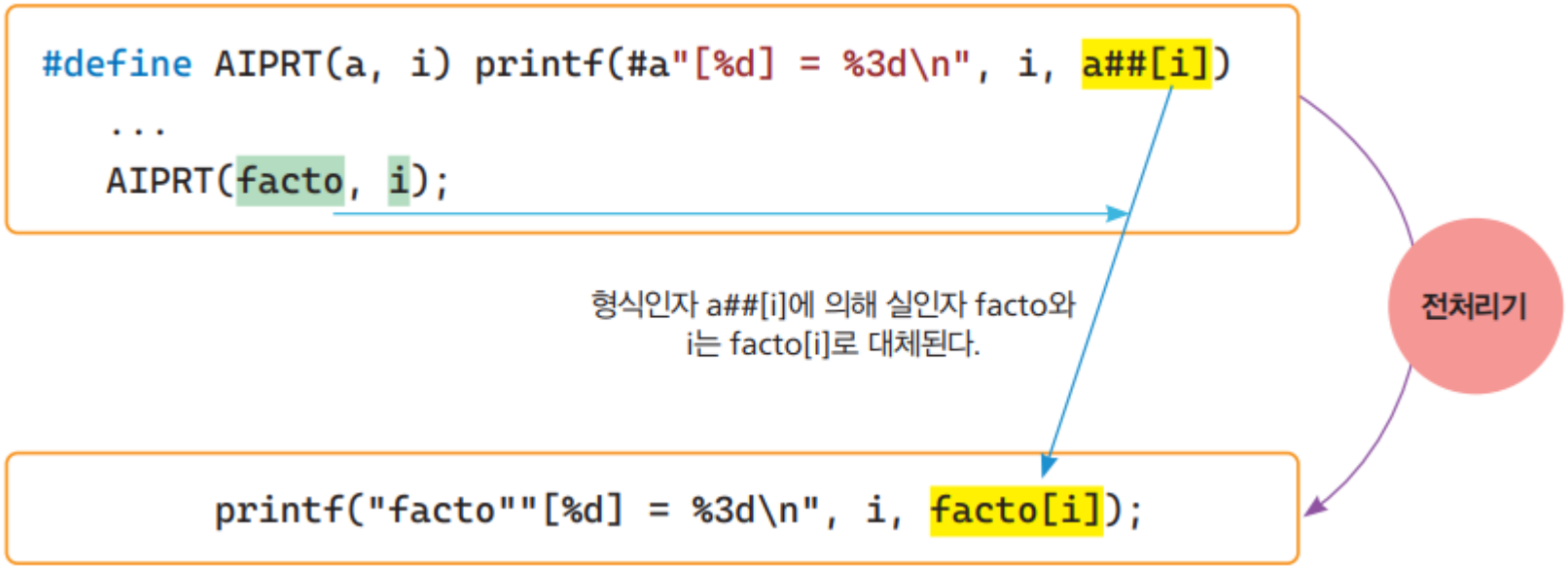
문자 만들기 연산자 #@

- **문자 만들기 연산자 #@ 인자 x의 매크로 정의**
  - #@a와 같이 형식 인자 앞에 위치
    - 연산자 #@는 뒤에 나오는 인자를 앞 뒤에 작은 따옴표를 붙여 문자로 만들어 줌: 'a'
  - 다음 매크로 CHPRT(a)는 인자 a를 문자로 만들어 함수 printf()로 출력하는 매크로
    - 매크로 호출 CHPRT($)로 문자 $를 출력



토큰 붙이기 연산자

- **## 연산자: 좌우의 토큰을 연결(concatenation)하는 기능을 수행**
  - 매크로 AIPRT(a, i)에서 연산식 a##[i]는 a와 [i]를 연결
  - 매크로 호출 AIPRT(facto, i)
    - ➢ 실인자 연산식 facto##[i]에 의해 facto와 i를 연결한 facto[i]로 대체



전처리 연산자 예제

<10operator.c>

```c
#include <stdio.h>

#define CHPRT(a) printf("%c\n", #@a)
#define PRT(a) printf(#a " = %d 일때, ", a)
#define APRT(a) printf(#a " = %3d ", a)
#define AIPRT(a, i) printf(#a "[%d] = %3d\n", i, a##[i])

int main(void)
{
    int prod = 1;
    int facto[6];
    CHPRT($); // 3행 매크로 호출

    for (int i = 1; i <= 5; i++)
    {
        prod *= i;
        facto[i] = prod;
        PRT(i); // 4행 매크로 호출
        APRT(facto[i]); // 5행 매크로 호출
        AIPRT(facto, i); // 6행 매크로 호출
    }

    return 0;
}
```

Visual Studio에서만 실행됨

```
$
i = 1 일때, facto[i] =   1  facto[1] =   1
i = 2 일때, facto[i] =   2  facto[2] =   2
i = 3 일때, facto[i] =   6  facto[3] =   6
i = 4 일때, facto[i] =  24  facto[4] =  24
i = 5 일때, facto[i] = 120  facto[5] = 120
```

Lab. 다양한 자료형의 변수 내용을 서로 교환하는 매크로

- **스왑(swap)**
  - 두 변수의 내용을 서로 교환하는 수행
  - 매크로 SWAP_INT(a, b, temp)
    - ➢ int형 변수 두 a, b를 이미 선언되어 있는 변수 temp를 사용하여 두 변수를 교환하는 매크로로 구현
  - 매크로 SWAP_DOUBLE(a, b)
    - ➢ double형 변수 두 a, b를 매크로 내부에서 직접 변수 _temp를 선언해 두 변수를 교환하는 매크로로 구현
  - 매크로 SWAP_TYPE(type, a, b)
    - ➢ 자료형 type 형인 변수 두 a, b를 매크로 내부에서 직접 자료형 type으로 변수 _swap_temp를 선언해 두 변수를 교환하는 매크로로 구현
- **교환**
  - 정수를 교환하는 매크로와 실수를 교환하는 매크로 각각 2번 호출
- **결과**

```
50 70
70 50
50 70
3.36 7.18
7.18 3.36
3.36 7.18
```

Lab 16-3. 다양한 자료형의 변수 내용을 서로 교환하는 매크로

<lab3swaptype.c>

```c
#include <stdio.h>

//자료형 int a, b의 교환 매크로, temp를 사용
#define SWAP_INT(a, b, temp) \
        temp = a; a = b; b = temp;

#define SWAP_DOUBLE(a, b) \
        double _temp = a; a = b; b = _temp;

//자료형을 지정하여 a, b의 교환 매크로, _swap_temp를 내부에서 지정해서 사용
#define SWAP_TYPE(type, a, b) \
        { \
            type _swap_temp; \
            _swap_temp = (b); \
            (b) = (a); \
            (a) = _swap_temp; \
        }
```

```c
int main(void)
{
    int a = 50, b = 70, c;
    printf("%d %d\n", a, b);
    SWAP_INT(a, b, c);

    printf("%d %d\n", a, b);
    SWAP_TYPE(int, a, b);
    printf("%d %d\n", a, b);

    double x = 3.36, y = 7.18;
    printf("%.2f %.2f\n", x, y);
    SWAP_DOUBLE(x, y);

    printf("%.2f %.2f\n", x, y);
    SWAP_TYPE(double, x, y);
    printf("%.2f %.2f\n", x, y);

    return 0;
}
```

Questions?