WEEK 14

파일 처리와 동적 메모리

학습목표

I. 다양한 파일 관련 함수 이해
II. 동적 메모리와 연결 리스트 이해
III. 전처리 지시자 이해

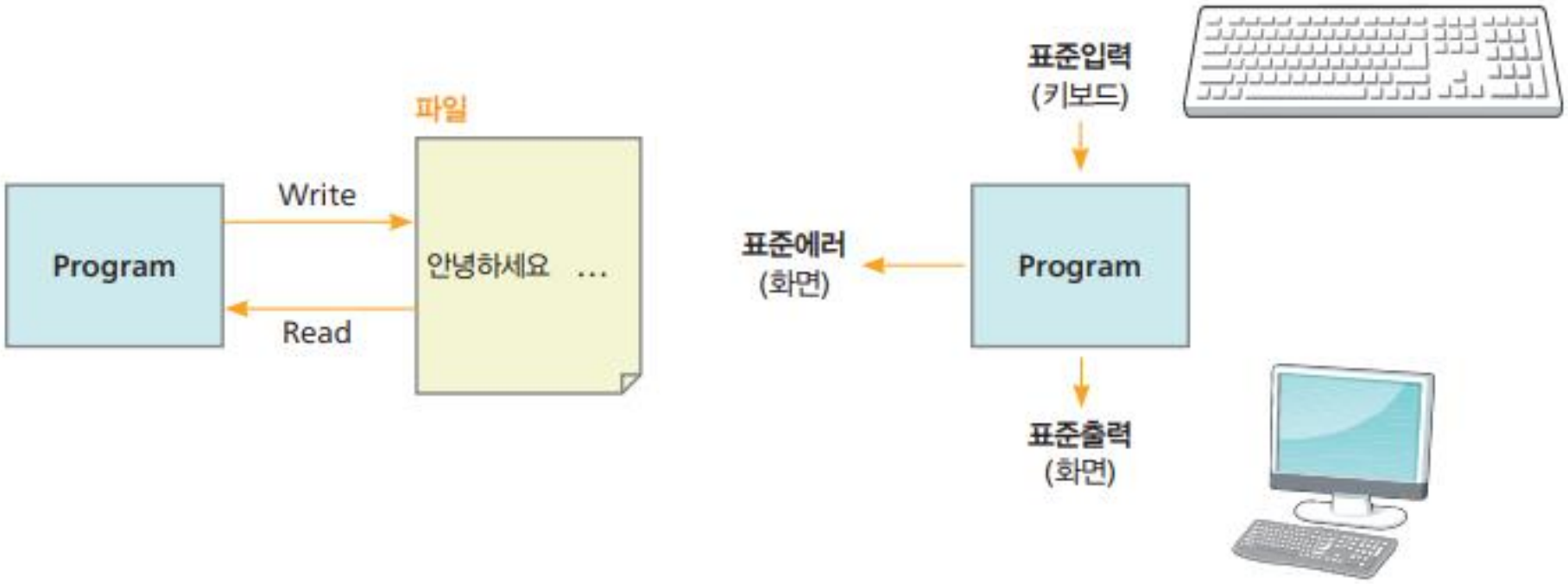
학습목차

I. 제 1교시 파일 처리
II. 제 2교시 동적 메모리와 연결 리스트
III. 제 3교시 전처리

Ⅰ. 파일 처리

1. 텍스트 파일 입출력
2. 이진 파일 입출력
3. 파일 접근 처리

1. 텍스트 파일 입출력

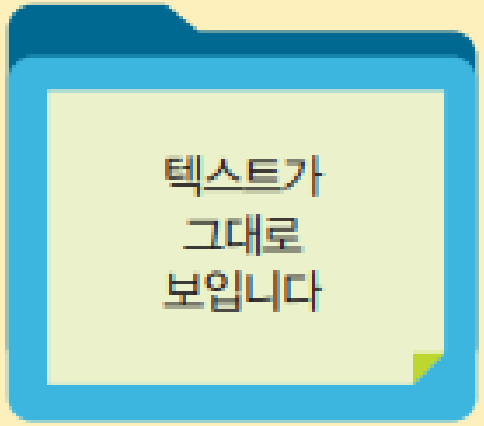


[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 텍스트 파일 입출력

텍스트 파일

이 텍스트 파일은 메모장과 같은 텍스트 편집기를 사용해 그 내용을 볼 수 있으며 필요하면 편집도 할 수 있다.



이진 파일

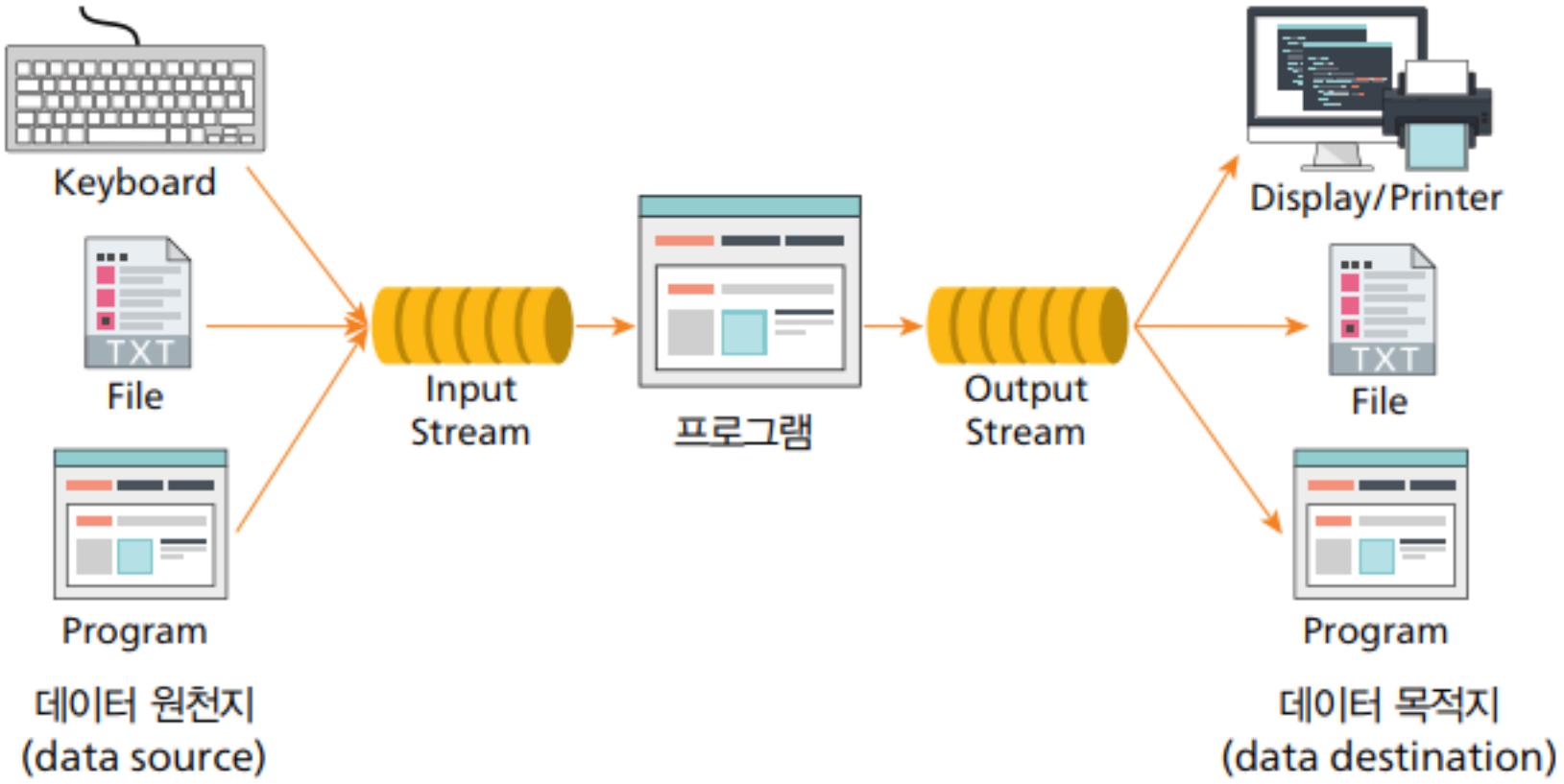
구조체의 변수 등 주기억장치의 내용을 그대로 저장

이미지 파일, 동영상 파일, 실행 파일과 같이 데이터로 구성된 파일로 일반 에디터로는 그 내용을 볼 수 없다.

[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 텍스트 파일 입출력

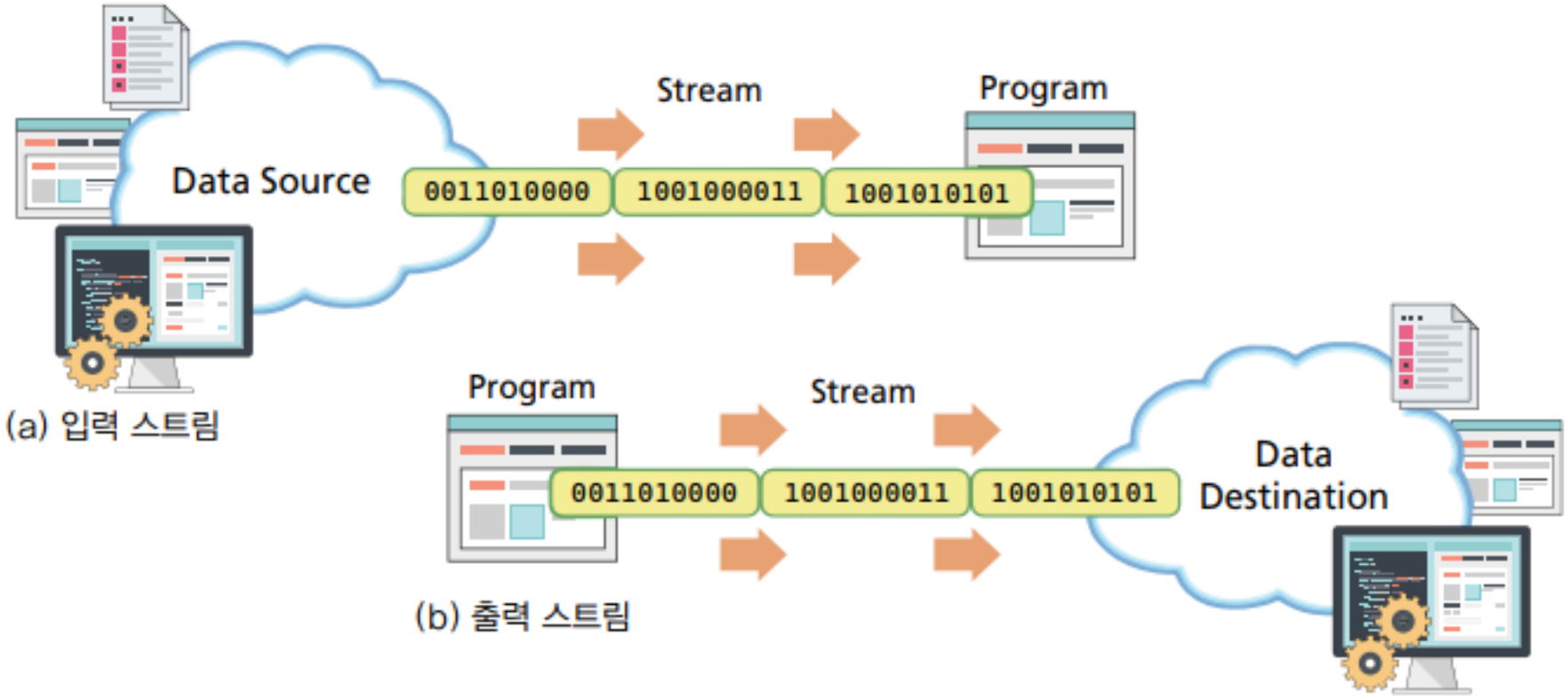
◆ 스트림의 이해



[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 텍스트 파일 입출력

◆ 스트림의 이해

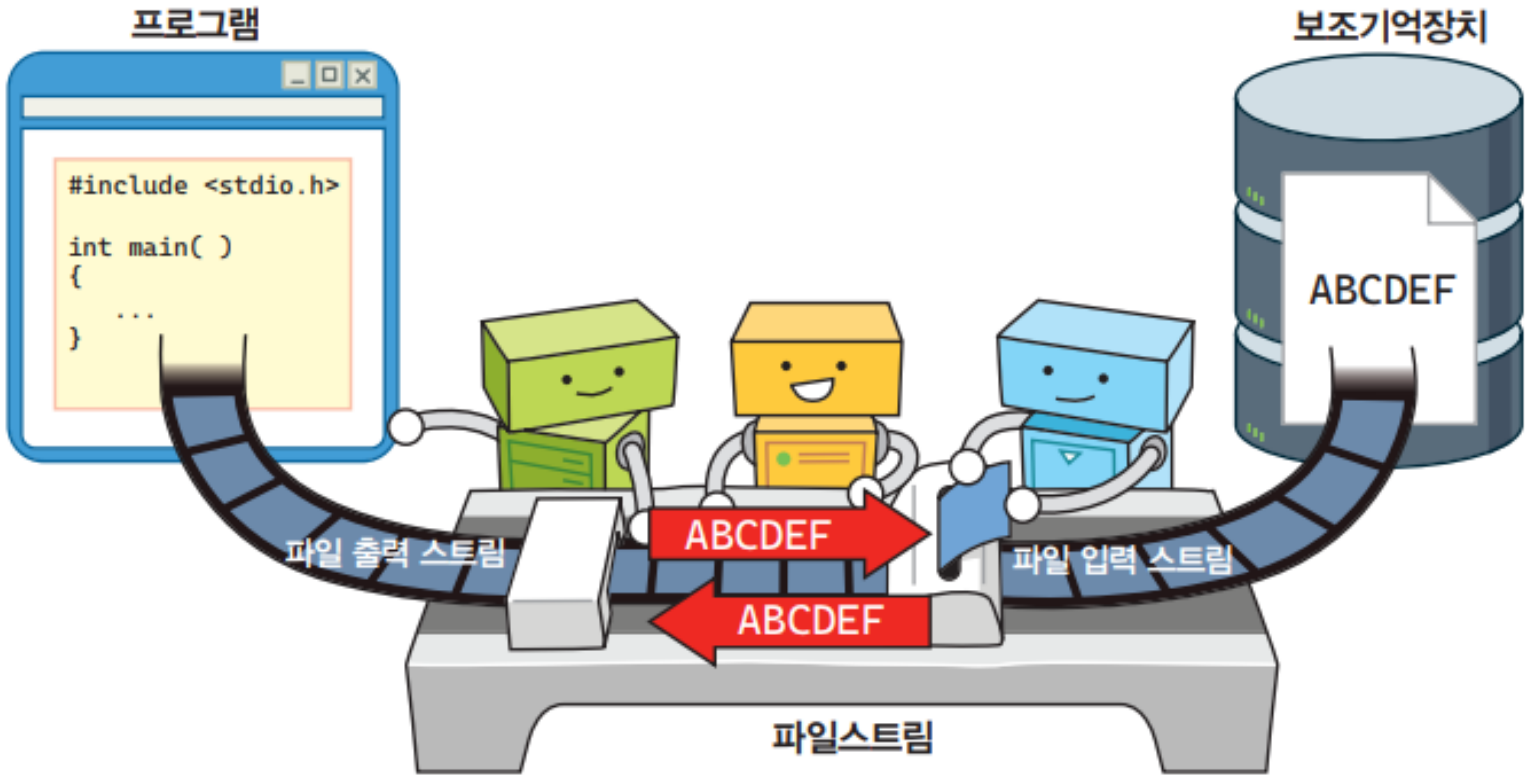


[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 텍스트 파일 입출력

◆ 파일 스트림

❖ 보조기억장치의 파일과 프로그램을 연결하는 전송경로



[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 텍스트 파일 입출력

함수 fopen()과 fopen_s() 함수원형

```
FILE * fopen(const char * _Filename, const char * _Mode);
errno_t fopen_s(FILE ** _File, const char * _Filename, const char * _Mode);
```

- 함수 fopen()은 파일명 _Filename의 파일 스트림을 모드 _Mode로 연결하는 함수이며, 스트림 연결에 성공하면 파일 포인터를 반환하며, 실패하면 NULL을 반환한다.
- 함수 fopen_s()는 스트림 연결에 성공하면 첫 번째 인자인 _ File에 파일 포인터가 저장되고 정수 0을 반환하며, 실패하면 양수를 반환한다. 현재 Visual C++에서는 함수 fopen_s()의 사용을 권장하고 있다.

```
FILE* f; //파일 포인터
char* fname = "basic.txt"; //파일이름

if ((f = fopen(fname, "w")) == NULL)
{
   printf("파일이 열리지 않습니다.\n");
   exit(1);
};
```

```
if (fopen_s(&f, "basic.txt", "w") != 0)
//if ( (f = fopen(fname, "w")) == NULL )
{
   printf("파일이 열리지 않습니다.\n");
   exit(1);
};
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 텍스트 파일 입출력

함수 fclose()

```
int fclose(FILE * _File);
```

함수 fclose()는 파일 스트림 f를 닫는 함수로서, 성공하면 0을 실패하면 EOF을 반환한다.

```
fclose(f);
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 텍스트 파일 입출력

함수 fprintf()와 fscanf() 함수원형

```
int fprintf(FILE * _File, const char * _Format, ...);
int fscanf(FILE * _File, const char * _Format, ...);
int fscanf_s(FILE * _File, const char * _Format, ...);
```

위 함수에서 _File은 서식화된 입출력 스트림의 목적지인 파일이며, _Format은 입출력 제어 문자열이며, 이후 기술되는 인자는 여러 개의 출력될 변수 또는 상수이다.

표준파일	키워드	장치(device)
표준입력	stdin	키보드
표준출력	stdout	모니터 화면
표준에러	stderr	모니터 화면

[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 텍스트 파일 입출력

```
scanf_s("%s%d%d", name, 30, &point1, &point2);

fprintf(f, "%d %s %d %d\n", ++cnt, name, point1, point2);
```

문자열이 저장되는 name과 그 크기를 지정해야 한다.

문자열이 저장되는 name과 그 크기를 지정해야 한다.

```
//파일 "grade.txt"에서 읽기
fscanf_s(f, "%d %s %d %d\n", &cnt, name, 30, &point1, &point2);
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 텍스트 파일 입출력

함수 fgets()와 fputs() 함수원형

```
char * fgets(char * _Buf, int _MaxCount, FILE * _File);
int  fputs(char * _Buf, FILE * _File);
```

- 함수 fgets()는 _File로부터 한 행의 문자열을 _MaxCount 수의 _Buf 문자열에 입력 수행
- 함수 fputs()는 _Buf 문자열을 _File에 출력 수행

```
char names[80];
FILE *f;

fgets(names, 80, f);
fputs(names, f);
```

함수 feof()와 ferror() 함수원형

```
int  feof(FILE * _File);
int  ferror(FILE * _File);
```

- 함수 feof()은 _File의 EOF를 검사
- 함수 ferror()는 _File에서 오류발생 유무를 검사

```
while ( !feof(stdin) )
{
   ...
   fgets(names, 80, stdin);    //표준입력
}
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 텍스트 파일 입출력

함수 fgetc()와 fputc() 함수원형

```
int   fgetc(FILE * _File);
int   fputc(int _Ch, FILE * _File);

int   getc(FILE * _File);
int   putc(int _Ch, FILE * _File);
```

- 함수 fgetc()아 getc()는 _File에서 문자 하나를 입력받는 함수
- 함수 fputc()와 putc()문자 _Ch를 파일 _File 에 출력하는 함수

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 이진 파일 입출력

함수 fwrite()와 fread() 함수원형

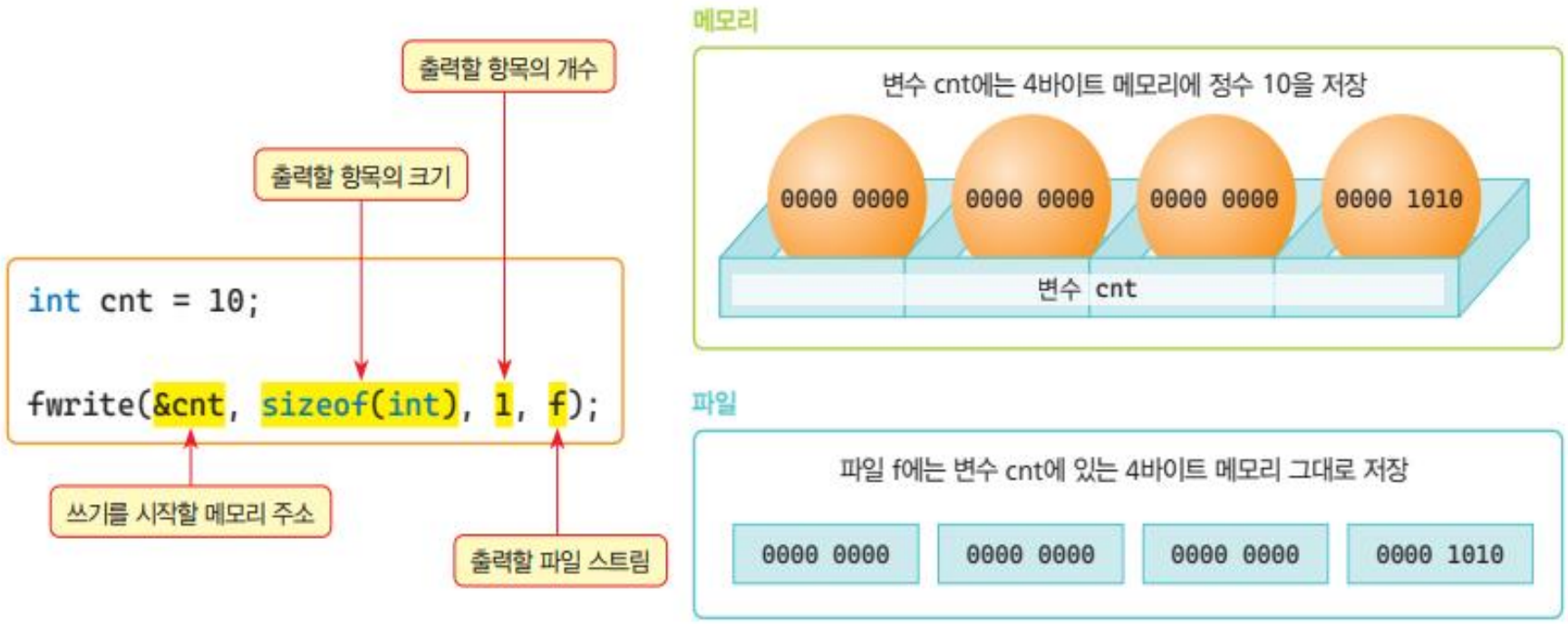
```
size_t fwrite(const void * _Buffer, size_t _ElementSize,
                                    size_t _ElementCount, FILE* _Stream);
size_t fread(void * _Buffer, size_t _ElementSize,
                                    size_t _ElementCount, FILE* _Stream);
```

- 함수 fwrite()는 _Buffer가 가리키는 메모리에서 _ElementSize만큼 _ElementCount개를 파일 _Stream에 쓰는(저장) 함수
- fread()는 반대로 파일 _Stream에서 _ElementSize의 _ElementCount개만큼 메모리 _Buffer에 읽어오는 함수, 반환값은 성공적으로 입출력을 수행한 항목의 수

```
int cnt = 10;
fwrite(&cnt, sizeof(int), 1, f);
fread(&cnt, sizeof(int), 1, f);
```

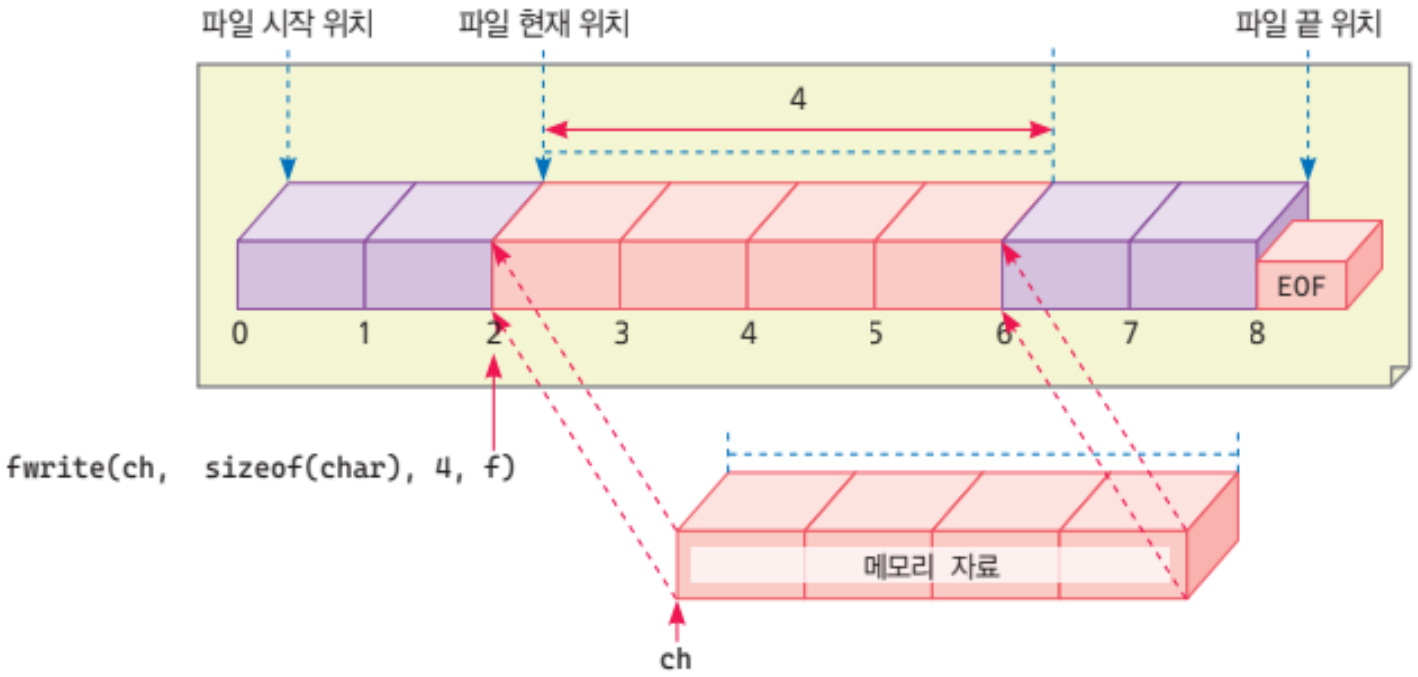
[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 이진 파일 입출력



[출처] 강환수 외, Perfect C 3판, 인피니티북스

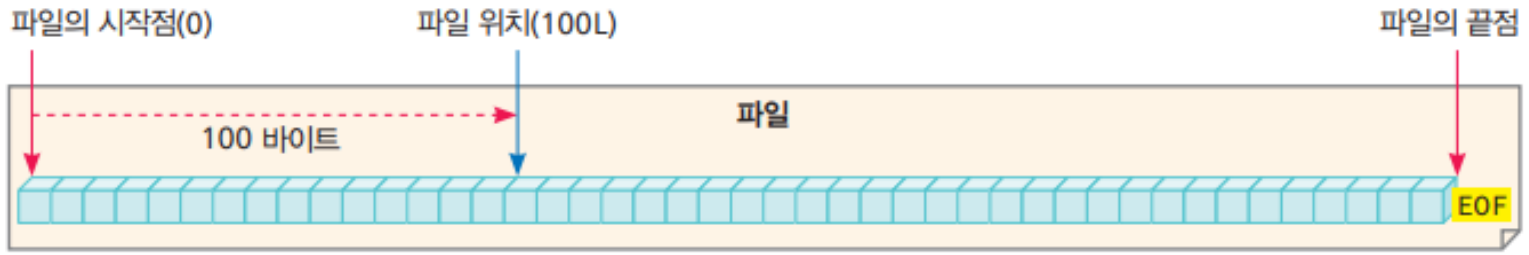
2. 이진 파일 입출력

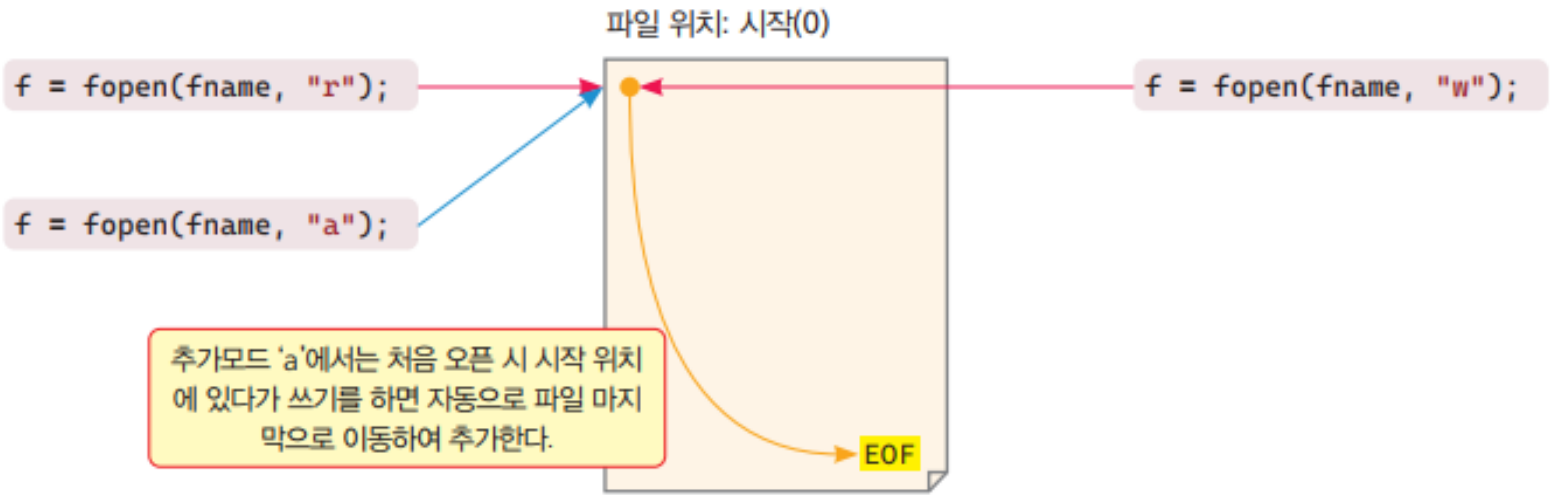


모드	의미
rb	이진 파일의 읽기(read) 모드로 파일을 연다.
wb	이진 파일의 쓰기(write) 모드로 파일을 연다.
ab	이진 파일의 추가(append) 모드로 파일을 연다.

[출처] 강환수 외, Perfect C 3판, 인피니티북스

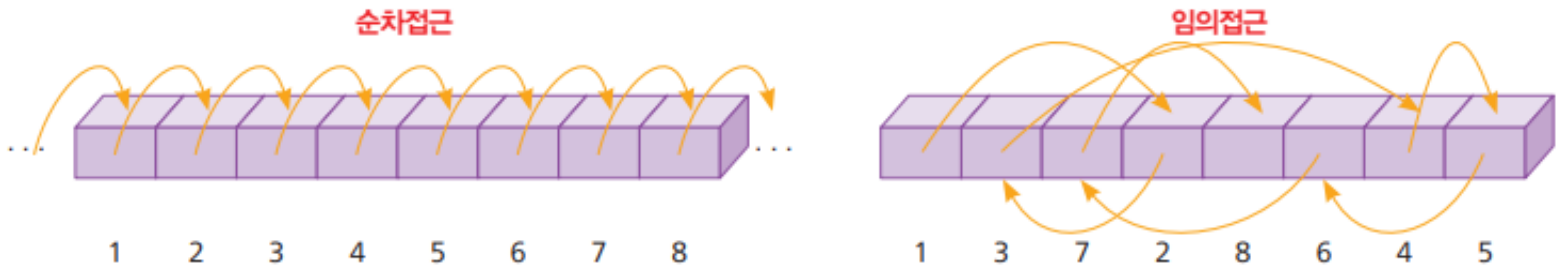
3. 파일 접근 처리





[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 파일 접근 처리



함수 fseek() 함수원형

```
int fseek(FILE * _File, long _Offset, int _Origin);
```

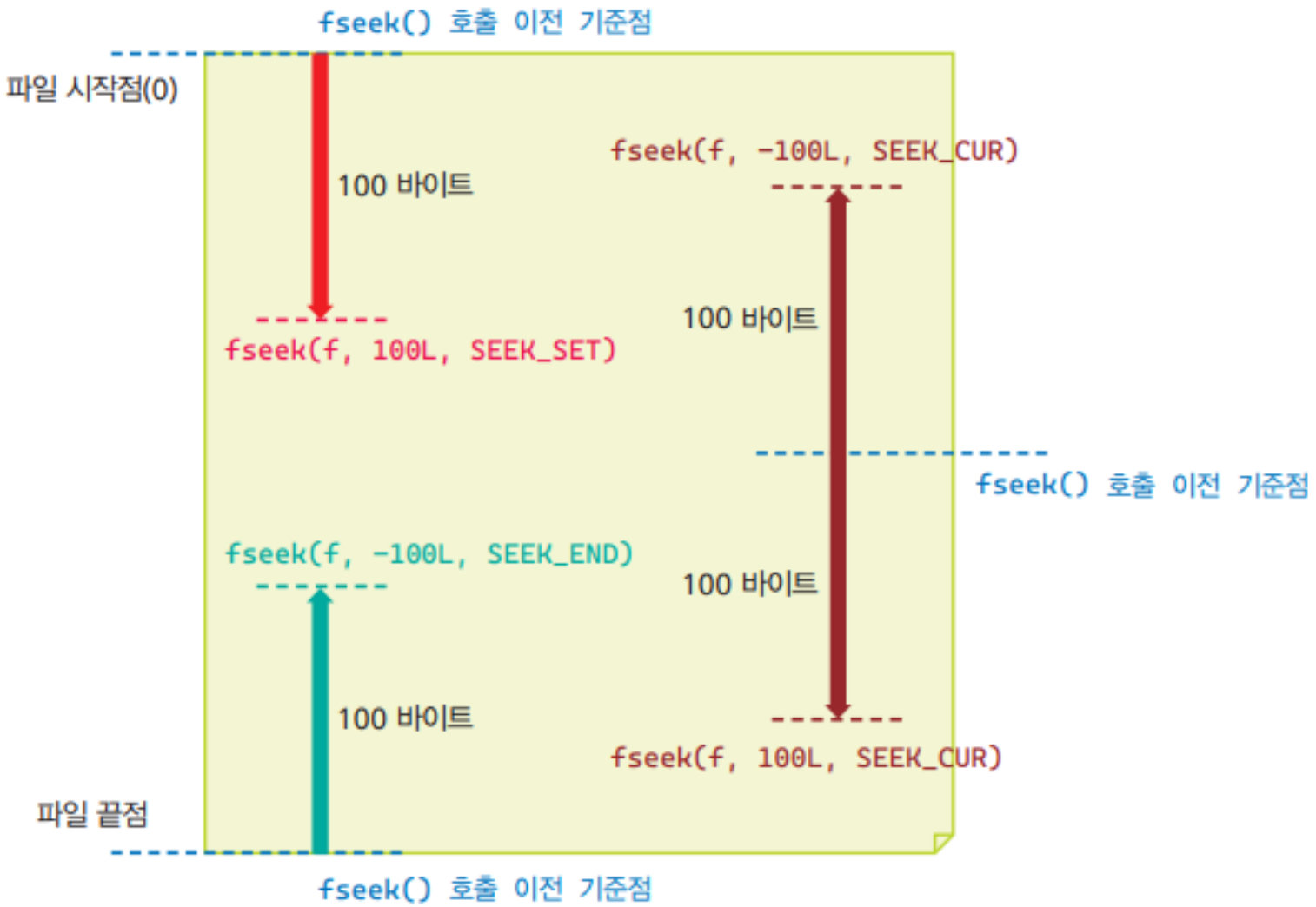
함수 fseek()는 파일 _File의 기준점 _Origin에서 _Offest만큼 파일 포인터를 이동하는 함수 , 성공하면 0을 반환하며 실패하면 0이 아닌 정수를 반환

```
fseek(f, 0L, SEEK_SET);
fseek(f, 100L, SEEK_CUR);
fseek(f, -100L, SEEK_END);
```

기호	값	의미
SEEK_SET	0	파일의 시작 위치
SEEK_CUR	1	파일의 현재 위치
SEEK_END	2	파일의 끝 위치

[출처] 강환수 외, Perfect C 3판, 인피니티북스

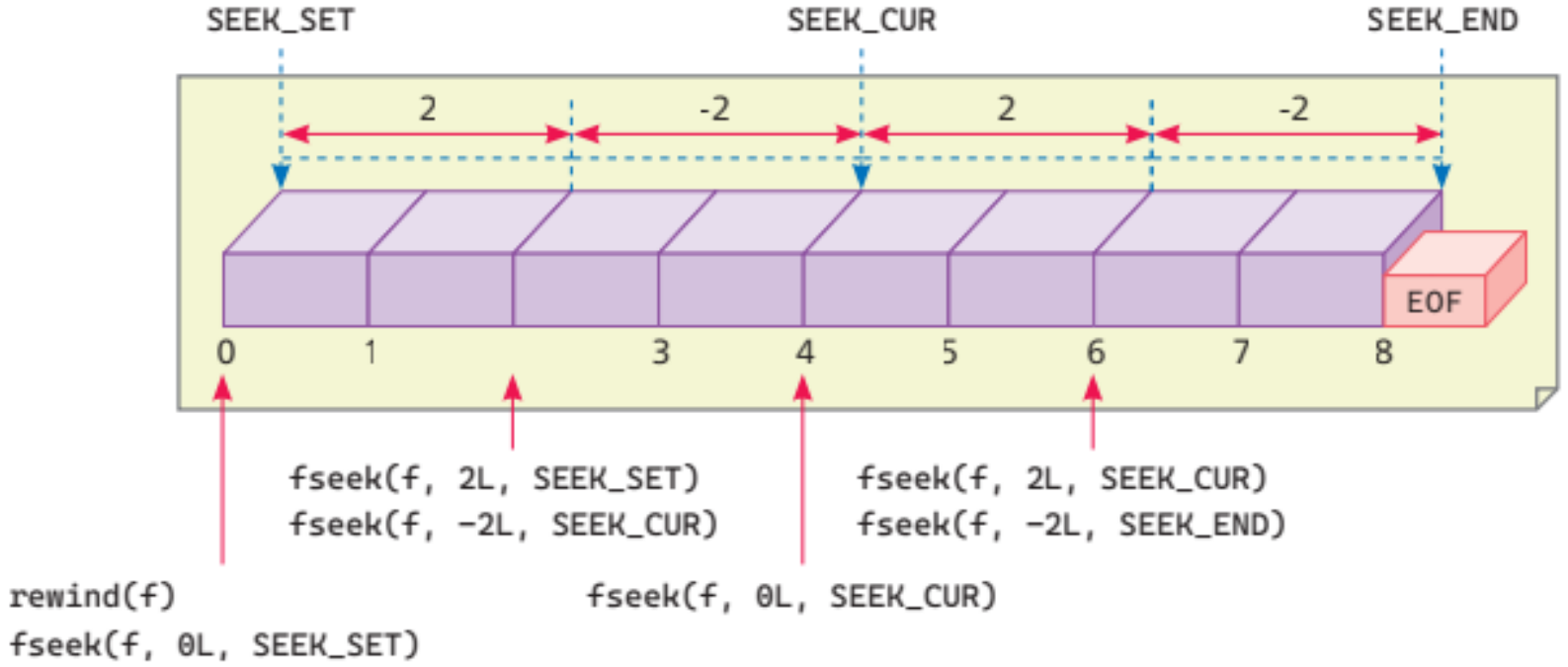
3. 파일 접근 처리



[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 파일 접근 처리

함수	기능
int fseek(FILE *, long offset, int pos)	파일 위치를 세 기준점(pos)으로부터 오프셋(offset)만큼 이동
long ftell(FILE *)	파일의 현재 파일 위치를 반환
void rewind(FILE *)	파일의 현재 위치를 0 위치(파일의 시작점)로 이동



[출처] 강환수 외, Perfect C 3판, 인피니티북스

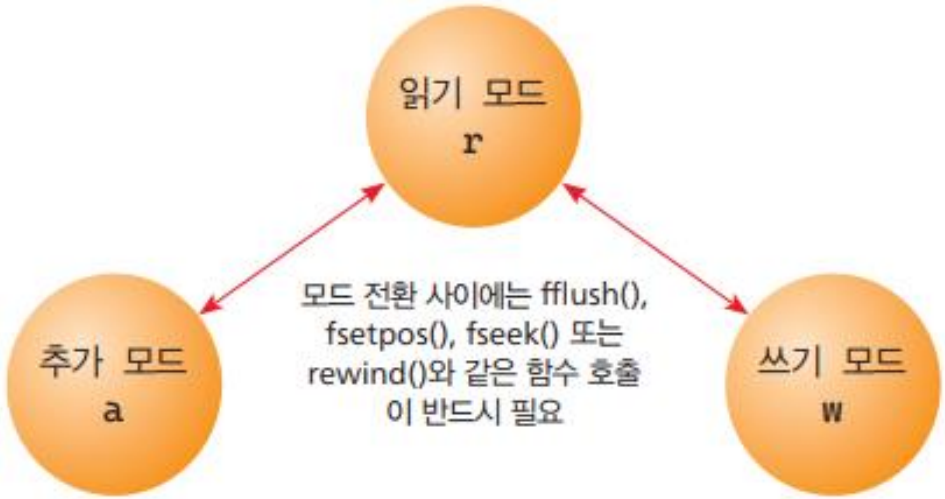
3. 파일 접근 처리

◆ 함수 fopen_s()의 모드 종류

모드	의미			
	파일 열기 모드	모드 전환	파일이 있는 경우	파일이 없는 경우
r	읽기(read)	쓰기(write) 불가능	파일의 처음에서 읽기 시작	에러 발생
w	쓰기(write)	읽기(read) 불가능	이전 내용을 지워지고 파일의 처음부터 쓰기 시작	새로 생성
a	추가(append)	읽기(read) 불가능	파일의 마지막에서 파일의 쓰기 시작하며, 파일 중간에 쓰는 것은 불가능	새로 생성
r+	읽기(read)	쓰기(write)	파일의 처음에서 읽기 시작	에러 발생
w+	쓰기(write)	읽기(read)	이전 내용을 지워지고 파일의 처음부터 쓰기 시작	새로 생성
a+	추가(append)	읽기(read)	파일의 마지막에서 파일의 쓰기 시작하며, 파일 중간에 쓰는 것은 불가능	새로 생성

[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 파일 접근 처리



◆ 이진 파일 열기 함수 fopen()의 모드 종류

모드		의미
rb		이진 파일의 읽기(read) 모드로 파일을 연다.
wb		이진 파일의 쓰기(write) 모드로 파일을 연다.
ab		이진 파일의 추가(append) 모드로 파일을 연다.
rb+	r+b	이진 파일의 읽기(read)와 쓰기(write) 모드로 파일을 연다.
wb+	w+b	이진 파일의 읽기(read)와 쓰기(write) 모드로 파일을 연다.
ab+	a+b	이진 파일의 추가(append) 모드로 파일을 연다.

[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 파일 접근 처리

자료	종류	표준 입출력	파일 입출력
문자	입력	int getchar(void)	int getc(FILE *) int fgetc(FILE *)
	출력	int putchar(int)	int putc(int, FILE *) int fputc(int, FILE *)
문자열	입력	char * gets(char *)	char * fgets(char *, int, FILE *)
	출력	int puts(const char *)	int fputs(const char *, FILE *,)
서식 자료	입력	int scanf(const char *, ···) int scanf_s(const char *, ···)	int fscanf(FILE *, const char *, ···) int fscanf_s(FILE *, const char *, ···)
	출력	int printf(const char *, ···)	int fprintf(FILE *, const char *, ···)

기능	함수 원형
파일 삭제	int remove(char const* _FileName);
파일 또는 폴더 이름 바꾸기	int rename(char const* _OldFileName, char const* _NewFileName);

[출처] 강환수 외, Perfect C 3판, 인피니티북스

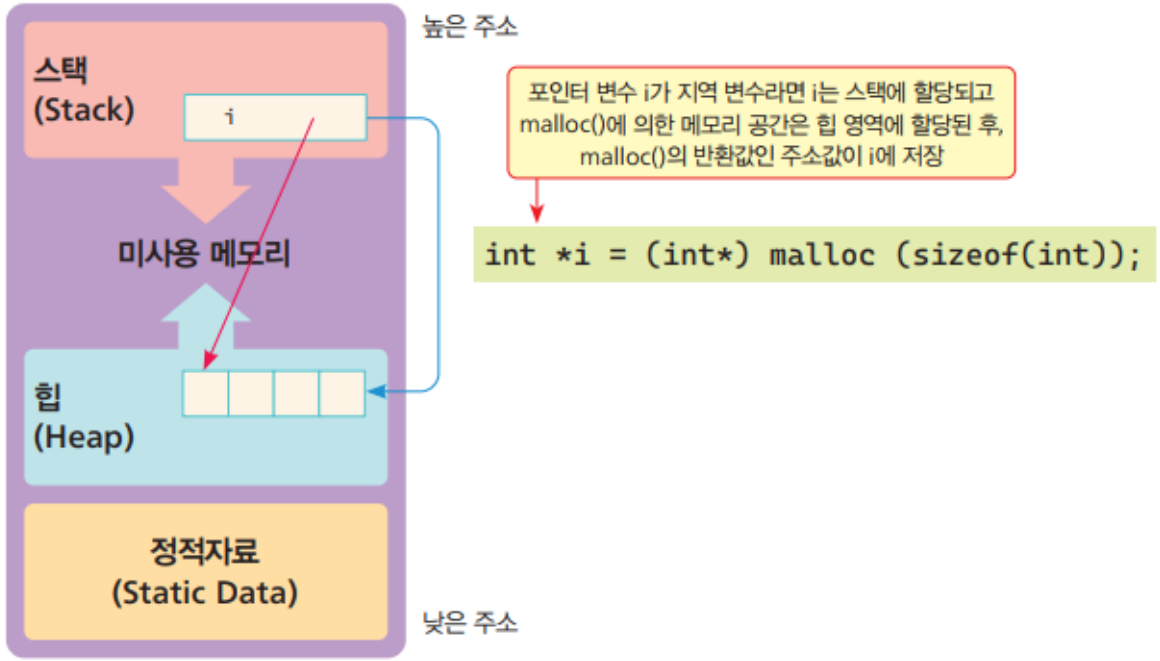
Ⅰ. 파일 처리

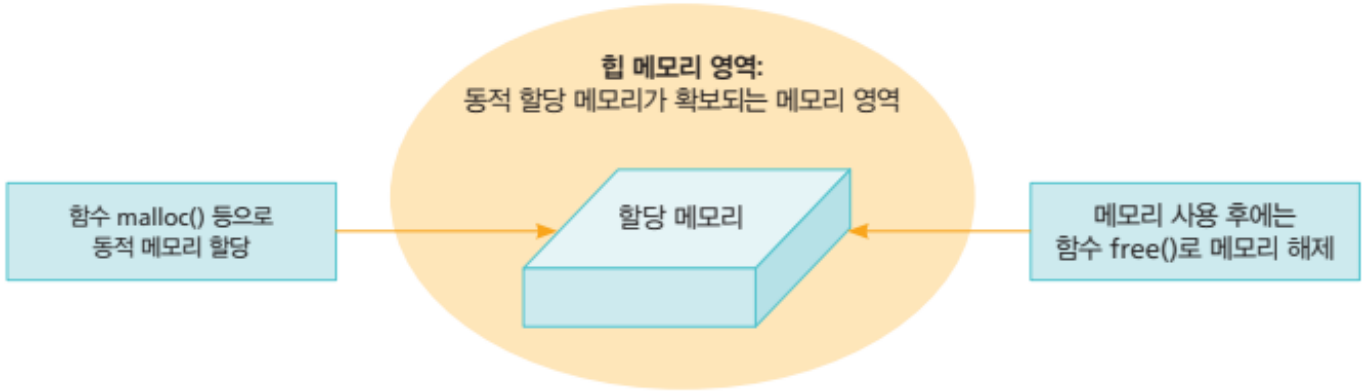
1교시 수업을 마치겠습니다.

Ⅱ. 동적 메모리와 연결 리스트

1. 동적 메모리
2. 자기참조 구조체
3. 연결리스트

1. 동적 메모리

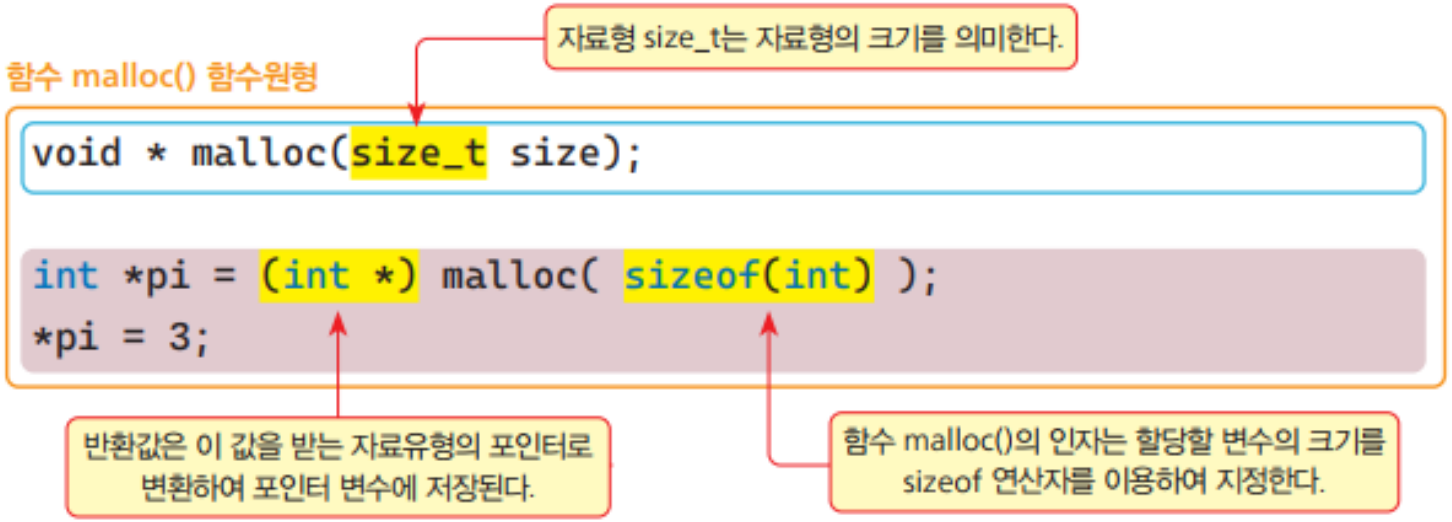




[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 동적 메모리

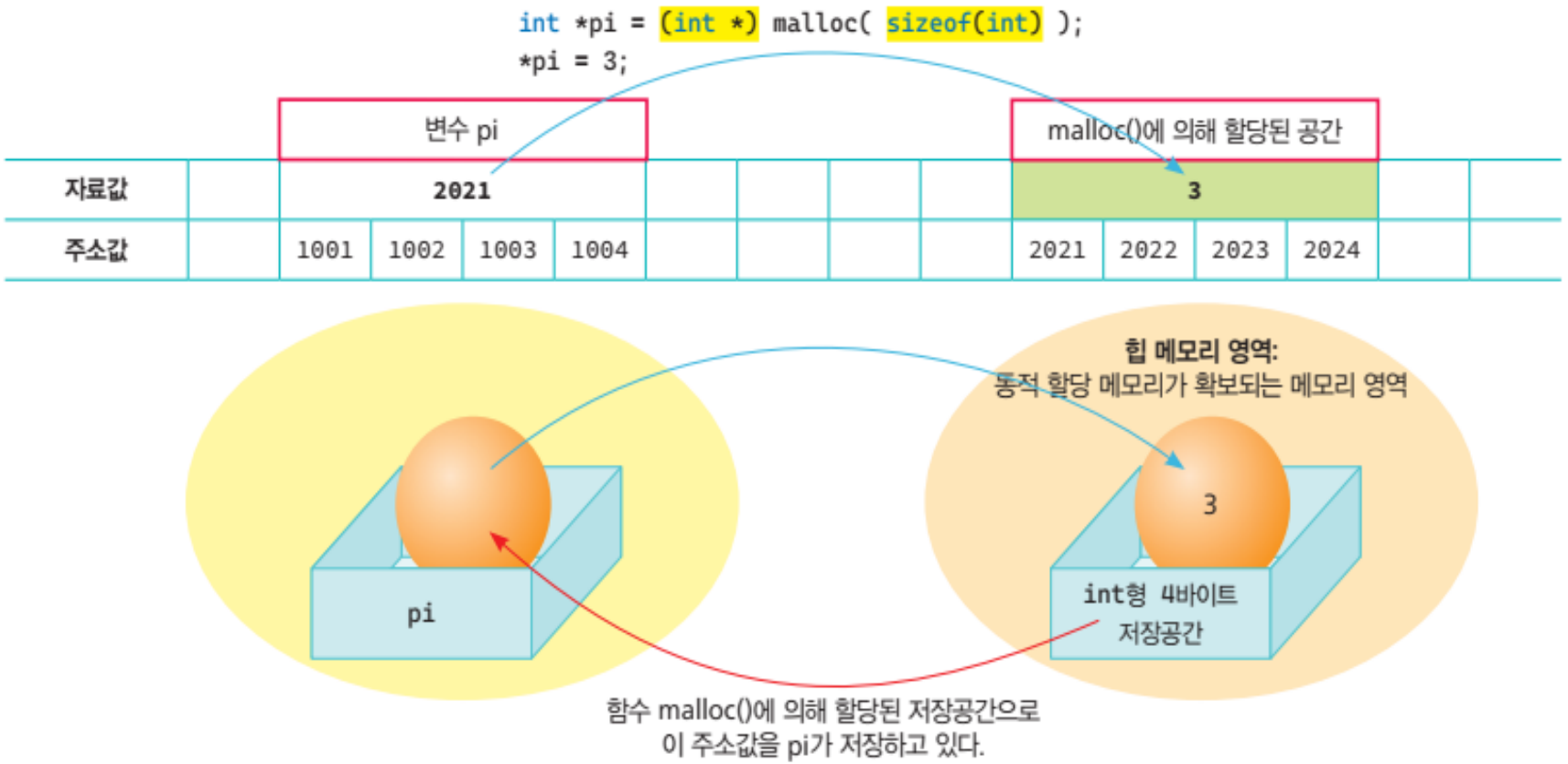
메모리 연산	기본값	함수 원형	기능
메모리 할당	없음	void * malloc(size_t)	인자만큼의 메모리 할당 후 기본 주소 반환
	0	void * calloc(size_t , size_t)	뒤 인자 만큼의 메모리 크기로 앞 인자 수 만큼 할당 후 기본 주소 반환
기존 메모리 변경	이전 값	void * realloc(void *, size_t)	앞 인자의 메모리를 뒤 인자 크기로 변경 후, 기본 주소 반환
메모리 해제	해당 없음	void free(void *)	인자를 기본 주소로 갖는 메모리 해제



[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 동적 메모리

◆ 할당



[출처] 강환수 외, Perfect C 3판, 인피니티북스

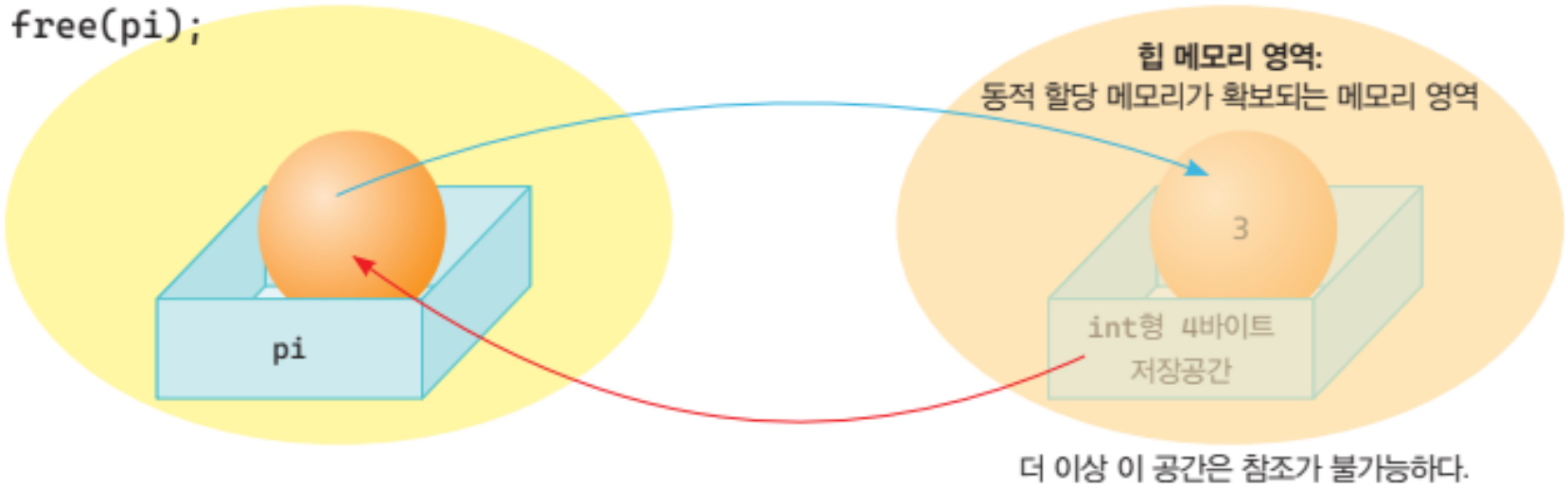
1. 동적 메모리

◆ 해제

함수 free() 함수원형

```
void free(void *);
```

```
free(pi);
```



[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 동적 메모리

실습예제 16-1 | Prj01 | 01malloc.c | 함수 malloc()을 이용하여 int형 저장공간을 확보하여 처리 | 난이도: ★★

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   int* pi = NULL;

   pi = (int*) malloc( sizeof(int) ); //동적 메모리 할당
   if (pi == NULL) //동적 메모리 할당 검사
   {
      printf("메모리 할당에 문제가 있습니다.");
      exit(1);
   };

   *pi = 7; //동적 메모리에 내용 값 7 저장
   printf("주소값: *pi = %p, 저장 값: p = %d\n", pi, *pi);

   free(pi); //동적 메모리 해제

   return 0;
}
```

함수 malloc()으로 동적 메모리 할당 후 int형 포인터인 pi에 저장하기 위해 자료형 변환 (int *)이 필요

만일을 대비해서 함수 malloc()의 반환 값을 점검하는 모듈이 필요

결과

```
주소값: *pi = 00000145F70D6A30, 저장 값: p = 7
```

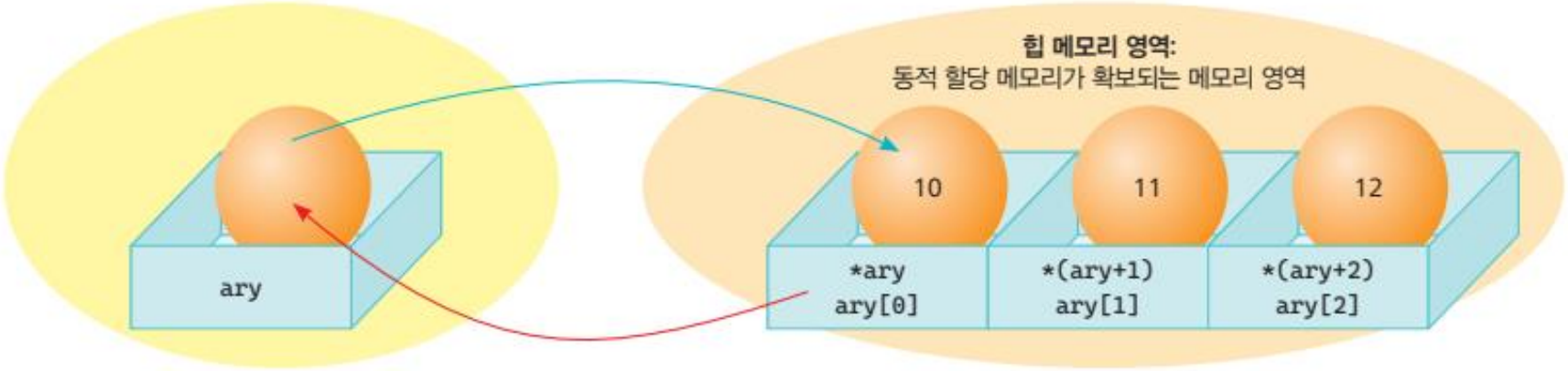
[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 동적 메모리

◆ 배열 공간 할당

```
int *ary;
ary = (int *) malloc( sizeof(int)*3 );
ary[0] = 10; ary[1] = 11; ary[2] = 12;
```

함수 malloc()에 의해 할당된 저장공간은 int형 3개이며 주소 값을 ary에 저장하고 있다. ary[i]로 각 원소를 참조할 수 있다.



[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 동적 메모리

함수 calloc() 함수원형

할당될 메모리 항목의 개수이다.

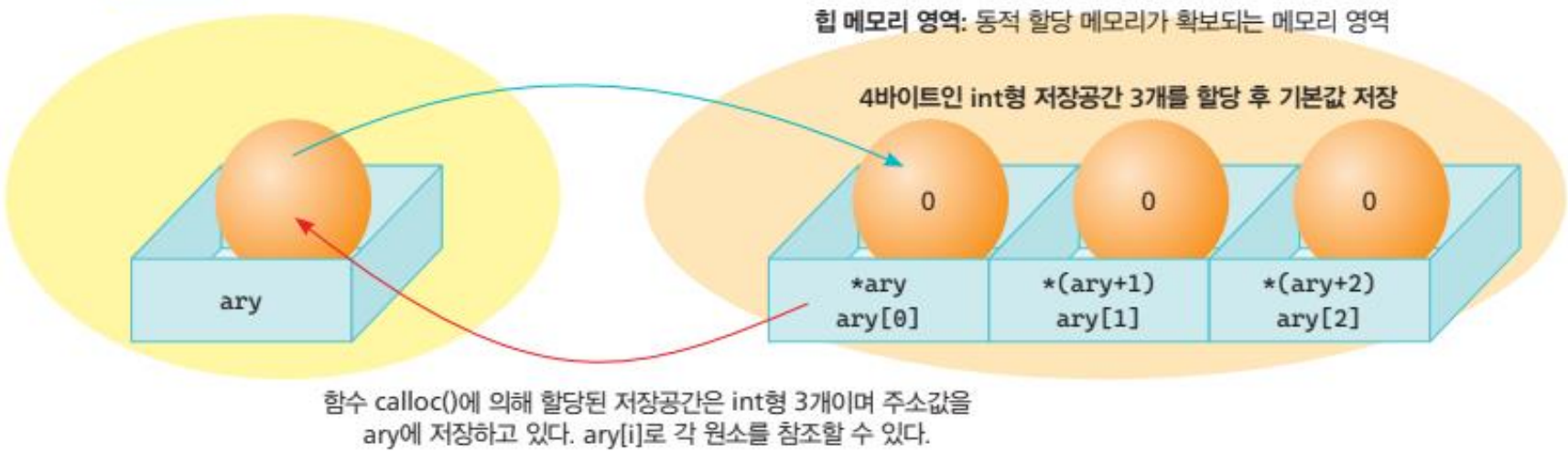
할당될 메모리 한 원소의 크기이다.

```
void * calloc(size_t num, size_t size);
```

- 함수 calloc()은 원소 크기가 size인 배열크기 num개의 공간을 할당하여 포인터를 반환하고 원소값은 모두 0으로 저장

반환값은 이 값을 받는 자료유형의 포인터로 변환하여 포인터 변수에 저장된다.

```
int *ary = NULL;
ary = (int *) calloc( 3, sizeof(int) )
```



함수 calloc()에 의해 할당된 저장공간은 int형 3개이며 주소값을 ary에 저장하고 있다. ary[i]로 각 원소를 참조할 수 있다.

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 자기참조 구조체

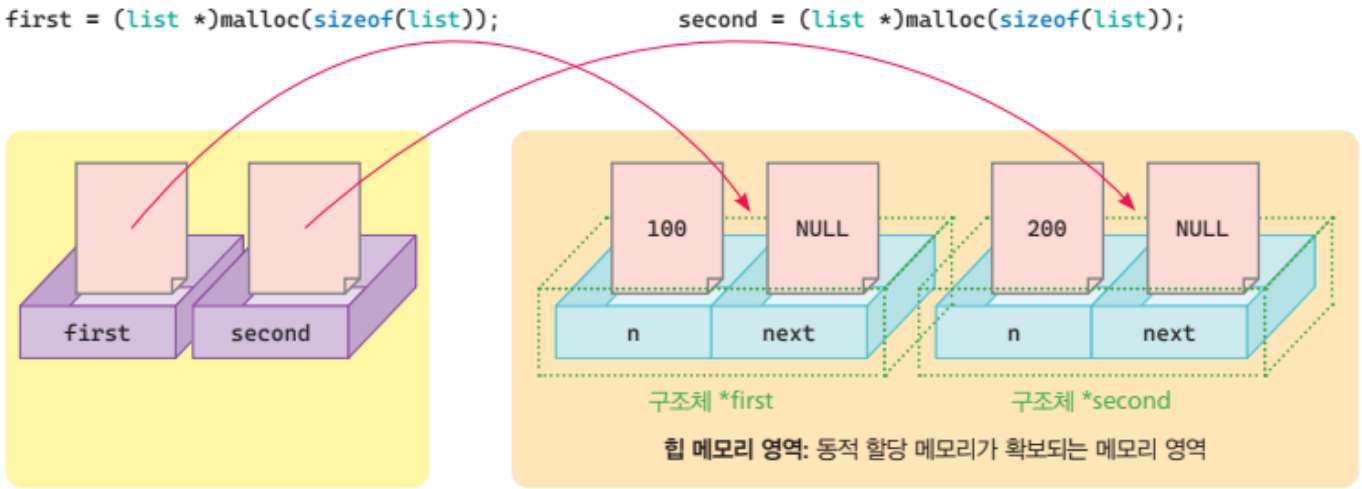
```
struct selfref {
    int n;
    struct selfref *next;
    //struct selfref one;    //컴파일 오류 발생
}
```

error C2079: 'one'은(는) 정의되지 않은 struct 'selfref'을(를) 사용합니다.

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 자기참조 구조체

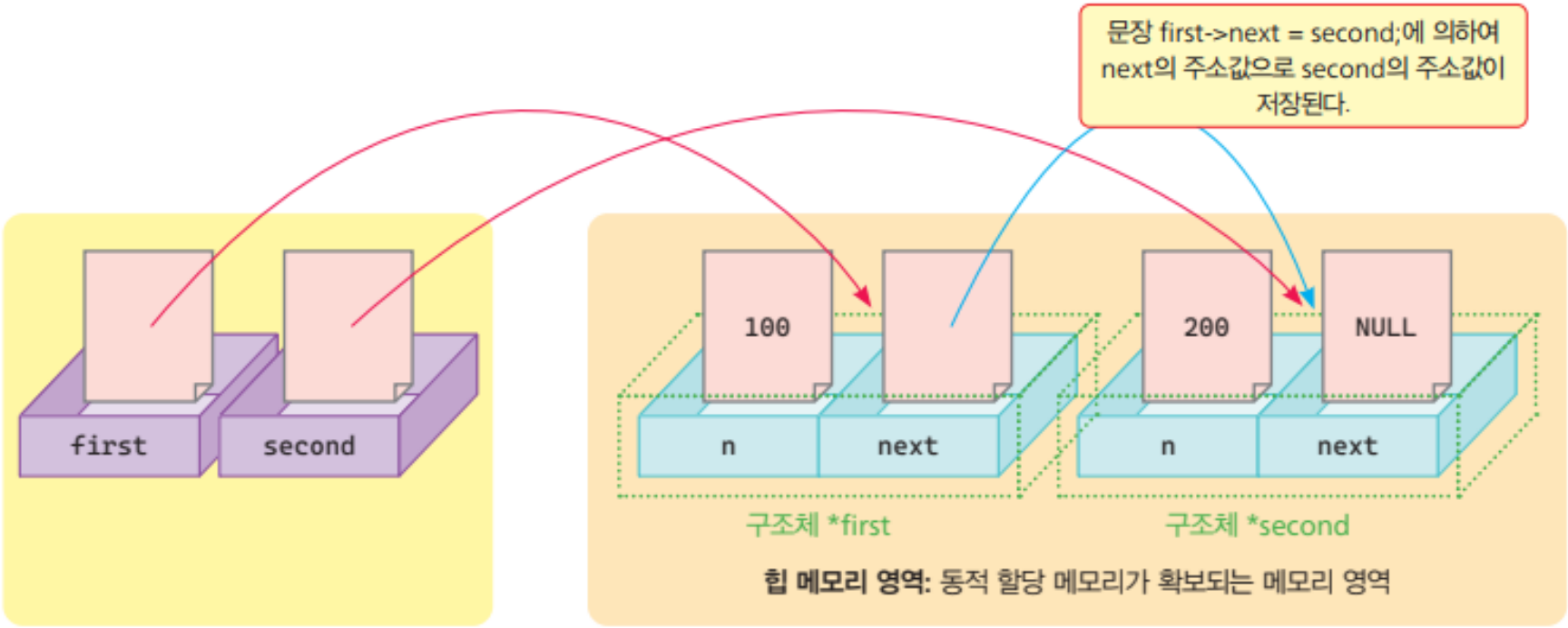
```
//❶ 우선 구조체 struct selfref를 하나의 자료형인 list 형으로 정의
typedef struct selfref list;

//❷ 두 구조체 포인터 변수 first와 second를 선언한 후,
//   함수 malloc()을 이용하여 구조체의 멤버를 저장할 수 있는 저장공간을 할당
list *first = NULL, *second = NULL;
first = (list *)malloc(sizeof(list));
second = (list *)malloc(sizeof(list));

//❸ 구조체 포인터 first와 second의 멤버 n에 각각 정수 100, 200을 저장하고,
//   멤버 next에는 각각 NULL을 저장
first->n = 100;
second->n = 200;
first->next = second->next = NULL;
```



[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 자기참조 구조체

```
//❹ first 다음에 second를 연결
first->next = second;  // 구조체 *first가 다음 *second 구조체를 가리키도록 하는 문장
```

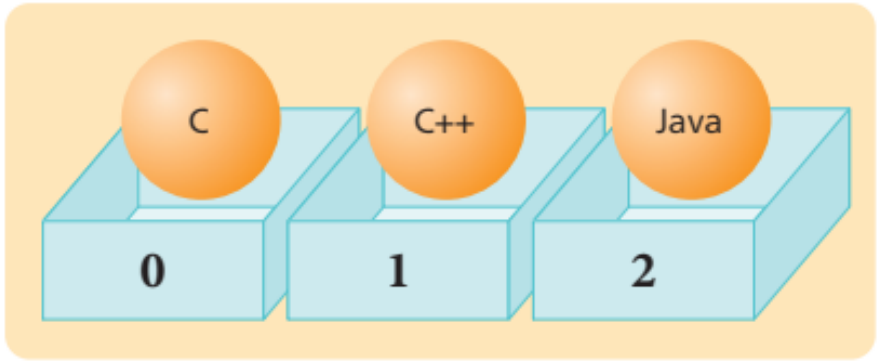


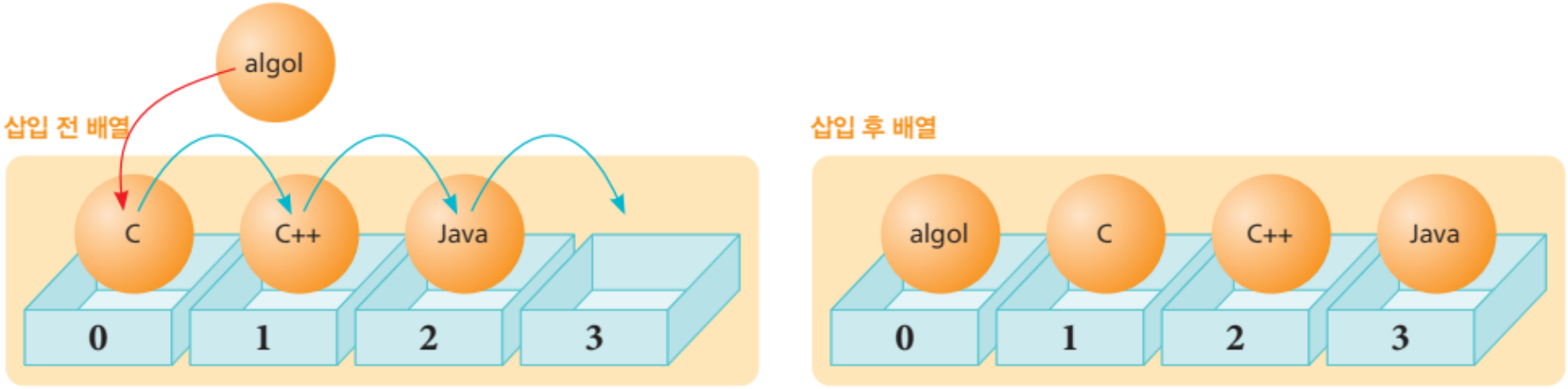
[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 연결 리스트

◆ 배열

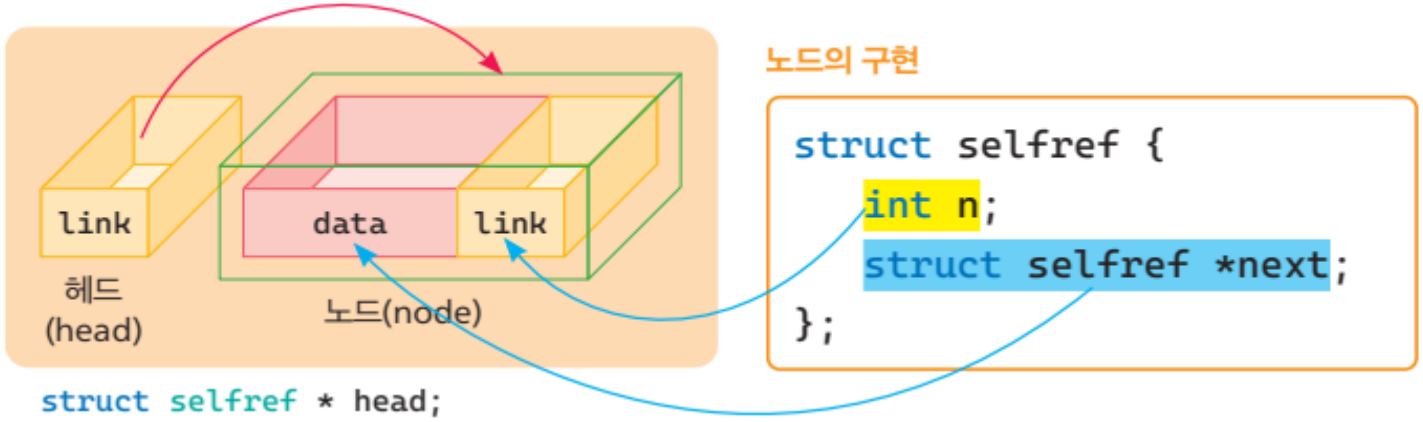
1	C
2	C++
3	Java





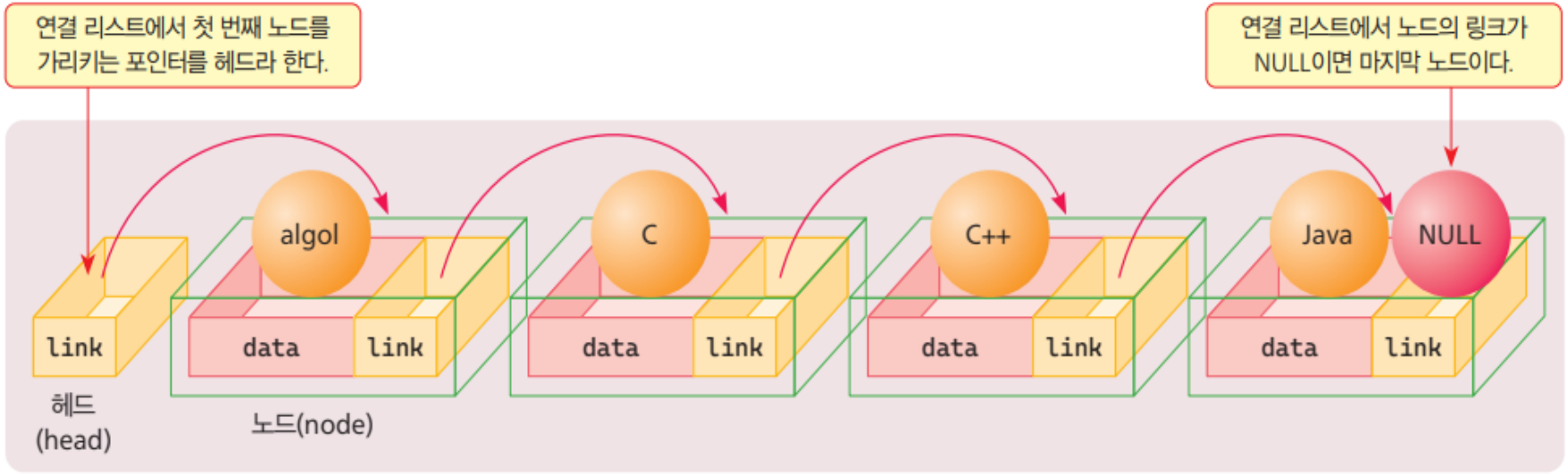
[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 연결 리스트



노드의 구현

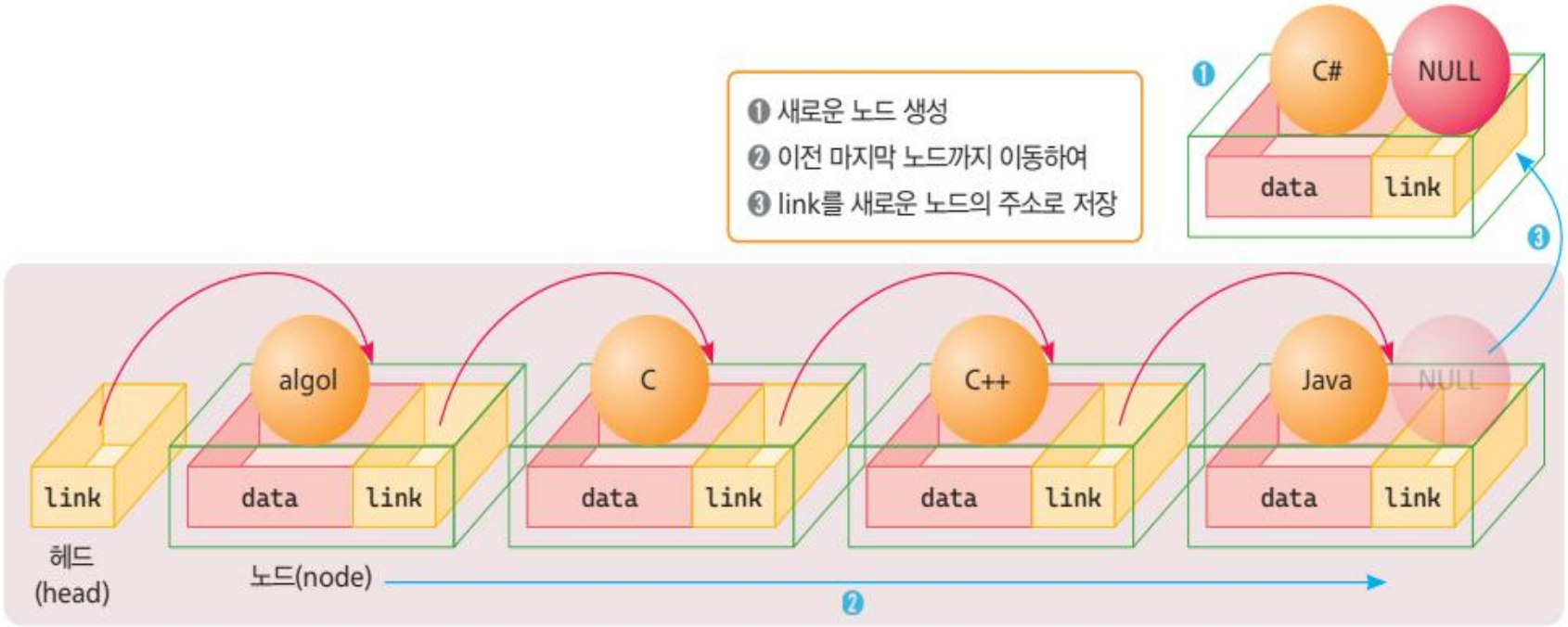
```
struct selfref {
    int n;
    struct selfref *next;
};
```



[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 연결 리스트

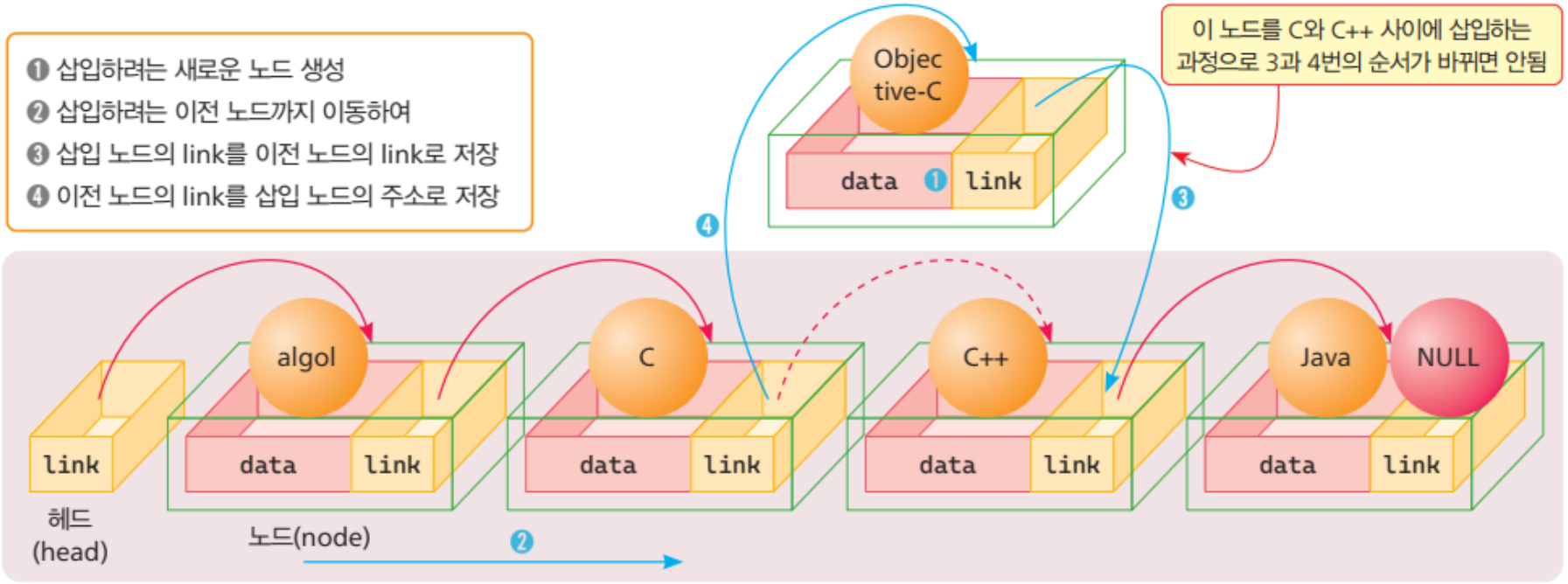
◆ 노드 추가



[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 연결 리스트

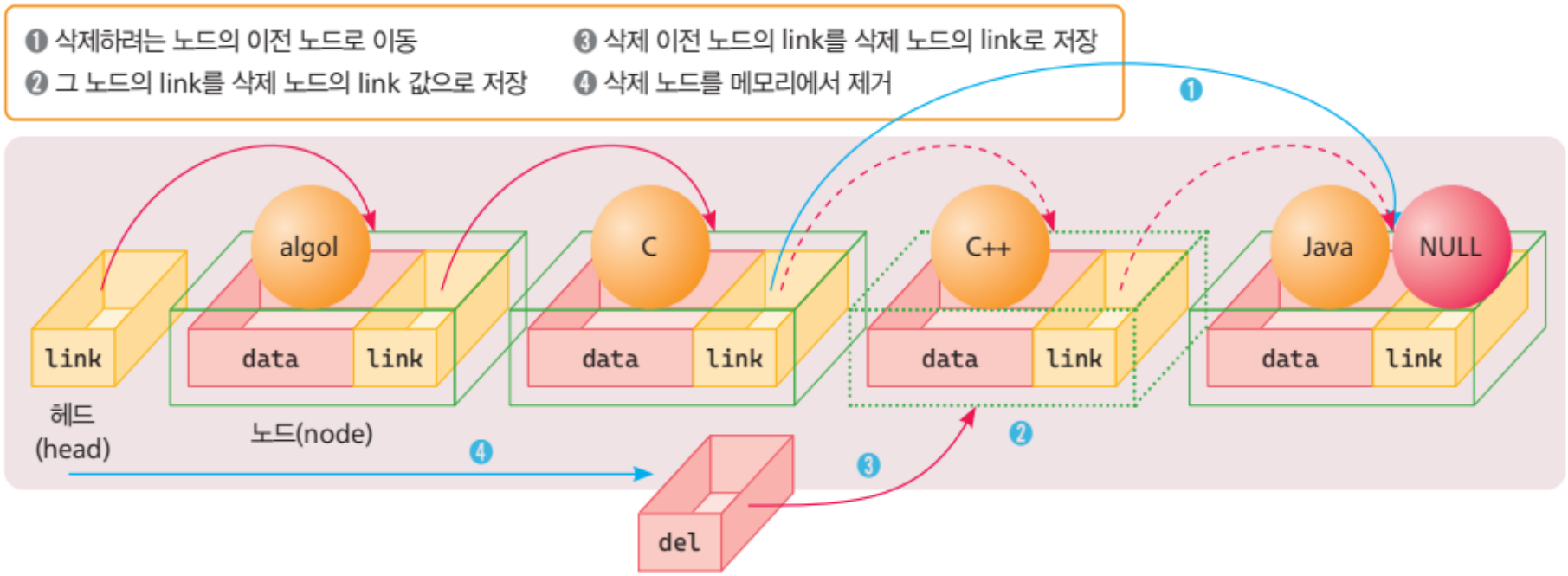
◆ 노드 삽입



[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 연결 리스트

◆ 노드 삭제



[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 연결 리스트

◆ 구조체 노드 정의와 생성

```c
struct linked_list { //자기참조 구조체 정의
    char* name;
    struct linked_list* next;
};

typedef struct linked_list NODE; //struct linked_list를 NODE로 재정의
typedef NODE* LINK; //NODE *를 LINK로 재정의
```

```c
LINK createNode(char* name) //노드를 생성하는 함수
{
    LINK cur; //새로 생성되는 노드의 주소를 저장할 변수 cur를 선언
    cur = (LINK)malloc(sizeof(NODE));
    if (cur == NULL)
    {
        printf("노드 생성을 위한 메모리 할당에 문제가 있습니다.\n");
        return NULL;
    }
    //언어 이름을 저장할 문자배열을 동적 할당하여 name에 저장
    cur->name = (char*)malloc(sizeof(char) * (strlen(name) + 1));
    strcpy(cur->name, name);
    cur->next = NULL; //다음 노드는 모르므로 NULL로 저장

    return cur;
}
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

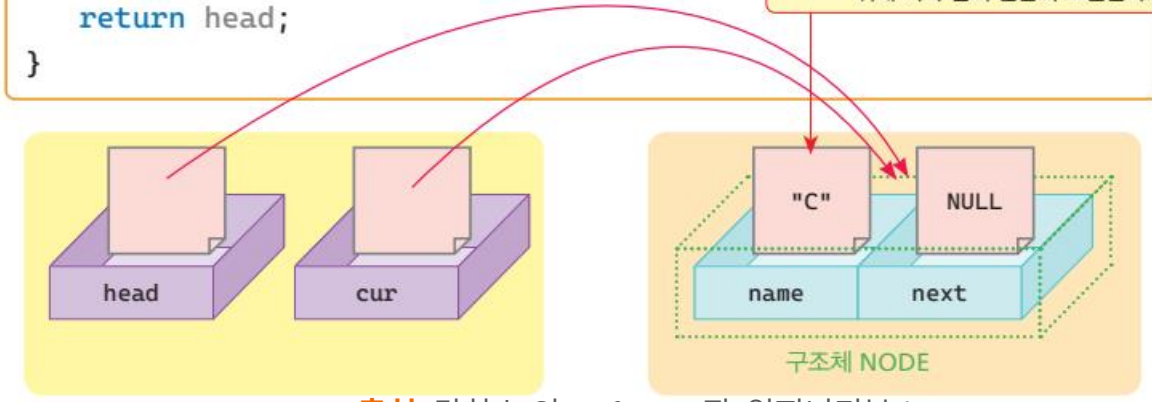
3. 연결 리스트

◆ 구조체 노드 추가

```
//cur 노드를 연결 리스트 head의 마지막 노드에 추가하는 함수
LINK append(LINK head, LINK cur)
{
   //지역 변수 nextNode를 선언하고 초기값으로 head를 저장
   LINK nextNode = head;
   ...
   return head;
}
```

```
//지역 변수 nextNode를 선언하고 초기값으로 head를 저장
LINK nextNode = head;
//만일 현재 헤드가 가리키는 것이 없다면, 즉 연결리스트의 노드가 하나도 없는 경우
if (head == NULL)
{
   head = cur; //추가하려는 노드가 head가 됨
   return head;
}
```

실제 name은 문자 포인터이므로 문자열 "C"가 저장된 저장 공간의 주소값을 갖는다. 편의를 위해 이와 같이 간단히 표현한다.

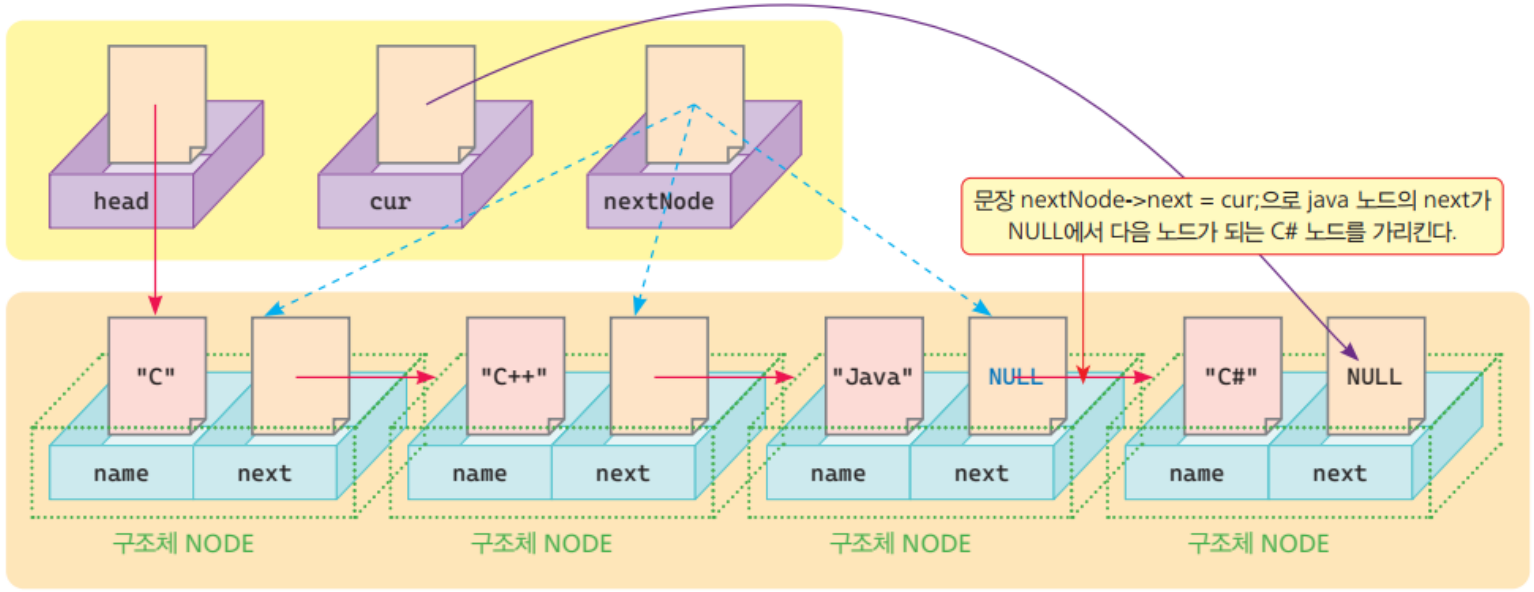


[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 연결 리스트

◆ 구조체 노드 추가

```
//멤버 next가 NULL일 때까지 이동하여 마지막 노드까지 이동
while (nextNode->next != NULL)
{
   nextNode = nextNode->next;
}
nextNode->next = cur;  //추가 노드를 현재 노드의 next에 저장
```



[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 연결 리스트

◆ 구조체 노드 출력

```
int printList(LINK head) //연결 리스트의 모든 노드 출력 함수
{
   int cnt = 0; //방문한 노드의 수를 저장
   LINK nextNode = head;
   //nextNode를 이용하여 연결 리스트의 처음부터 끝까지 순회
   while (nextNode != NULL)
   {
      //리스트의 순서로 노드를 방문하여 방문 횟수와 문자열 자료를 출력
      printf("%3d번째 노드는 %s\n", ++cnt, nextNode->name);
      nextNode = nextNode->next;
   }

   return cnt; //총 노드 방문 횟수를 반환하고 함수를 종료
}
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

Ⅱ. 동적 메모리와 연결 리스트

2교시 수업을 마치겠습니다.

Ⅲ. 전처리

1. 전처리 지시자 종류와 매크로
2. 조건부 컴파일 지시자
3. 전처리 연산자

1. 전처리 지시자 종류와 매크로

명령어	설명
#include	지정된 헤더파일 내용을 현재에 복사
#define	기호상수 정의
#undef	지정한 기호상수를 삭제
#if	주어진 연산식이 참이면 컴파일
#else	조건부 컴파일 블록의 마지막을 표시
#elif	조건부 컴파일 블록의 표시
#endif	조건부 컴파일 블록의 종료 표시
#ifdef	주어진 이름이 정의되었다면 컴파일
#ifndef	주어진 이름이 정의되지 않았다면 컴파일

[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 전처리 지시자 종류와 매크로

매크로	설명
__DATE__	가장 최근에 소스파일을 컴파일한 날짜를 [Mmm dd yyyy]으로 표시
__FILE__	현재 소스파일 이름으로 절대경로로 표시
__LINE__	현재 소스파일에서 이 문장이 있는 줄 번호
__TIME__	가장 최근에 소스파일을 컴파일한 시각으로 [시:분:초]로 표시
__TIMESTAMP__	현재 소스파일을 마지막으로 수정한 시각으로 [요일 월 날짜 시:분:초 년]으로 표시

실습예제 16-7 Prj07 07sysmacro.c 이미 정의된 __DATE__와 같은 예약 매크로의 이용 난이도: ★★

```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;
}
```

결과

```
Dec 15 2020
C:\Kang C code\ch16\Prj07\07sysmacro.c
7
13:47:21
Tue Dec 15 13:47:21 2020
```

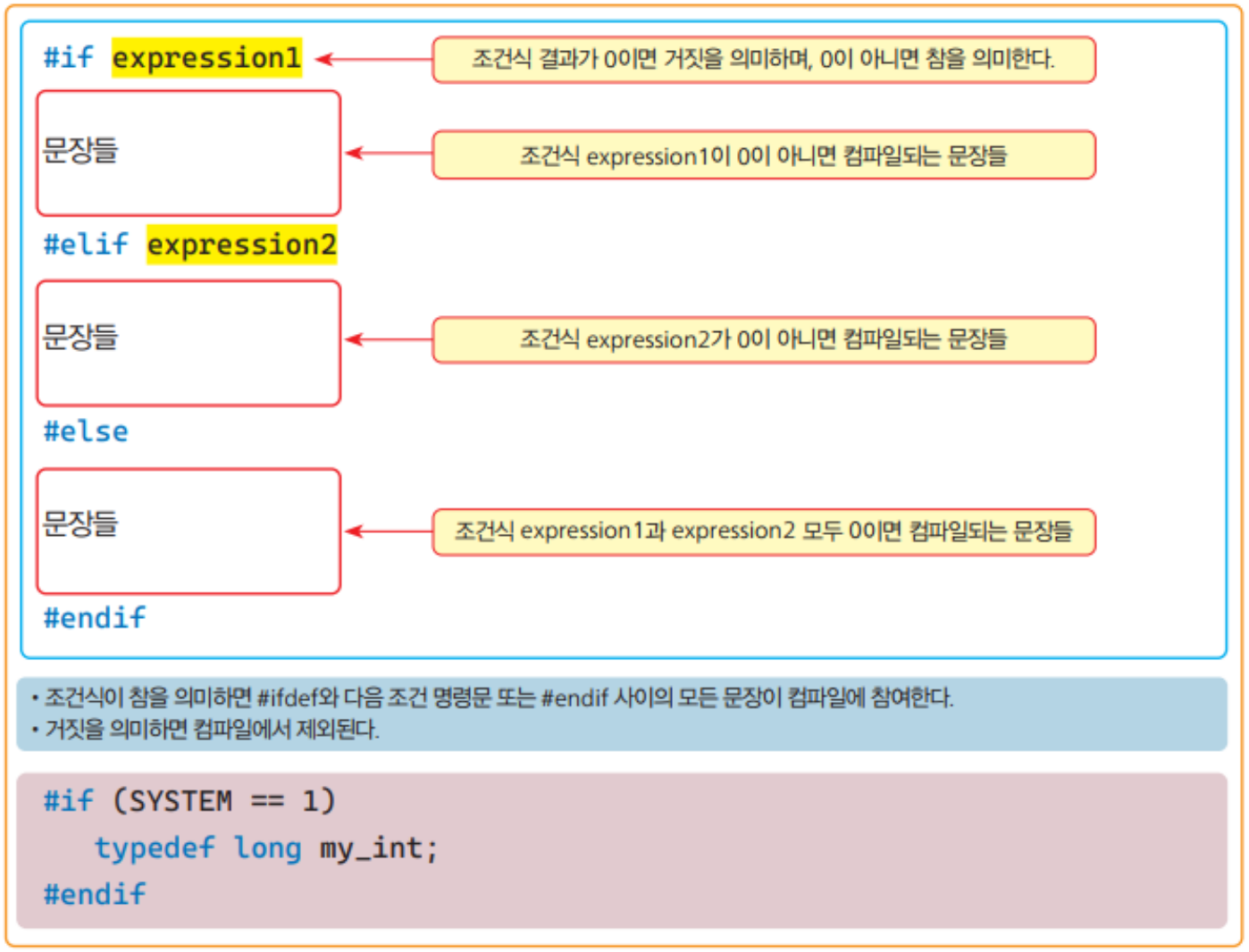
현재 소스파일 이름으로 절대경로로 표시

소스파일의 최종 수정된 날짜와 시각 정보를 표시

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 조건부 컴파일 지시자

명령어 #if #elif #else #endif



- 조건식이 참을 의미하면 #ifdef와 다음 조건 명령문 또는 #endif 사이의 모든 문장이 컴파일에 참여한다.
- 거짓을 의미하면 컴파일에서 제외된다.

```
#if (SYSTEM == 1)
   typedef long my_int;
#endif
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 조건부 컴파일 지시자

❖ 명령문 #if 조건식

- 기호상수와 정수 상수, 문자 상수만 사용 가능
- 실수 상수와 문자열 상수, 변수 등은 사용 불가능
- 그 결과도 반드시 정수
- 조건식에는 관계연산자와 논리연산자 그리고 사칙연산을 사용 가능
  - 조건식에 변수는 사용 불가능

❖ 다음은 잘못된 #if 조건식

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

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

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

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 조건부 컴파일 지시자

❖ 전처리 연산자 defined (기호상수)

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

❖ if defined #ifdef #endif

지시자 #ifdef #endif

```
#ifdef 기호상수
   ...
#endif
```

명령어 #define 또는 명령행에서 정의된 기호상수이다.

• 기호상수가 정의되었다면 #ifdef와 #endif 사이의 모든 문장이 컴파일에 참여한다. 정의되지 않았다면 컴파일에서 제외된다.

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

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 조건부 컴파일 지시자

❖ 전처리 연산자 #ifndef

전처리 명령어 #ifndef

```
#ifndef 기호상수
    ...
#endif
```

명령어 #define 또는 명령행에서 정의된 기호상수이다.

• 기호상수가 정의되지 않았다면 #ifdef와 #endif 사이의 모든 문장이 컴파일에 참여한다. 정의되었다면 컴파일에서 제외된다.

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

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 조건부 컴파일 지시자

❖ 기호 상수 삭제 #undef

- #undef는 이미 정의된 기호 상수를 해지하는 지시자
- 다음 구문
  - 20으로 정의된 기호상수 SIZE
  - 전처리기 지시자 #undef SIZE로 그 효력을 상실하게 함

```
#define SIZE 20
#ifdef SIZE
#undef SIZE
#endif
```

  - 일반적으로 기호상수를 삭제하기 전
  - #ifdef로 이전에 정의됨을 확인한 후 삭제하는 것을 추천

[출처] 강환수 외, Perfect C 3판, 인피니티북스

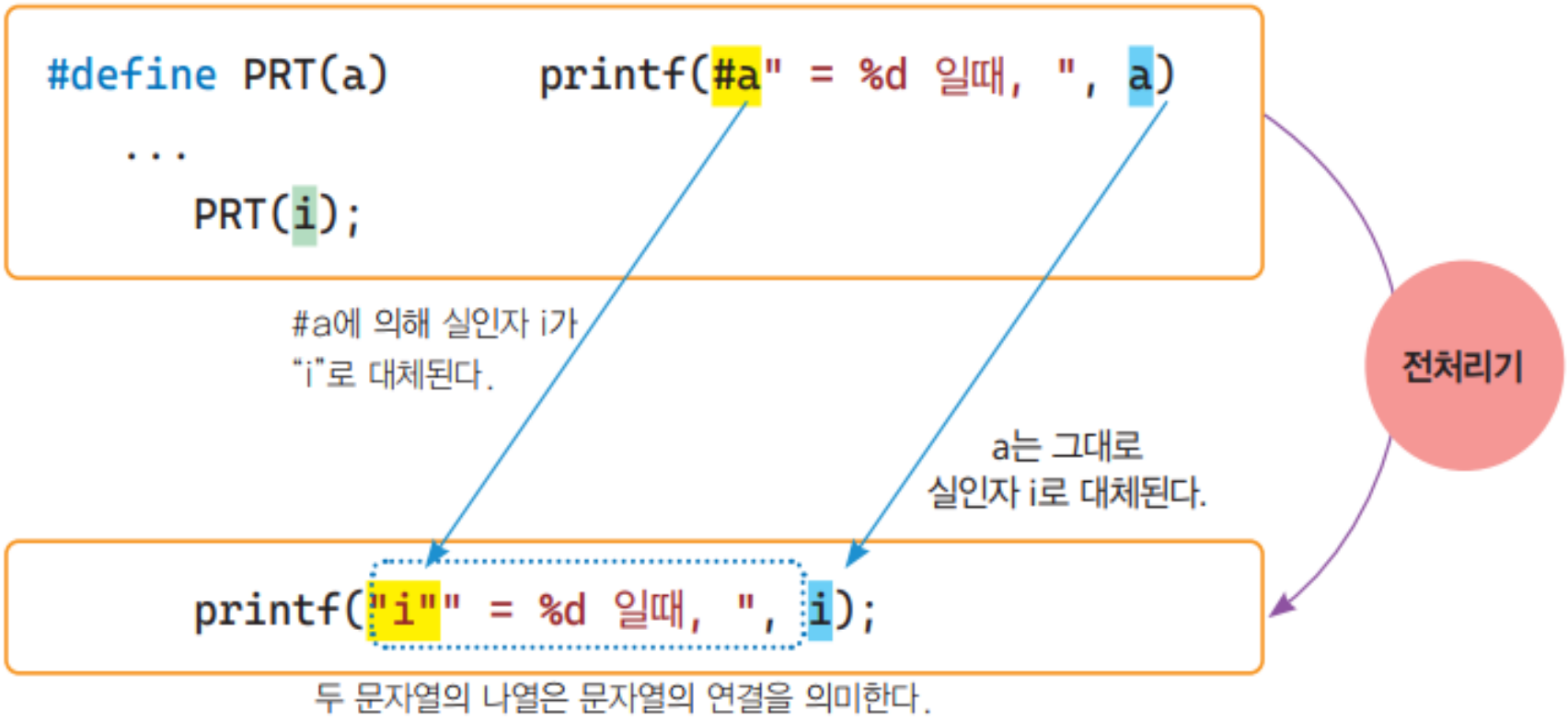
3. 전처리 연산자

연산자	이름	사용 예	기능
#	문자열 만들기 연산자 (Stringizing operator)	#인자	인자 앞 뒤에 큰따옴표를 붙여 인자를 문자열로 만드는 연산자
#@	문자 만들기 연산자 (charizing operator)	#@인자	인자 앞 뒤에 작은따옴표를 붙여 인자를 문자로 만드는 연산자
##	토큰 붙이기 연산자 (token-pasting operator)	인자##인자	인자를 다른 토큰들과 연결해주는 연산자
defined	정의 검사 연산자 (defined operator)	defined WINDOWS	상수로 정의되어 있는지 검사하는 연산자

[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 전처리 연산자

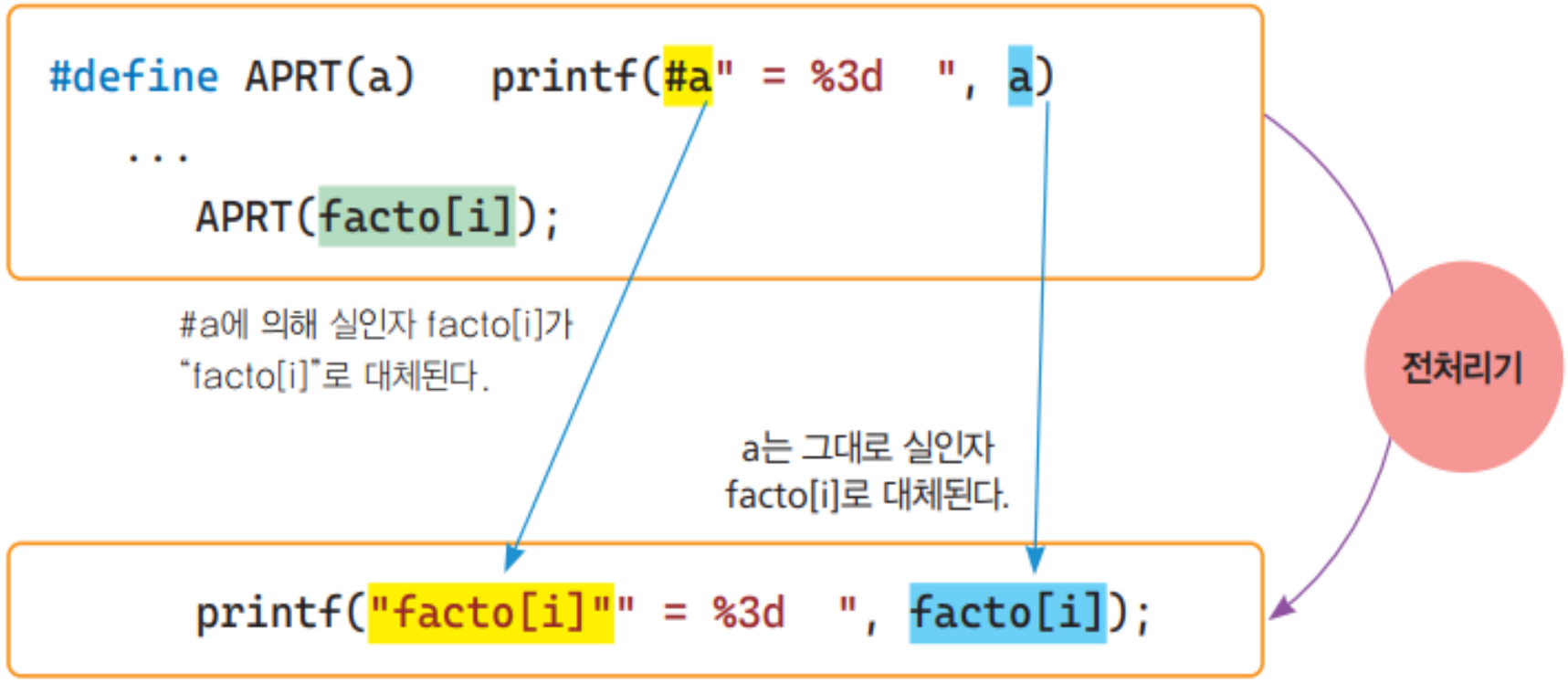
◆ 문자열 만들기 연산자



[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 전처리 연산자

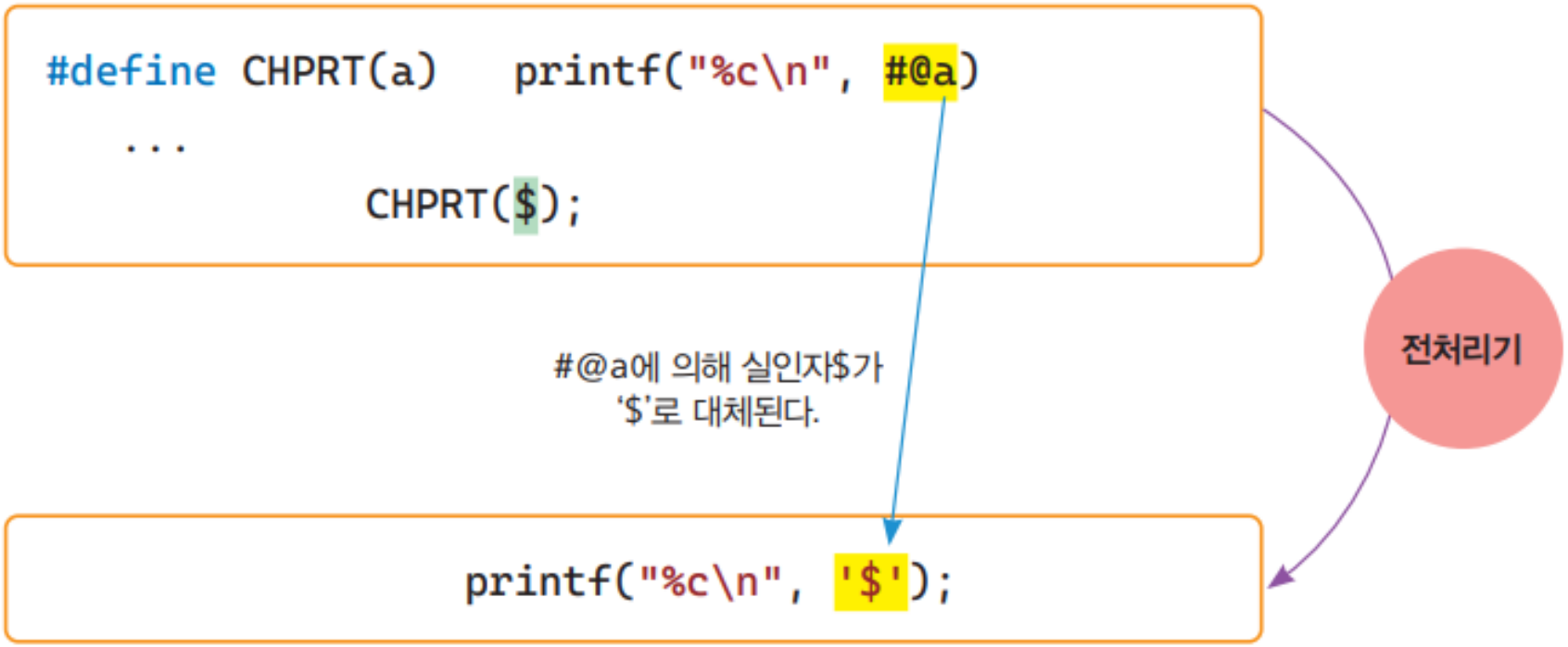
◆ 문자열 만들기 연산자



[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 전처리 연산자

◆ 문자열 만들기 연산자 #@



[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 전처리 연산자

◆ 토큰 붙이기 연산자

```
#define AIPRT(a, i) printf(#a"[%d] = %3d\n", i, a##[i])
    ...
    AIPRT(facto, i);
```

형식인자 a##[i]에 의해 실인자 facto와
i는 facto[i]로 대체된다.

전처리기

```
printf("facto""[%d] = %3d\n", i, facto[i]);
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

Ⅲ. 전처리

3교시 수업을 마치겠습니다.