

1 | 구조체 개념

1 구조체 개념

- ▶ 구조체도 배열처럼 여러 개의 데이터를 그룹으로 묶어서 하나의 자료형으로 정의하고 사용하는 자료형
- ▶ 배열은 같은 자료형 만을 그룹으로 묶을 수 있지만, 구조체는 서로 다른 자료형을 그룹으로 묶을 수 있으므로 복잡한 자료 형태를 정의하는데 유용하게 사용됨

1 구조체 개념

▶ 자료를 체계적으로 관리하려면 일정한 단위 형식으로 구성해야 하는데, 이러한 단위 형식을 **레코드**라고 함

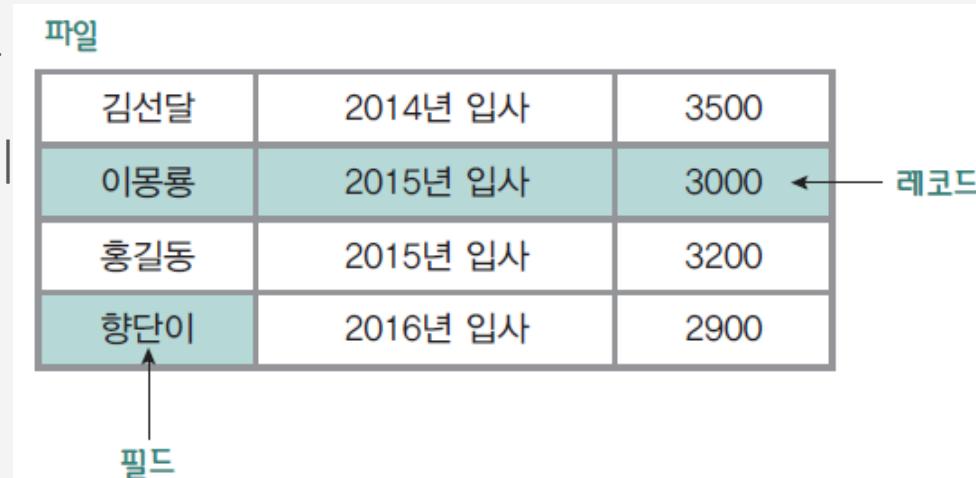
- 레코드를 구성하는 하위 항목을 **필드**라고 함
- 레코드가 여러 개 모이면 파일이 됨

파일

김선달	2014년 입사	3500
이몽룡	2015년 입사	3000
홍길동	2015년 입사	3200
향단이	2016년 입사	2900

레코드

필드



1 구조체 개념

- ▶ 여러 형태의 필드를 묶어 하나의 구성단위 역할을 하는 레코드는 구조체를 사용하여 정의할 수 있음
- ▶ 즉, 여러 자료형의 필드를 가지고 있는 레코드를 만들 때 구조체를 사용함

2 구조체 선언

구조체는 여러 자료형의 변수들을 그룹으로 묶어서
하나의 자료형으로 선언하여 사용함

▶ 구성 요소 : 구조체이름, 자료형, 데이터 항목으로 구성

- 구조체의 이름
: 구조체로 정의하는 새로운 자료형의 이름
- 항목
: 구조체를 구성하는 내부 변수들의 이름

2 구조체 선언

구조체는 여러 자료형의 변수들을 그룹으로 묶어서
하나의 자료형으로 선언하여 사용함

- ▶ 구성 요소 : 구조체이름, 자료형, 데이터 항목으로 구성
 - 구조체의 항목은 배열의 각 배열요소에 해당
 - 배열요소는 모두 같은 자료형으로 되어있으므로
배열요소에 대한 선언 없이 사용이 가능하지만,
구조체에서는 각 항목이 다른 자료형을 가질 수
있기 때문에 항목별로 자료형과 항목이름(변수이름)
을 선언해야 함

3 구조체형의 선언과 사용 형식

```
struct 구조체형이름 {  
    자료형 항목1;  
    자료형 항목2;  
    ...  
    자료형 항목n;  
};
```

(a) 선언 형식

```
struct 구조체이름 구조체변수이름;
```

(b) 사용 형식

3 구조체형의 선언과 사용 형식

▶ 구조체 사용 단계

- 1 구조체형 선언 : 내부 구조를 정의
- 2 구조체 변수 선언 : 구조체형에 따른 변수를 선언
- 3 구조체 변수의 사용 : 내부 항목에 데이터를 저장하고 사용



3 구조체형의 선언과 사용 형식

▶ 구조체 사용 예 : 직원관리 프로그램에서 사용할 수 있는 구조체

- 구조체 이름은 employee이고 데이터 항목은 3개
- name은 직원이름, year는 입사 연도, pay는 현재 연봉을 저장함

```
struct employee {  
    char name[10];  
    int year;  
    int pay;  
};
```

※ 출처 : C로 배우는 쉬운 자료구조 (개정3판), 이지영, 한빛미디어

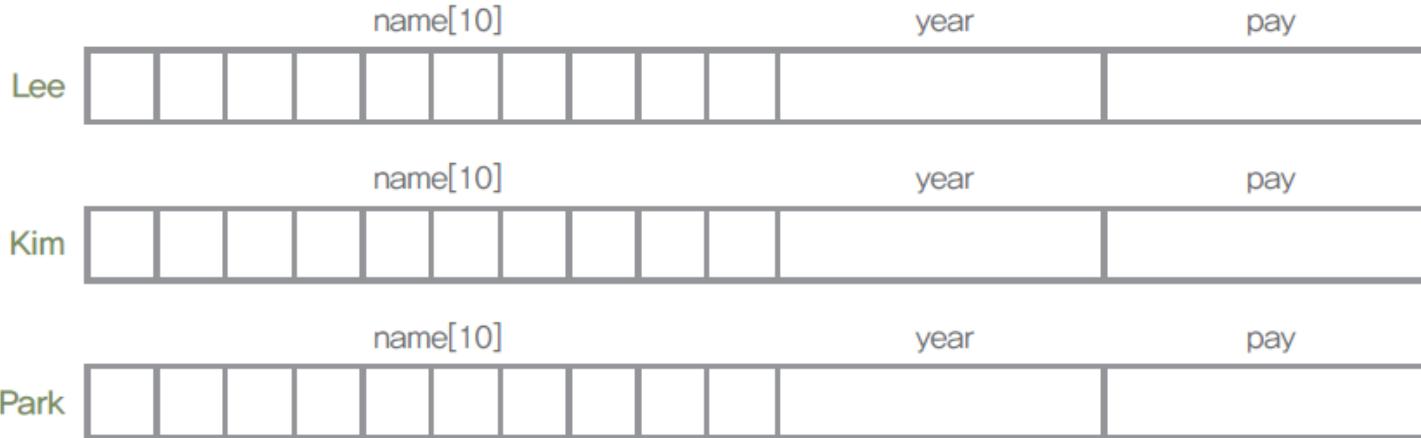


<구조체형 employee 선언 예>

3 구조체형의 선언과 사용 형식

▶ 구조체 사용 예 : 직원관리 프로그램에서 사용할 수 있는 구조체

```
struct employee Lee, Kim, Park;
```



<구조체형 employee에 대한 구조체 변수 선언 예>

* 출처 : C로 배우는 쉬운
자료구조(개정3판), 이지영,
한빛미디어

4

구조체 변수의 선언 방법

방법	예
구조체형을 선언한 후에 구조체 변수 선언	<pre>struct employee { char name[10]; int year; int pay; }; struct employee Lee;</pre>

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

4 구조체 변수의 선언 방법

방법	예
구조체형과 구조체 변수 를 연결하여 선언	<pre>struct employee { char name[10]; int year; int pay; } Lee;</pre>

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

4 구조체 변수의 선언 방법

방법	예
구조체형 이름을 생략하고 구조체 변수 이름만 선언	<pre>struct { char name[10]; int year; int pay; } Lee;</pre>

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

5 구조체의 변수의 초기화

- ▶ 일반 변수 초기화와 마찬가지로 구조체 변수 초기화 하려면 구조체 변수를 선언하면서 변수의 초기값 지정함
- ▶ 일반 변수는 값을 하나만 가지므로 초기값도 하나지만, 구조체는 내부 항목이 여러 개일 수 있으므로 내부 항목의 자료형과 개수를 순서에 맞추어 초기값 리스트로 지정하고 중괄호(()) 사용함
- ▶ 중괄호(())에 담긴 각 값은 순서대로 구조체 변수의 내부 데이터 항목에 대입됨

5 구조체의 변수의 초기화

▶ 초기화 예제

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

```
struct employee { // 구조체형 선언
    char name[10];
    int year;
    int pay;
};

struct employee Lee = { "Ann", 2015, 4200 }; // 구조체 변수의 초기화
```

	name[10]				year	pay	
Lee	A	n	n	\0		2015	4200

2 | 구조체 연산

1 구조체 변수의 데이터 항목 참조하는 방법

구조체 연산자 사용하여 구조체 변수에 있는
각 데이터 항목 참조함

▶ 구조체 연산자 종류

구조체 연산자	설명
점 연산자(.)	구조체 변수의 데이터 항목을 지정함
화살표 연산자(→)	구조체형 포인터에서 포인터가 가리키는 구조체 변수의 데이터 항목을 지정함

2 구조체 연산자

▶ 점 연산자 .

: 구조체 변수에 있는 데이터 항목을 개별적으로
지정할 때 사용함

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

```

❶ struct employee Lee;
❷ Lee.name = "Ann";
❸ Lee.year = 2015;
❹ Lee.pay = 4200;
```

	name[10]				year	pay
❶ Lee	A	n	n	\0	2015	4200
	❷				❸	❹

2 구조체 연산자

예제

점 연산자를 이용해
데이터 항목 참조하기

```
#include <stdio.h>
#include <string.h>
struct employee {
    char name[10];
    int year;
    int pay;
};
void main() {
    int i;
    struct employee Lee[4] = {
        { "이진호", 2014, 4200 },
        { "이한영", 2015, 3300 },
        { "이상원", 2015, 3500 },
        { "이상범", 2016, 2900 }    };
    for (i = 0; i<4; i++) {
        printf("\n 이름 : %s", Lee[i].name);
        printf("\n 입사 : %d", Lee[i].year);
        printf("\n 연봉 : %d \n", Lee[i].pay);
    }
    getchar();
}
```

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

2 | 구조체 연산

2 구조체 연산자

예제

점 연산자를 이용해
데이터 항목 참조하기

명령 프롬프트	
-	×
이름 : 이진호	↑
입사 : 2014	
연봉 : 4200	
이름 : 이한영	
입사 : 2015	
연봉 : 3300	
이름 : 이상원	
입사 : 2015	
연봉 : 3500	
이름 : 이상범	
입사 : 2016	
연봉 : 2900	▼

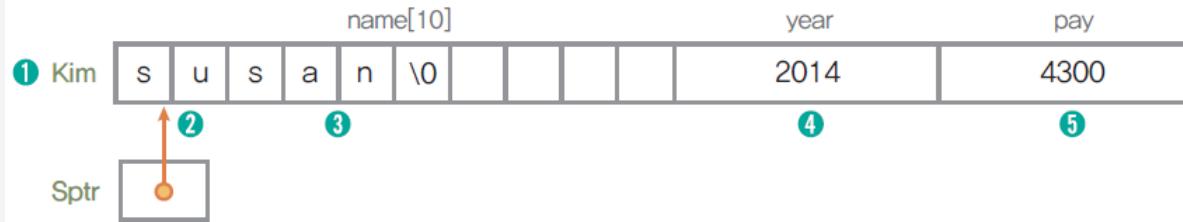
※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

2 구조체 연산자

▶ 화살표 연산자 →

: 구조체 포인터 변수에서 포인터가 가리키는 구조체 변수의 데이터항목을
지정하기 위해서 화살표 연산자 사용함

- ❶ struct employee Kim;
- ❷ struct employee *Sptr = &Kim;
- ❸ Sptr->name = "susan";
- ❹ Sptr->year = 2014;
- ❺ Sptr->pay = 4300;



※ 출처 : C로 배우는 쉬운 자료구조
(개정3판), 이지영, 한빛미디어

3 구조체 포인터를 이용한 구조체 변수의 데이터 항목 지정 방법

- ▶ (a)와 같은 화살표연산자를 (b)와 같이 포인터 참조 연산자(*)를 사용하여 표현할 수도 있음
- ▶ 점 연산자(.)가 참조연산자(*)보다 연산 우선순위가 높으므로 아래 (b)와 같이 참조연산자(*)를 먼저 처리 될 수 있도록 괄호를 사용해야 함 (그렇지 않으면 오류가 발생함)

```
Sptr->name = "susan";  
Sptr->year = 2014;  
Sptr->pay = 4300;
```

(a) 구조체 포인터의 화살표 연산자 사용



```
(*Sptr).name = "susan";  
(*Sptr).year = 2014;  
(*Sptr).pay = 4300;
```

(b) 구조체 포인터의 참조 연산자 사용

※ 출처 : C로 배우는 쉬운
자료구조(개정3판), 이지영,
한빛미디어

3 구조체 포인터를 이용한 구조체 변수의 데이터 항목 지정 방법

예제

화살표 연산자를 이용해
데이터 항목 참조하기

```
#include <stdio.h>
#include <string.h>

struct employee {
    char name[10];
    int year;
    int pay;
};

void main() {
    struct employee Lee;
    struct employee *Sptr = &Lee;
    strcpy(Sptr->name, "이순신");
    Sptr->year = 2015;
    Sptr->pay = 5900;

    printf("\n 이름 : %s", Sptr->name);
    printf("\n 입사 : %d", Sptr->year);
    printf("\n 연봉 : %d", Sptr->pay);

    getchar();
}
```

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

3 구조체 포인터를 이용한 구조체 변수의 데이터 항목 지정 방법

예제

화살표 연산자를 이용해
데이터 항목 참조하기



※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

4 구조체 연산 활용

- ▶ 구조체에서는 데이터 항목 참조 연산, 구조체
변수복사, 구조체 변수의 주소 구하기 연산 등을 사용
할 수 있음

4 구조체 연산 활용

▶ 데이터 항목 참조 연산

- 점연산자와 화살표연산자 이용하여 구조체 데이터 항목 개별적으로 참조함

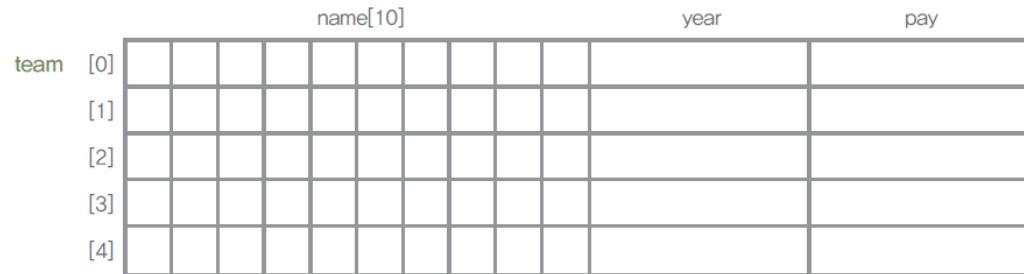
```
struct employee Lee;
struct employee *Sptr;
Sptr = &Lee;
Lee.year = 2015;
Sptr->pay = 3000;
Sptr->name = "Ann";
```

4 구조체 연산 활용

▶ 구조체 변수 복사 연산

- 같은 구조체에 있는 구조체 변수들끼리 내용을 한 번에 복사하는 연산

```
struct employee Lee, Kim, team[5];
```



```
Lee = Kim;
Lee = team[2];
team[2] = team[3];
```

4 구조체 연산 활용

▶ 구조체 변수의 주소 구하기 연산

- 포인터의 주소연산자 사용하여 구조체 변수의 주소 구하거나, 구조체 변수가 배열인 경우에는 배열의 특성에 따라 구조체 배열 변수의 이름에서 주소를 구할 수 있음

```
struct employee Lee, team[5];
struct employee *Sptr1, *Sptr2;
```

```
Sptr1 = &Lee;
Sptr2 = team;
```

3 | 재귀호출

1 재귀호출(순환호출)

- ▶ 함수가 자기 자신을 호출하여 순환 수행되는 것으로 순환호출 또는 recursion 이라고 함
- ▶ 함수 실행 특성에 따라 일반적인 호출방식보다 재귀호출방식을 사용하여 함수를 만들면 프로그램의 크기를 줄이고 간단하게 작성 가능함
- ▶ 내가 나를 호출하는 것이므로 내 현재 작업을 처리하기 위해 같은 유형의 하위 작업이 필요함
- ▶ 전체 문제를 한 번에 해결하기보다 같은 유형의 하위 작업으로 분할하여 작은 문제부터 해결하는 방법이 효율적인 경우 사용함

* 하위 작업이란?

: 현재 수행 중인 작업의 하위 단계 즉, 좀 더 작은 단위 작업을 말함

1 재귀호출(순환호출)

▶ 베이스케이스(Base Case)

- 한 번에 해결할 수 없는 현재 작업을 한 단계 작게 분할한 하위작업에 대해 재귀 호출하는 과정을 반복하다 보면, 한 번에 해결할 수 있을 정도로 분할된 작업 단위가 충분히 작아지는 단계를 만나게 되는데 이 단계를 베이스케이스라고 함
- 베이스케이스에서 구한 답을 재귀 호출자에 반환하는 과정이 재귀호출의 역순으로 반복되어 결국 처음의 문제 대한 답을 구하게 되는 것이 재귀호출을 이용한 문제 해결의 원리임

3 | 재귀호출

1 재귀호출(순환호출)

▶ 재귀호출의 개념 예



※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

1 재귀호출(순환호출)

▶ 재귀호출의 개념 예

“마트료시카”라는 러시아 인형을 떠올려보자

- 인형을 열면 그 안에 다른 작은 인형이 들어있음
- 처음에는 인형 하나로 보이지만 열어보면 모양은 같지만 크기가 조금씩 다른 인형이 포개져 있음
- 이런 원리의 “마트료시카” 인형을 만든다면 전체를 한번에 만들기보다는 큰 인형, 작은 인형, 더 작은 인형으로 분할해서 작은 것부터 단계적으로 하나씩 만들어 완성해야 함

한번에 완성할 수 있는
제일 작은 인형
→ 베이스케이스



※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

2 팩토리얼 함수

대표적인 재귀호출 함수는 **팩토리얼(Factorial)**임

- ▶ n에 대한 팩토리얼 함수는 1부터 n까지 모든 자연수를 곱하는 연산

2 팩토리얼 함수

▶ $n! = n \times (n-1)!$ 을 구하려면 $(n-1)!$ 값을 알아야 함

- 그래서 $(n-1)! = (n-1) \times (n-2)!$ 을 구해야 하고, $(n-2)!$ 값이 필요하므로 그래서 $(n-2)! = (n-2) \times (n-3)!$ 을 구해야 함
- 그러면 다시 $(n-3)!$ 값이 필요함
- 이와 같이 현재 필요한 팩토리얼을 구하려면 하위 값에 대한 팩토리얼을 구하는 작업을 1! (베이스케이스)까지 반복해야 함
- 1!값을 알게 되면 그 값을 이용하여 바로 상위 값을 구하고, 구한 값을 이용하여 다시 그 상위 값을 구하기를 반복하여 결과적으로 $n!$ 값을 구함

이런 경우와 같이
재귀호출을 사용하여
함수를 작성할 수 있음

$$n! = n \times (n-1)!$$

$$(n-1)! = (n-1) \times (n-2)!$$

$$(n-2)! = (n-2) \times (n-3)!$$

...

$$2! = 2 \times 1!$$

$$1! = 1 \text{ (베이스케이스)}$$

2 팩토리얼 함수

▶ 팩토리얼(Factorial) 함수 예

```
long int fact(int n) {  
    if (n <= 1)  
        return (1);  
    else  
        return (n * fact(n - 1));  
}
```

(a) 재귀호출을 이용한 팩토리얼 함수

```
long int fact(int n) {  
    int i, f = 1;  
    if (n <= 1)  
        return (1);  
    else {  
        for (i = n; i >= 0; i++)  
            f = f * i;  
        return f;  
    }  
}
```

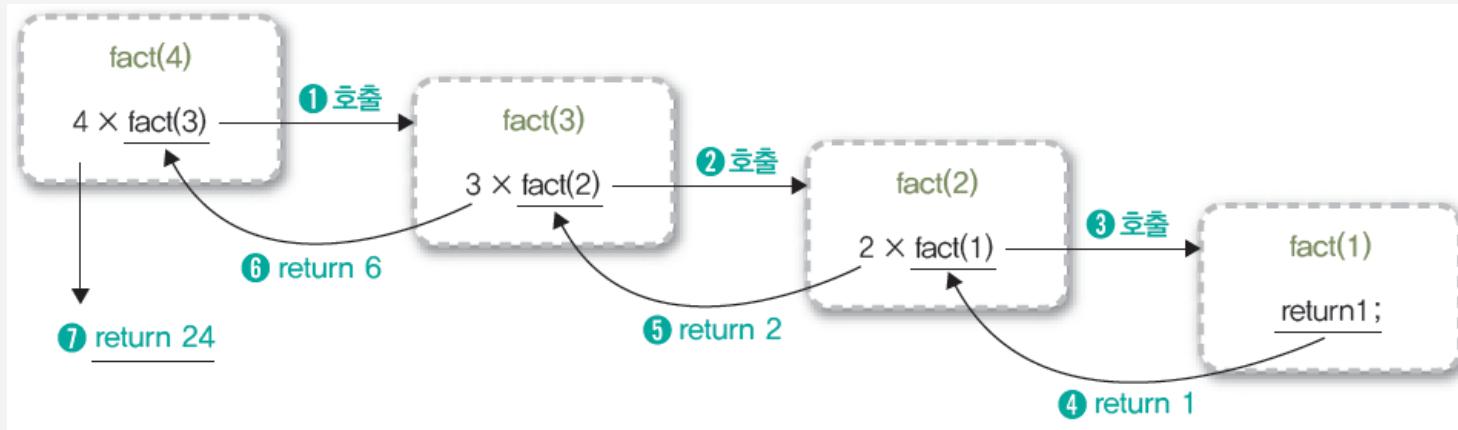
(b) 반복문을 이용한 팩토리얼 함수

※ 출처 : C로 배우는 쉬운
자료구조(개정3판), 이지영,
한빛미디어

2 팩토리얼 함수

▶ Factorial 함수에서 n=4일 때의 실행

$$\begin{aligned} 4! &= \text{fact}(4) = 4 * \text{fact}(3) = 4 * 3 * \text{fact}(2) = 4 * 3 * 2 * \text{fact}(1) = 4 * 3 * 2 * 1 \\ 4! &= 24 \end{aligned}$$



※ 출처 : C로 배우는 쉬운 자료구조 (개정3판), 이지영, 한빛미디어

2 팩토리얼 함수

예제

재귀호출을 이용해
팩토리얼 값 구하기

```
#include <stdio.h>
long int fact(int);
void main() {
    int n, result;
    printf("\n 정수를 입력하세요 : ");
    scanf("%d", &n);
    result = fact(n);
    printf("\n\n %d의 팩토리얼 값은 %ld입니다.\n", n, result);
    getchar(); getchar();
}

long int fact(int n) {
    int value;
    if (n <= 1) {
        printf("\n fact(1) 함수 호출!");
        printf("\n fact(1) 값 1 반환!!!");
        return 1;
    }
    else {
        printf("\n fact(%d) 함수 호출!", n);
        value = (n * fact(n - 1));
        printf("\n fact(%d) 값 %ld 반환!!!", n, value);
        return value;
    }
}
```

3 | 재귀호출

2 팩토리얼 함수

예제

재귀호출을 이용해
팩토리얼 값 구하기

04. 명령 프롬프트

정수를 입력하세요 : 4

```
fact(4) 함수 호출!  
fact(3) 함수 호출!  
fact(2) 함수 호출!  
fact(1) 함수 호출!  
fact(1) 값 1 반환!  
fact(2) 값 2 반환!  
fact(3) 값 6 반환!  
fact(4) 값 24 반환!
```

4의 팩토리얼 값은 24입니다.

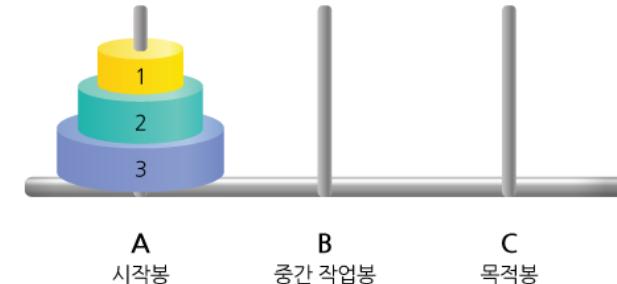
3 하노이 탑

- ▶ 팩토리얼 함수는 단순한 반복 구조라 재귀 호출 방식을 사용하지 않고 반복문으로 구현할 수 있음
- ▶ 하지만 복잡한 재귀 구조 문제는 재귀호출을 해야 해결할 수 있음
- ▶ 대표적인 예가 하노이 탑퍼즐임

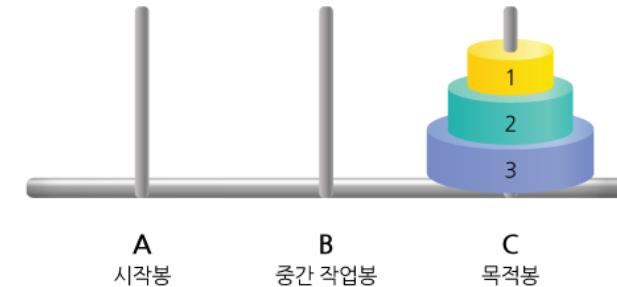
3 하노이 탑

▶ 개념 예

- 세 개의 기둥과 기둥에 꽂을 수 있는 크기가 다양한 원반이 n 개 있음
- (a)의 A처럼 모든 원반이 크기 순서대로 쌓여 있는 상태에서 게임을 시작하므로 A가 시작봉이 됨
- (b)의 C처럼 모든 원반이 크기 순서대로 쌓이면 게임이 끝나므로 C가 목적봉이 됨



ⓐ 하노이 탑 퍼즐 시작 상태

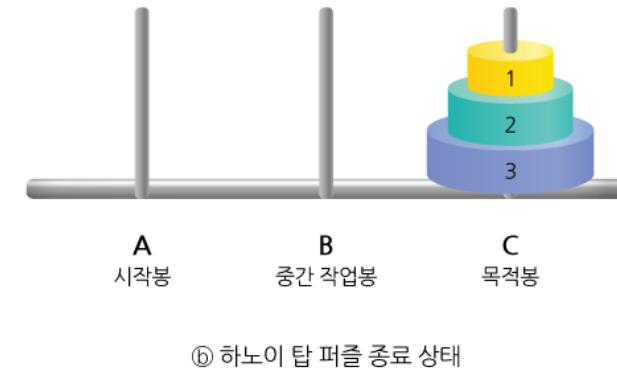
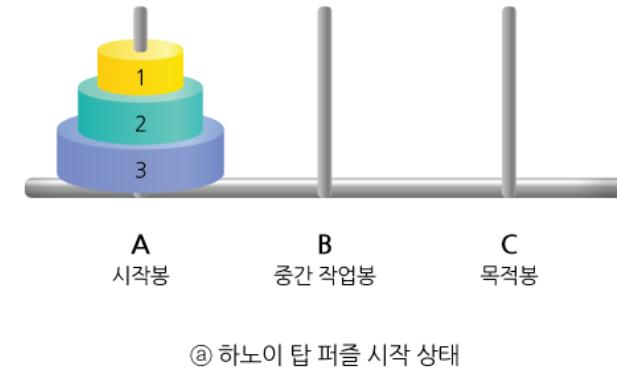


ⓑ 하노이 탑 퍼즐 종료 상태

3 하노이 탑

▶ 개념 예

- 한번에 하나의 원반만 옮길 수 있고, 큰 원반을 작은 원반 위로 옮길 수 없음
- 따라서 중간 작업 단계를 거쳐야 하므로 중간단계인 B를 중간 작업봉이라 함
- `hanoi(3, 'A', 'B', 'C')`는 원반 세 개를 시작봉 A에서 작업봉 B로 이동시키면서 목적봉 C로 이동하는 함수가 됨



3 하노이 탑

- ▶ 원반 세 개 있는 하노이 탑 작업 과정(1)
: hanoi(3, 'A', 'B', 'C')

◎ 초기 상태

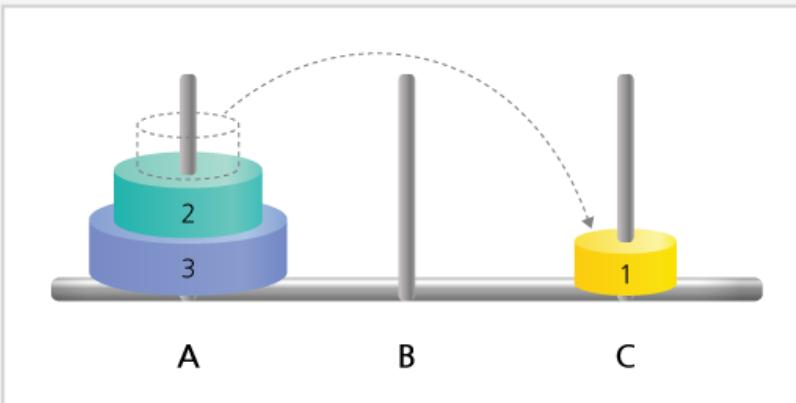
hanoi(3, 'A', 'B', 'C')

- 원반 개수 3개,
- 시작봉 : A
- 중간 작업봉 : B
- 목적봉 : C

3 하노이 탑

- ▶ 원반 세 개 있는 하노이 탑 작업 과정(1)
: hanoi(3, 'A', 'B', 'C')

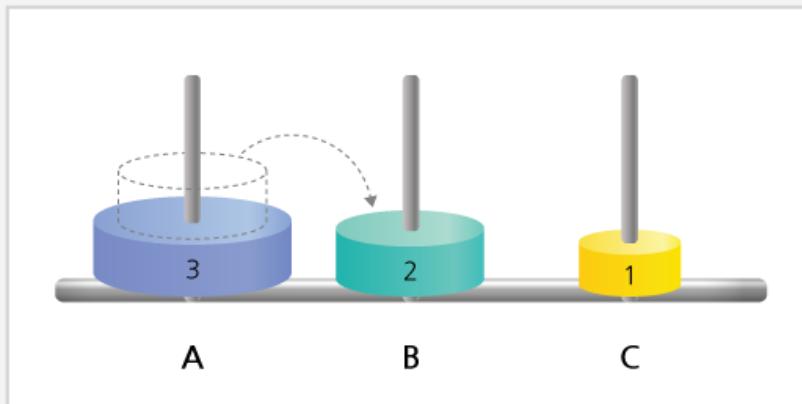
- ① 시작봉 A에서 맨 위에 있는 원반 1을 C로 옮김



3 하노이 탑

▶ 원반 세 개 있는 하노이 탑 작업 과정(1)
: hanoi(3, 'A', 'B', 'C')

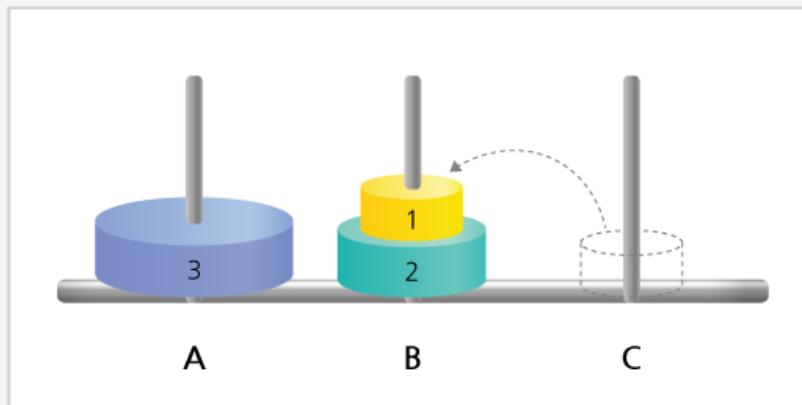
② A에서 맨 위에 있는 원반 2을 B로 옮김



3 하노이 탑

- ▶ 원반 세 개 있는 하노이 탑 작업 과정(1)
: hanoi(3, 'A', 'B', 'C')

- ③ C에서 있는 원반 1을 B로 옮김



1~3단계

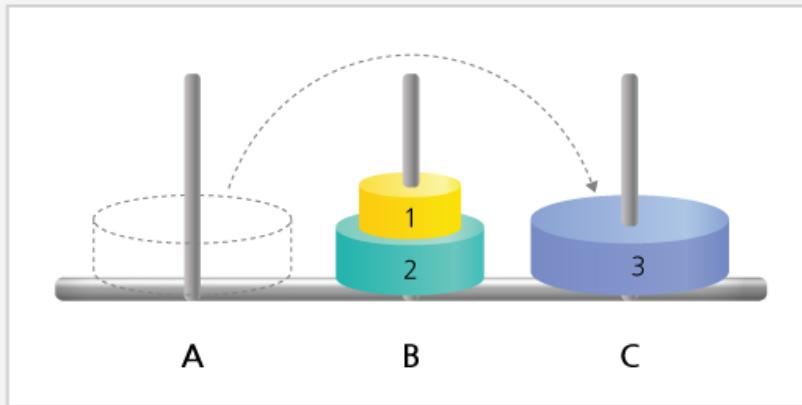
: A에 있던 원반 두 개를 C를
이용하여 B로 옮기는 작업
→ hanoi(2, 'A', 'C', 'B')

* 시작봉 : A, 중간 작업봉 : C, 목적봉 : B

3 하노이 탑

- ▶ 원반 세 개 있는 하노이 탑 작업 과정(1)
: hanoi(3, 'A', 'B', 'C')

- ④ 시작봉 A에 있는 마지막 원반 3을 목적봉 C로 옮김



4단계

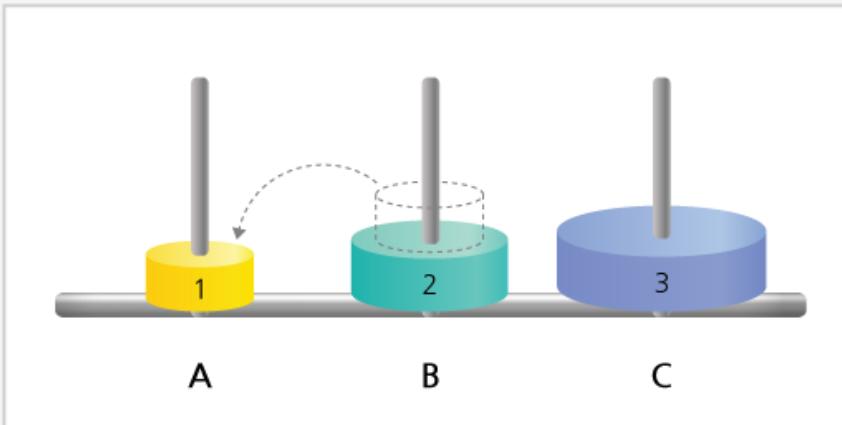
: A에 있던 원반 한 개를
C로 옮기는 작업
→ hanoi(1, 'A', 'B', 'C')

* 원반 개수가 1개 이므로 베이스케이스

3 하노이 탑

▶ 원반 세 개 있는 하노이 탑 작업 과정(2)
: hanoi(3, 'A','B','C')

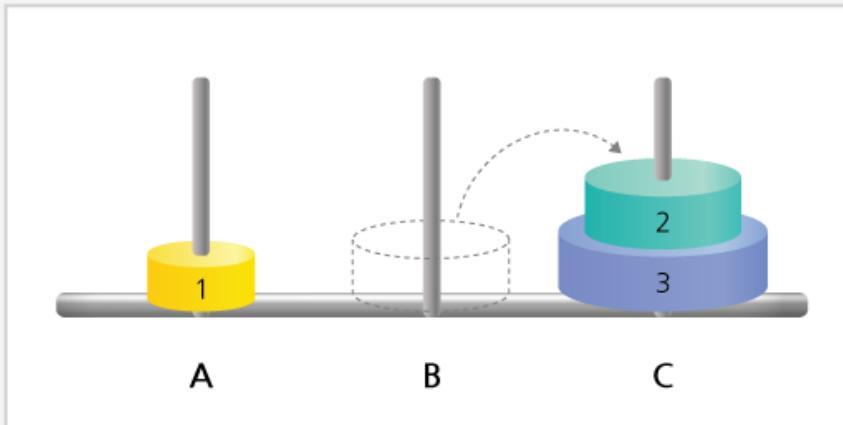
⑤ B에 있는 원반 1을 A로 옮김



3 하노이 탑

▶ 원반 세 개 있는 하노이 탑 작업 과정(2)
: hanoi(3, 'A', 'B', 'C')

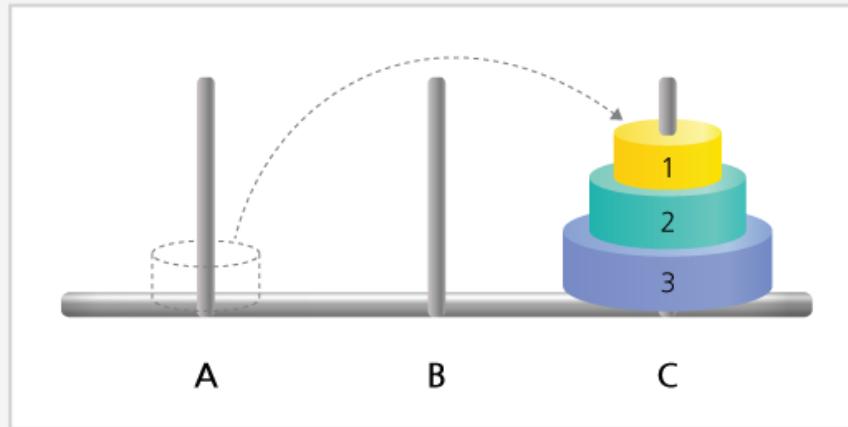
⑥ B에 있는 원반 2을 C로 옮김



3 하노이 탑

- ▶ 원반 세 개 있는 하노이 탑 작업 과정(2)
: hanoi(3, 'A', 'B', 'C')

⑦ A에 있는 원반 1을 C로 옮김



5~7단계
: B에 있던 원반 두 개를
C로 옮기는 작업
→ hanoi(2, 'B', 'A', 'C')

* 시작봉 : B, 중간 작업봉 : A, 목적봉 : C

3 하노이 탑

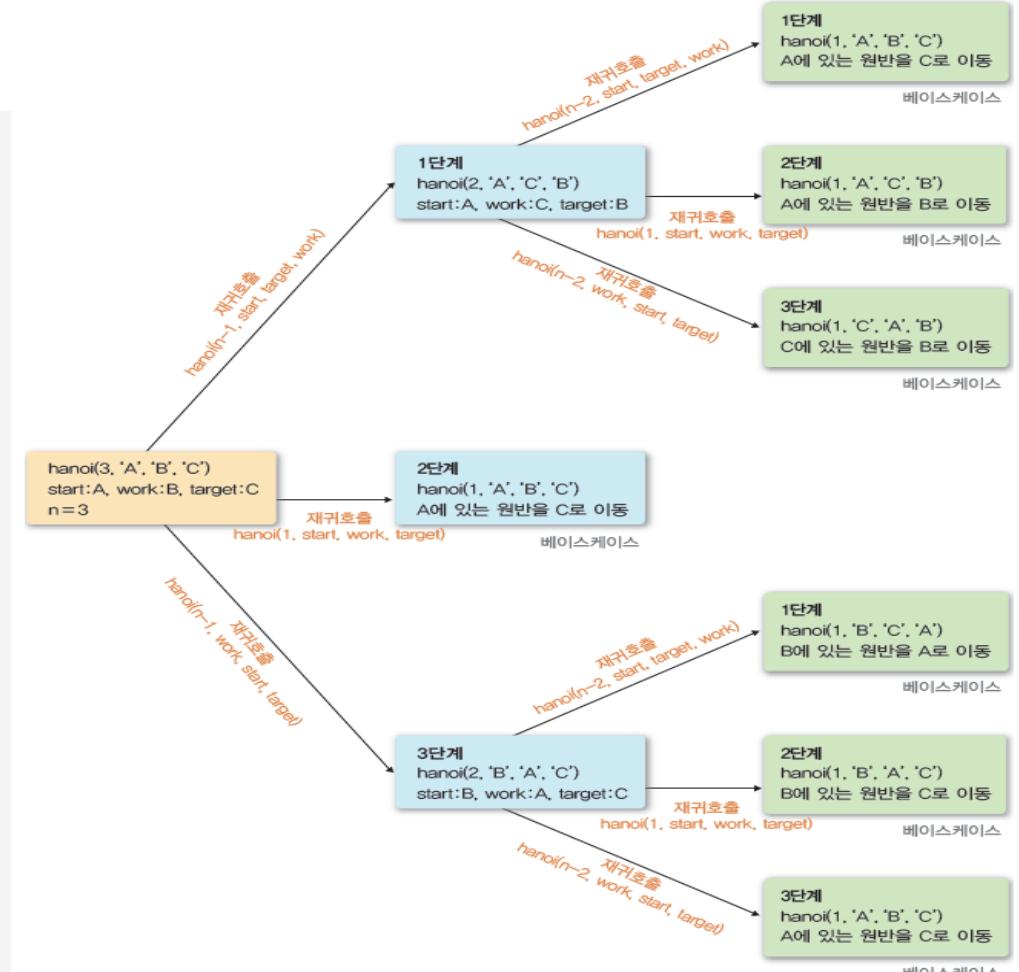
▶ 하노이 탑 퍼즐의 일반화 개념

- 1단계(①~③) → **hanoi(n-1, start, target, work)**
: 시작봉(start)에 있는 원본 1~원반 n-1을 목적봉(target)을 이용하여 중간 작업봉(work)으로 옮김
- 2단계(④) → **hanoi(1, start, work, target)**
: 시작봉에 있는 마지막 원반 n을 목적봉으로 옮김
- 3단계(⑤~⑦) → **hanoi(n-1, work, start, target)**
: 중간 작업봉에 있는 원반 1~원반 n-1을 시작봉을 이용하여 목적봉으로 옮김

3 | 재귀호출

3 하노이 탑

▶ 원반 세 개일 때 하노이 탑 과정을 재귀호출을 이용해 해결하는 과정



* 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

3 하노이 탑

예제

재귀호출을 이용해
하노이 탑 퍼즐 풀기

```
void honoi(int n, int start, int work, int target);  
  
void main() {  
    honoi(3, 'A', 'B', 'C');  
    getchar();  
}  
  
void honoi(int n, int start, int work, int target){  
    if(n==1) printf("%c에서 원반 %d를 %c로 옮김 \n", start, n, target);  
    else {  
        honoi(n-1, start, target, work);  
        printf("%c에서 원반 %d를 %c로 옮김 \n", start, n, target);  
        honoi(n-1, work, start, target);  
    }  
}
```

※ 출처 : C로 배우는 쉬운 자료구조
(개정3판), 이지영, 한빛미디어



3 하노이 탑

예제

재귀호출을 이용해
하노이 탑 퍼즐 풀기

The screenshot shows a Windows Command Prompt window with a red title bar containing the text "명령 프롬프트". The window displays the following text in Korean:

```
A에서 원반 1를(을) C로 옮김
A에서 원반 2를(을) B로 옮김
C에서 원반 1를(을) B로 옮김
A에서 원반 3를(을) C로 옮김
B에서 원반 1를(을) A로 옮김
B에서 원반 2를(을) C로 옮김
A에서 원반 1를(을) C로 옮김
```

※ 출처 : C로 배우는 쉬운 자료구조
(개정3판), 이지영, 한빛미디어