14강

동적 메모리

동양미래대학교 강환수 교수

본 강의 사용 및 참조 자료

- Perfect C, 3판, 강환수 외 2인 공저, 인피니티북스, 2021

16장 동적 메모리와 전처리

목차

01

동적 메모리 개요

정적 메모리 할당

- static memory allocation
  - 변수와 배열, 구조체 모두 프로그램 실행 전에 필요한 만큼의 변수를 선언하여 사용
    - 프로그램 실행 중에 변수의 수를 늘리거나 줄이는 것이 불가능
  - 컴파일 이전에 저장공간 수나 크기를 정한 메모리 할당 방법
    - 프로그램이 실행되기 이전에 변수의 저장 공간 크기가 결정
    - 프로그램 또는 함수가 시작되면 메모리에 할당되어 사용
      - 그 모듈이나 프로그램이 종료되면 변수가 메모리에서 삭제되는 방식
    - 메모리의 사용 예측이 부정확한 경우
      충분한 메모리를 미리 확보해야 하므로 비효율

동적 메모리 할당

- dynamic memory allocation
  - 프로그램 실행 중에 필요한 메모리를 할당하는 방법
    - 정적 할당 방식인 변수 선언에 비해 상대적으로 다소 어려움
  - 메모리 사용 예측이 정확하지 않고
    - 실행 중에 메모리 할당이 필요 시 적합

정적 메모리 할당 방식

```
int i;
long prod = 1;
int facto[6];
char *[] = {"algol", "pascal"
            "C", "C++" };
```

동적 메모리 할당 방식

```
int* pi = NULL;

pi = (int*) malloc( sizeof(int) ); //동적 메모리 할당
*pi = 7; //동적 메모리에 내용 값 7 저장
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 698쪽

동적 메모리 관련 함수

- 함수 malloc( )의 호출로 힙(heap) 영역에 확보
- 메모리는 사용 후 함수 free( )로 해제
  - 메모리 해제를 하지 않으면
    메모리 부족과 같은 문제를 일으킬 수 있으니 꼭 해제하는 습관

동적 메모리 할당 함수

- malloc( ), calloc( ), realloc( ) 3가지
  - 헤더파일 stdlib.h 필요
  - 반환 형이 void 포인터(void *)
    - 메모리 할당에 요구한 자료의 포인터 형으로 변환
- 동적으로 할당된 메모리를 해제
  - 함수 free( )

함수 malloc()

- 동적 메모리 할당 기능의 기본 함수 malloc()
  - 인자인 자료형 크기 size만큼의 메모리를 할당
  - 성공하면 할당된 공간의 void 포인터를 반환
  - 실패하면 NULL을 반환

함수 malloc() 함수원형

```
void * malloc(size_t size);
```

자료형 size_t는 자료형의 크기를 의미한다.

```
int *pi = (int *) malloc( sizeof(int) );
*pi = 3;
```

반환값은 이 값을 받는 자료유형의 포인터로 변환하여 포인터 변수에 저장된다.

함수 malloc()의 인자는 할당할 변수의 크기를 sizeof 연산자를 이용하여 지정한다.

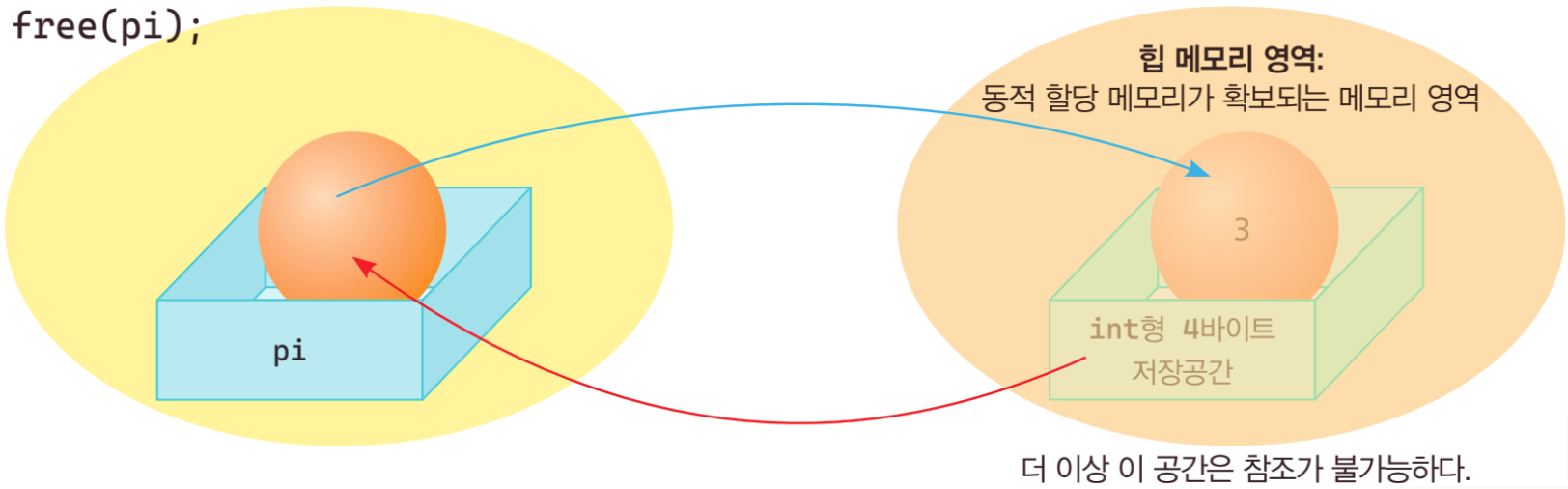
[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 700쪽

함수 malloc() 인자

- 인자
  - 메모리 할당의 크기를 지정
- 반환
  - 할당된 메모리의 시작 주소
    - 모든 자료형의 포인터로 이용할 수 있도록 void *를 사용
  - 확보된 공간의 주소는 int *의 변수에 저장
    - 간접연산자 *pi를 이용하여 원하는 값을 수정 가능

함수 free()

- 메모리 해제에 이용되는 함수
  - 함수 malloc()의 반환 주소를 저장한 변수 pi를 해제
  - 인자로 해제할 메모리 공간의 주소값을 갖는 포인터를 이용
  - 변수 pi가 가리키는 4바이트의 자료값이 해제
    - 더 이상 사용할 수 없음



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 702쪽

실습예제

Prj01 01malloc.c 함수 malloc()을 이용하여 int형 저장공간을 확보하여 처리 난이도: ★★

```
01  #include <stdio.h>
02  #include <stdlib.h>
03
04  int main(void)
05  {
06     int* pi = NULL;
07
08     pi = (int*) malloc( sizeof(int) ); //동적 메모리 할당
09     if (pi == NULL) //동적 메모리 할당 검사
10     {
11        printf("메모리 할당에 문제가 있습니다.");
12        exit(1);
13     };
14
15     *pi = 7; //동적 메모리에 내용 값 7 저장
16     printf("주소값: *pi = %p, 저장 값: p = %d\n", pi, *pi);
17
18     free(pi); //동적 메모리 해제
19
20     return 0;
21  }
```

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

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

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 702쪽

함수 malloc()

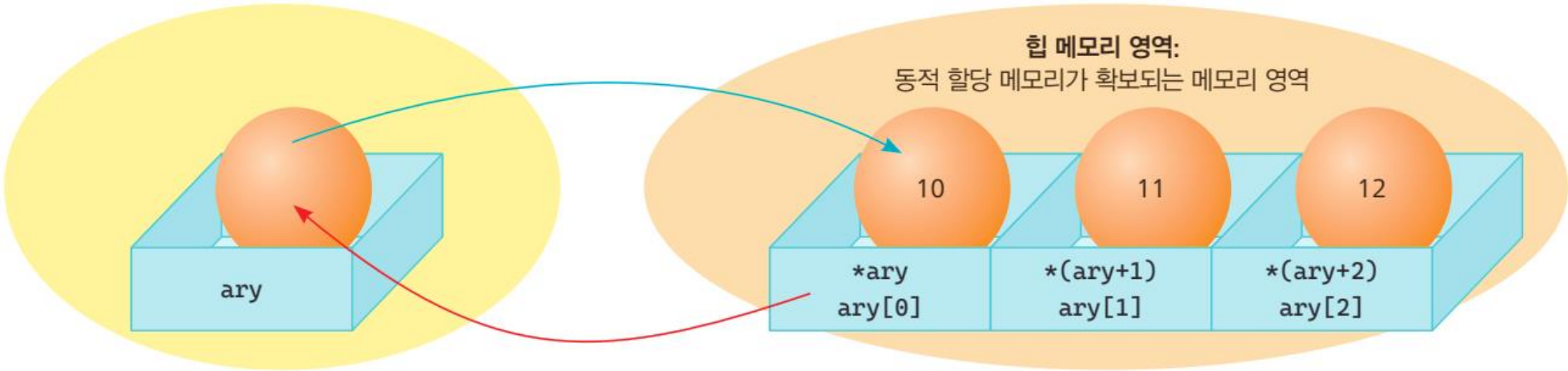
- 배열과 같이 동일한 메모리 공간 여러 개를 동적으로 확보
  - int형 배열을 확보하는 방법
    - 함수 malloc()이 반환하는 값은 포인터인 int *의 변수로 저장
    - malloc()의 인자
      - sizeof(int) * (확보하려는 배열의 원소의 개수)로 지정

함수 malloc()으로 정수형 저장공간 할당

- 변수 ary를 이용하여 다음과 같이 배열 형태와 같이 이용 가능

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

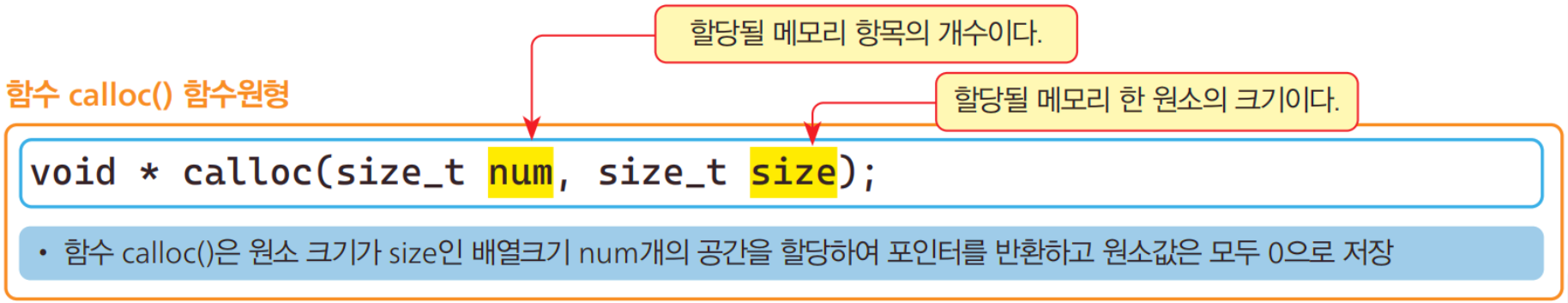
함수 malloc()에 의해 할당된 저장공간은 int형 3개이며 주소 값을 ary에 저장하고 있다. ary[i]로 각 원소를 참조할 수 있다.



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 703쪽

함수 calloc()

- 초기값을 자료형에 알맞게 0을 저장
  - 함수 malloc()
    - 메모리 공간을 확보하고 초기값을 저장하지 않으면 쓰레기값이 저장
- 함수 원형이 헤더 파일 stdlib.h에 정의
  - 앞의 인자: 할당되는 원소의 개수, 뒤의 인자: 한 원소의 크기



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 705쪽

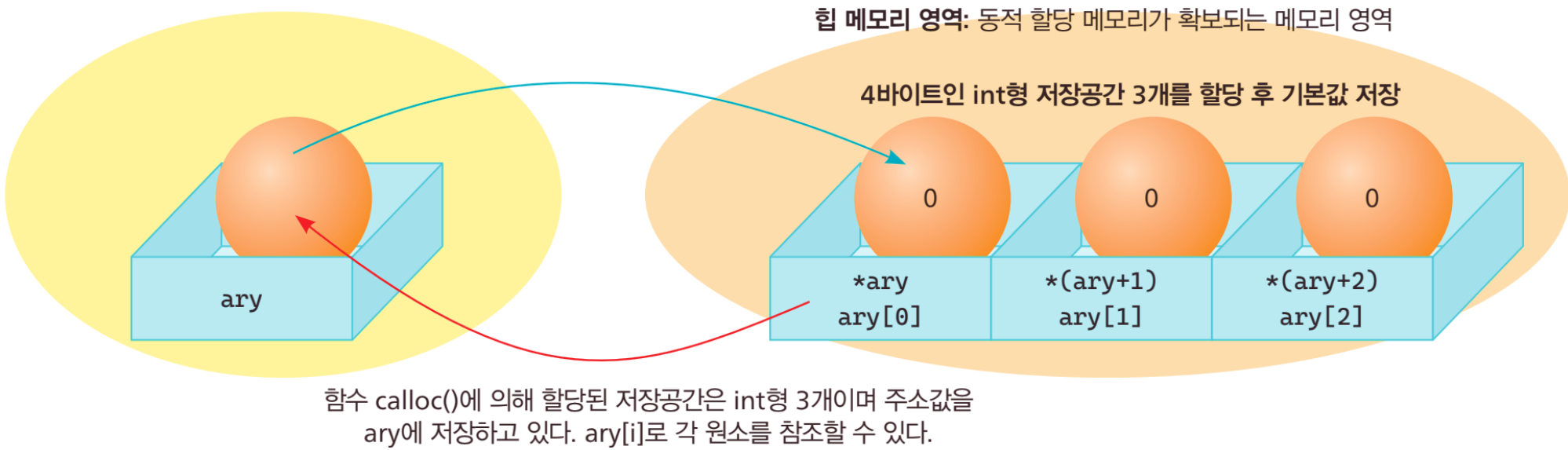
함수 calloc()의 저장공간 할당과 기본값 저장

함수 호출 calloc(3, sizeof(int))로 할당

- int형 원소 3개, 저장공간에 기본값 0

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.

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 705쪽

함수 realloc()

- 이미 확보한 저장공간을 새로운 크기로 변경
  - 함수 realloc()에 의하여 다시 확보하는 영역
    - 기존의 영역을 이용하여 그 저장 공간을 변경하는 것이 원칙
      - 새로운 영역을 다시 할당하여 이전의 값을 복사할 수도 있음
    - 성공적으로 메모리를 할당하면 변경된 저장공간의 시작 주소를 반환
      - 실패하면 NULL을 반환
    - 인자
      - 첫 인자: 변경할 저장공간의 주소
      - 두 번째 인자: 변경하고 싶은 저장공간의 총 크기

함수 realloc() 함수원형

- 함수 realloc( )에서 첫 번째 인자가 NULL
  - 지정된 크기만큼의 새로운 공간을 할당
    - 함수 malloc( )과 같은 기능을 수행

함수 realloc() 함수원형

할당되는 총 메모리 크기이다.

```
void * realloc(void *p, size_t size);
```

- 함수 realloc()은 이미 확보한 메모리 p를 다시 지정한 크기 size로 변경하는 함수이며, 이미 확보한 p가 NULL이면 malloc()과 같은 기능을 수행

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 706쪽

02

자기참조 구조체

자기참조 구조체(self reference struct)

- 멤버 중의 하나가 자기 자신의 구조체 포인터 변수를 갖는 구조체
- 구조체 selfref
  - 멤버로 int 형 n과 struct selfref * 형 next로 구성
    - 즉 멤버 next의 자료형은 지금 정의하고 있는 구조체의 포인터 형
  - 구조체는 자기 자신 포인터를 멤버로 사용 가능
    - 자기 자신은 멤버로 사용 불가능

```
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.

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 709쪽

연결 리스트(linked list)

자기 참조 구조체

- 동일 구조체의 표현을 여러 개 만들어 연결할 수 있는 기능

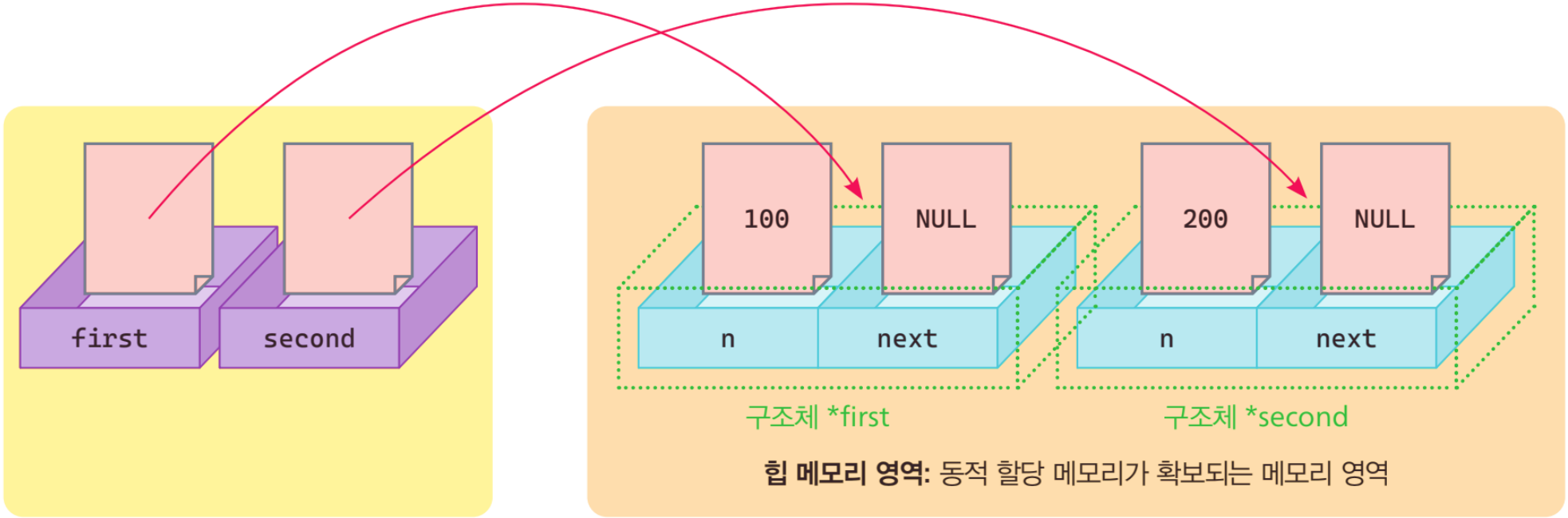
```
//❶ 우선 구조체 struct selfref를 하나의 자료형인 list 형으로 정의
typedef struct selfref list;

//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;
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 709쪽

구조체의 동적 할당

```
first = (list *)malloc(sizeof(list));
```

```
second = (list *)malloc(sizeof(list));
```




```
first->n = 100;
second->n = 200;
first->next = second->next = NULL;
```

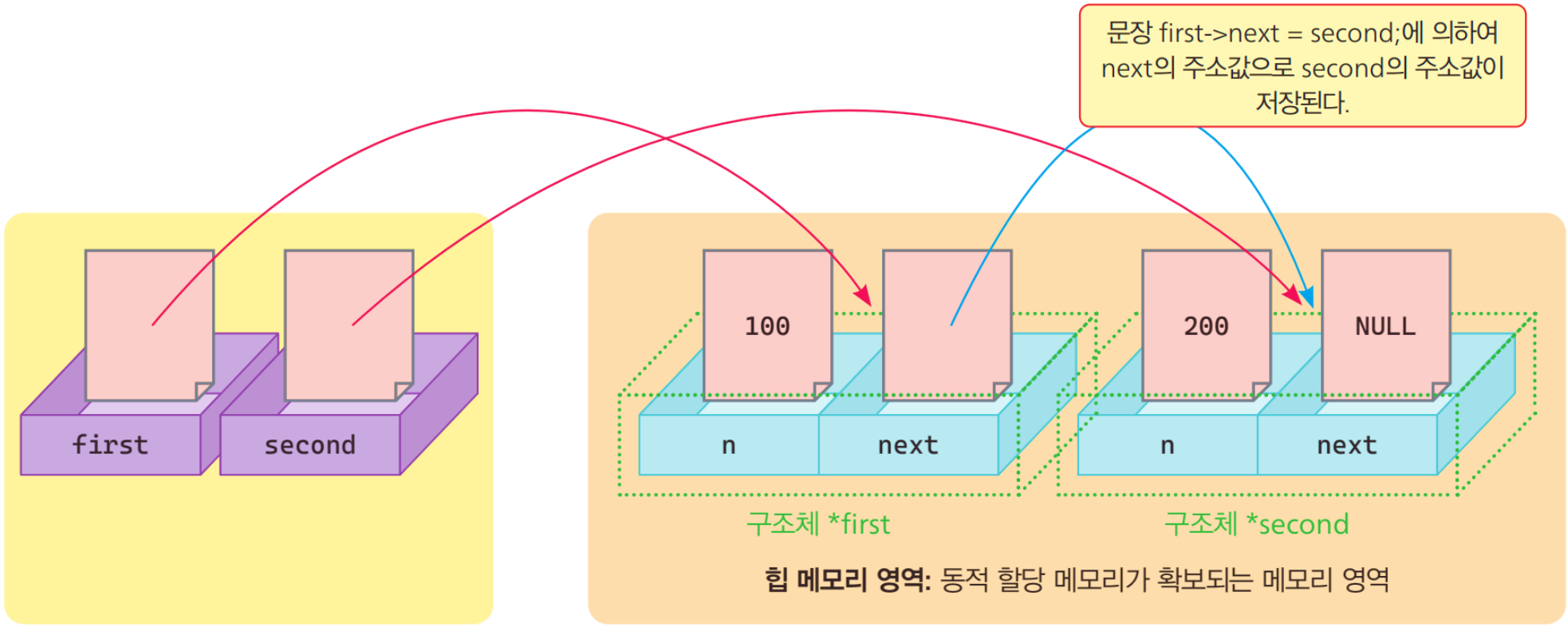

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 710쪽

구조체를 연결

- 만일 구조체 *first가 다음 *second 구조체를 가리키도록 하려면
  - 문장 first → next = second; 가 필요
    - 즉 구조체 *first의 멤버 next에 구조체 포인터 second의 내용인 주소값을 저장
  - 이제 first가 가리키는 구조체 멤버 next를 사용하여 다음 구조체를 연결

구조체의 주소값을 저장

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



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 710쪽

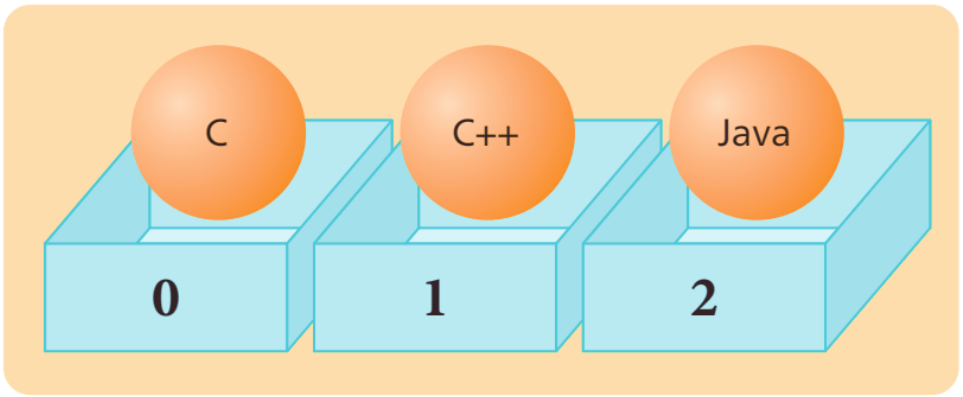
03

연결 리스트

배열

- 프로그램 언어와 개발 시기를 순차적으로 표현
  - C, C++, Java 순서로 나열 방법
    - 표
    - 배열

1	C
2	C++
3	Java



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 715쪽

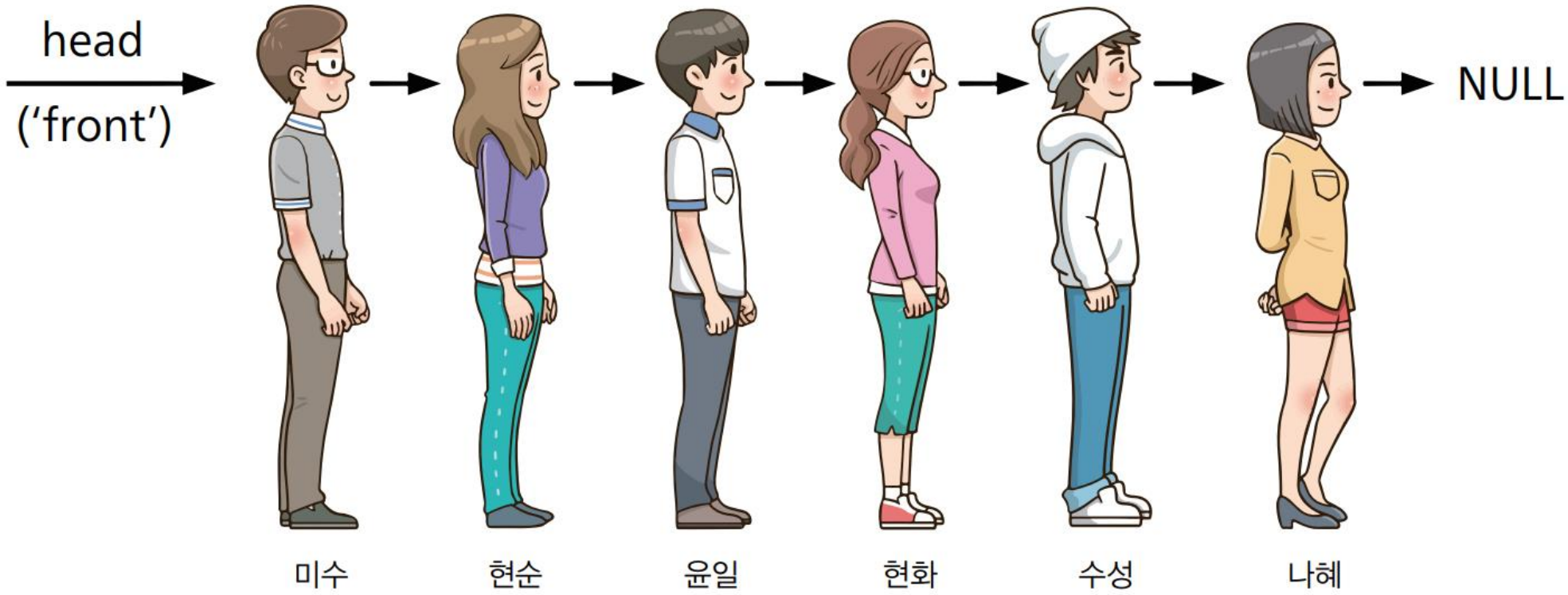
배열 장 단점

- 자료를 순차적으로 저장하기 가장 쉬운 방법
  - 배열이름과 첨자(index)를 사용
    - 원하는 원소를 직접 임의참조(random access) 가능
  - 단점
    - 컴파일 전에 배열의 크기가 이미 결정
      - 실행 중간에 배열크기 수정 불가능
    - 맨 앞이나 중간에 새로운 자료를 삽입
      - 삽입되는 자료 이후의 원소가 모두 이동
    - 중간에 하나 삭제하는 경우도 마찬가지 어려움

연결 리스트

- 배열과 함께 순차적 자료 표현에 적합한 구조
- 연결 리스트 예
  - 헤드(head)는 “미수”를 가리키고
    - “미수”는 다시 “현순”을 가리키고
    - 계속해서 “윤일”, “현화”, “수성”, “나혜”
    - 그리고 다시 나혜는 마지막
      - 가리키는 사람이 없는 것(NULL)과 같은 구조

연결 리스트 이해



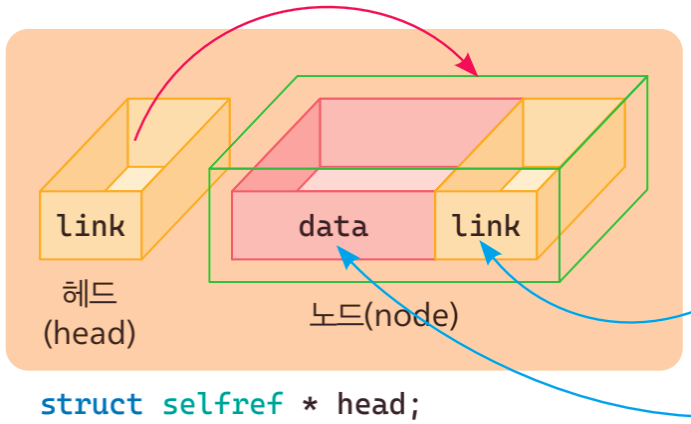
[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 716쪽

연결 리스트 개요

- 헤드에서 시작하여 가리키는 곳을 계속 따라가면 순차적 자료를 표현
- 원소인 노드(node)가 순차적으로 연결된 자료구조
  - 노드
    - 배열의 원소에 해당
    - 자료(data)와 링크(link)로 구성
    - 자기참조 구조체로 정의
- 첨자대신 링크(link)라는 포인터로 다음 노드를 가리키는 구조

노드의 표현

- 노드의 자료: 필요한 여러 변수의 조합으로 구성
  - 노드의 링크: 자기 구조체의 포인터로 구현
- 헤드(head)
  - 첫 번째 노드를 가리키는 포인터
- 테일(tail)
  - 마지막 노드를 가리키는 포인터



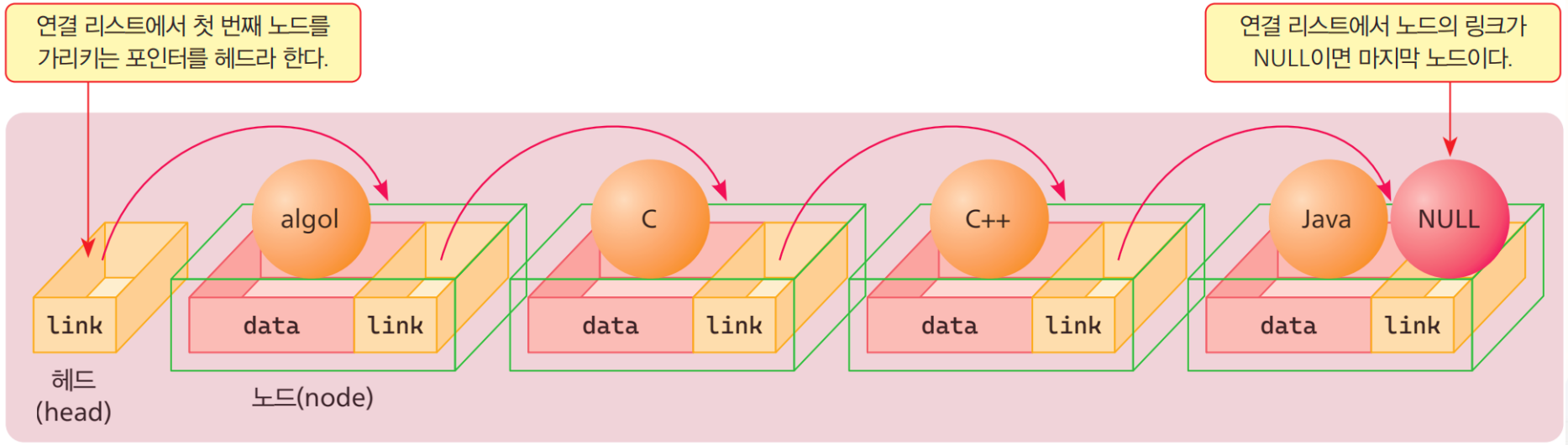
노드의 구현

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


[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 716쪽

연결 리스트 표현

- 헤드 포인터 노드에서 시작해서
  - 화살표(링크)를 따라 이동하면 자료를 순서대로 참조
- 연결 리스트에서 마지막 노드의 링크는 NULL로 저장
  - 만일 연결 리스트에 노드가 하나도 없다면 헤드는 NULL



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 717쪽

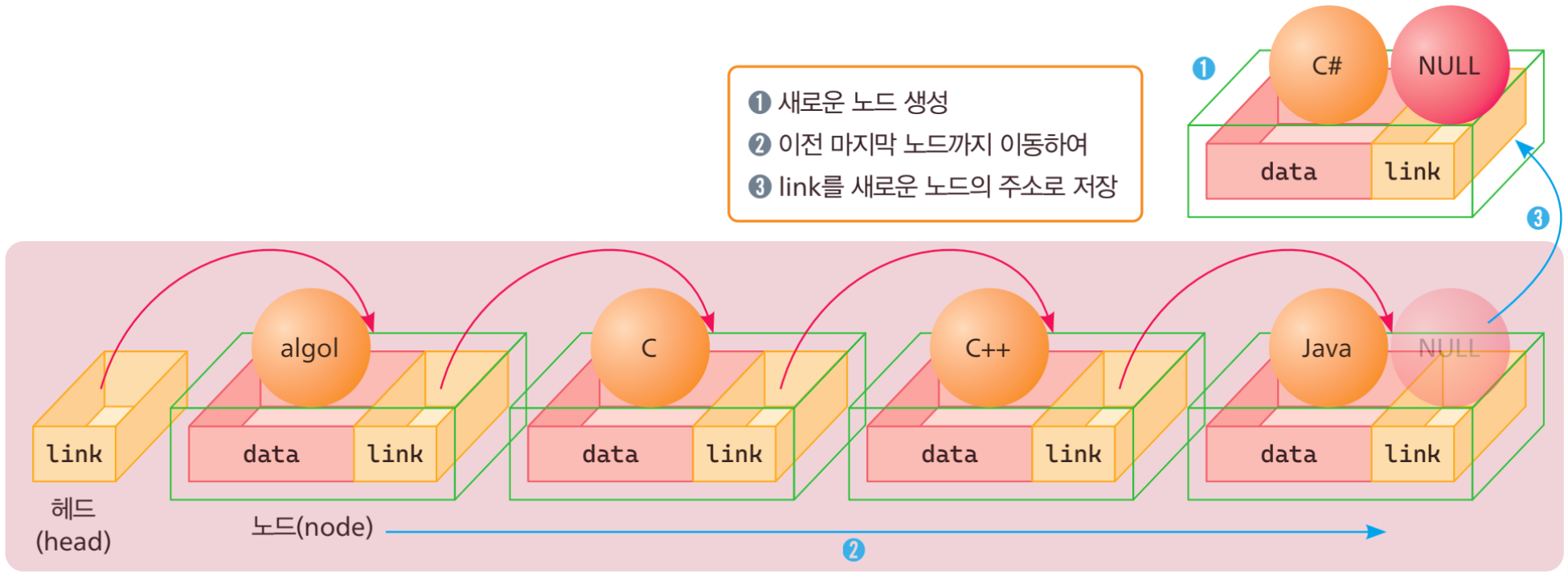
연결 리스트 노드 순회(node traversal)

- 연결 리스트에서 모든 노드를 순서대로 참조하는 방법
  - 헤드부터 계속 노드 링크의 포인터로 이동하면 가능
    - 링크가 NULL이면 마지막 노드
    - 각 노드의 자료를 참조, 원하면 출력도 가능

노드 추가 1/2

- 새로운 노드를 하나 생성, 연결 리스트의 마지막 노드로 추가
- "C#" 노드를 만들어 기존의 연결 리스트에 추가하는 방법
  1. 첫 번째로 추가할 노드를 먼저 생성한 후,
     자료를 저장하고 링크를 NULL로 저장
  2. 기존 연결 리스트를 순회하여 마지막으로 이동
  3. 마지막 노드의 링크를 새로 생성한 노드의 주소값으로 저장
     - 연결 리스트의 마지막 노드로 연결

노드 추가 2/2



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 718쪽

노드 삽입 1/2

- 연결 리스트 중간에 한 노드를 삽입하는 과정
  - 기존의 연결 리스트인 노드 “C”와 노드 “C++” 사이에 노드 “Objective-C”를 삽입하는 과정

❶ 가장 먼저 삽입 노드를 동적으로 생성하여 적당한 자료를 저장

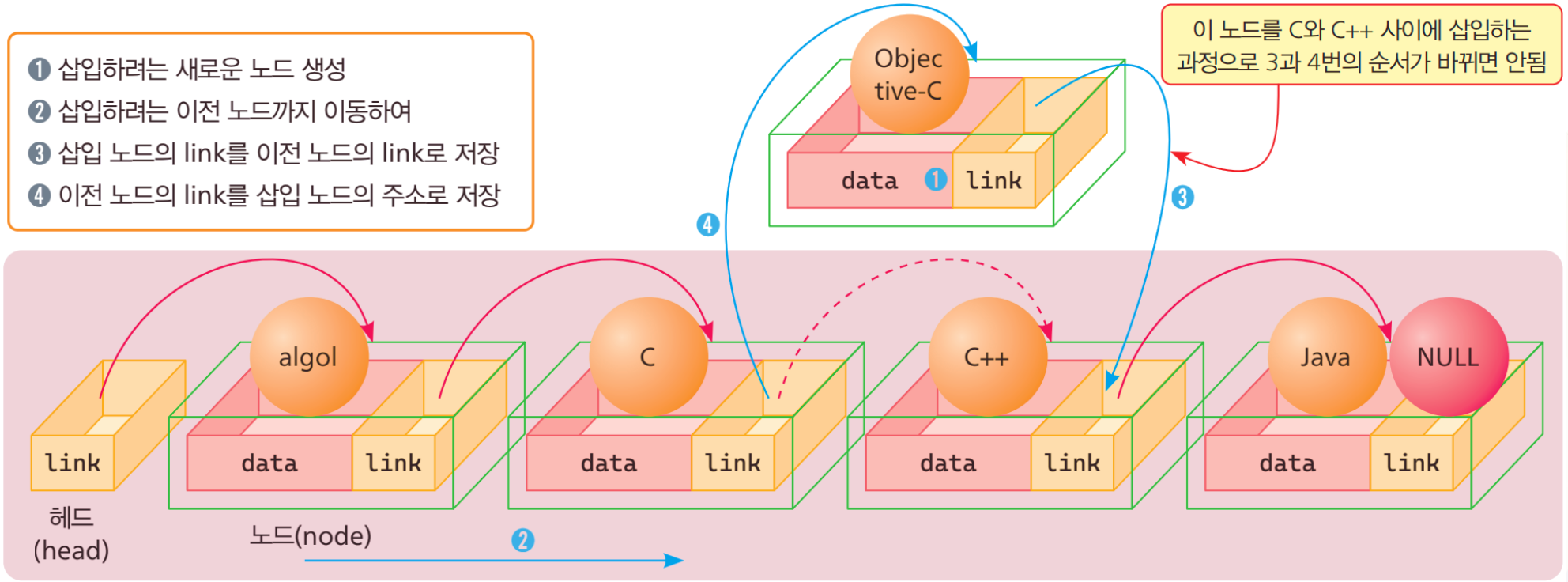
❷ 이제 삽입하려는 바로 이전 노드인 노드 “C”로 이동

❸ 삽입하는 “Objective-C” 노드의 링크에 노드 “C”의 링크를 저장

❹ 다음에는 노드 “C”의 링크를 새로 삽입하는 “Objective-C” 노드를 가리키도록 삽입하는 “Objective-C” 노드의 주소값을 저장

노드 삽입 2/2

❶ 삽입하려는 새로운 노드 생성
❷ 삽입하려는 이전 노드까지 이동하여
❸ 삽입 노드의 link를 이전 노드의 link로 저장
❹ 이전 노드의 link를 삽입 노드의 주소로 저장



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 719쪽

노드 삭제 1/2

- 노드 "C++"를 삭제하려면

❶ 가장 먼저 삭제하려는 노드 바로 이전 노드 "C"로 이동

- 또는 삭제 이전 노드를 바로 알려줄 수도 있음

❷ 삭제할 이전 노드의 링크인 삭제하려는 노드 "C++"의 주소를 저장

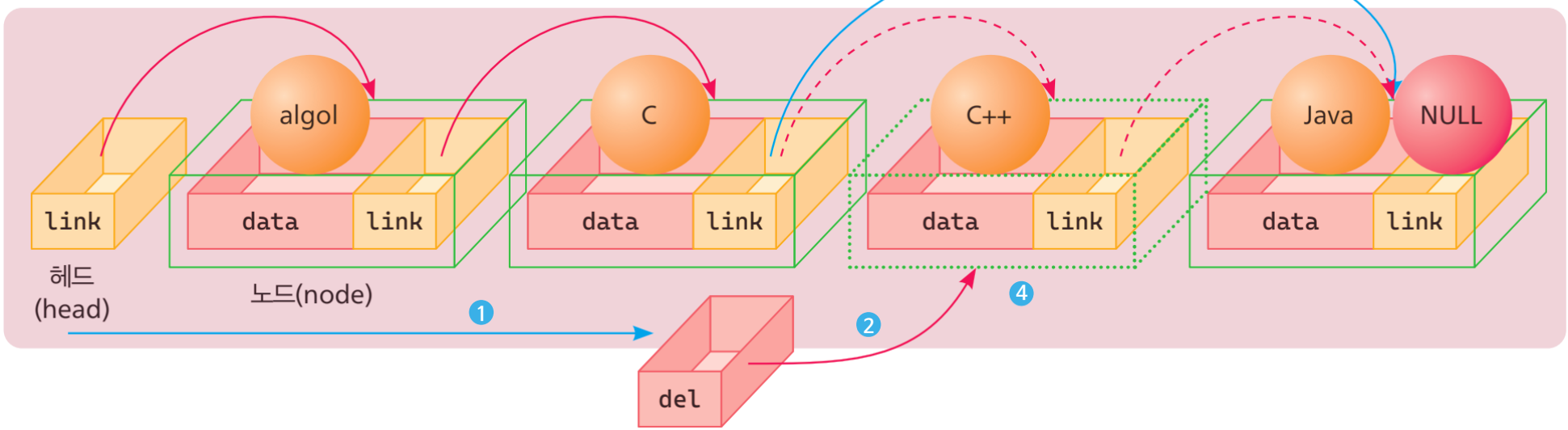
- 포인터 변수(del)를 마련

❸ 노드 "C"의 링크를 삭제하려는 노드 "C++"의 링크 노드 값으로 저장

❹ 이제 포인터 변수 del로 삭제 노드 "C++"를 메모리에서 제거

노드 삭제 2/2

❶ 삭제하려는 노드의 이전 노드로 이동
❷ 변수 del을 마련해 삭제하려는 노드의 주소를 저장
❸ 삭제 이전 노드의 link를 삭제 노드의 link로 저장
❹ 변수 del로 삭제 노드 "C++"를 메모리에서 제거



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 720쪽

정 리 하 기

정리하기

- 동적 메모리 할당이란
  프로그램 실행 중에 필요한 메모리를 할당하는 방법이다.
- 함수 malloc( )으로 힙(heap) 영역에 저장공간을 확보하며
  사용 후 함수 free( )로 저장공간을 해제한다.
- 자기참조 구초제는
  멤버 중의 하나가 자기 자신의 구조체 포인터 변수를 갖는
  구조체이다.

정리하기

- 헤드에서 시작하여 가리키는 곳을 계속 따라가는 순차적 자료구조를 연결 리스트(linked list)라 하며 자기참조 구초체로 연결 리스트를 생성한다.
- 연결 리스트의 각 항목을 노드(node)라 하며, 마지막 노드를 추가하고 중간에 노드를 삽입하고, 특정한 노드를 삭제한다.

다음시간 안내

15강

09~14강 요약