COMP319 Algorithms
Lecture 2

C programming review, part 2

Instructor: Gil-Jin Jang **장길진**

Original slides:

한빛아카데미(주)

C로 배우는 쉬운 자료구조, 개정 3판, 이지영

Hanbit Academy

Easy data structure in C, 3rd edition, Jiyoung Lee

C로 배우는 쉬운 자료구조, 개정 3판

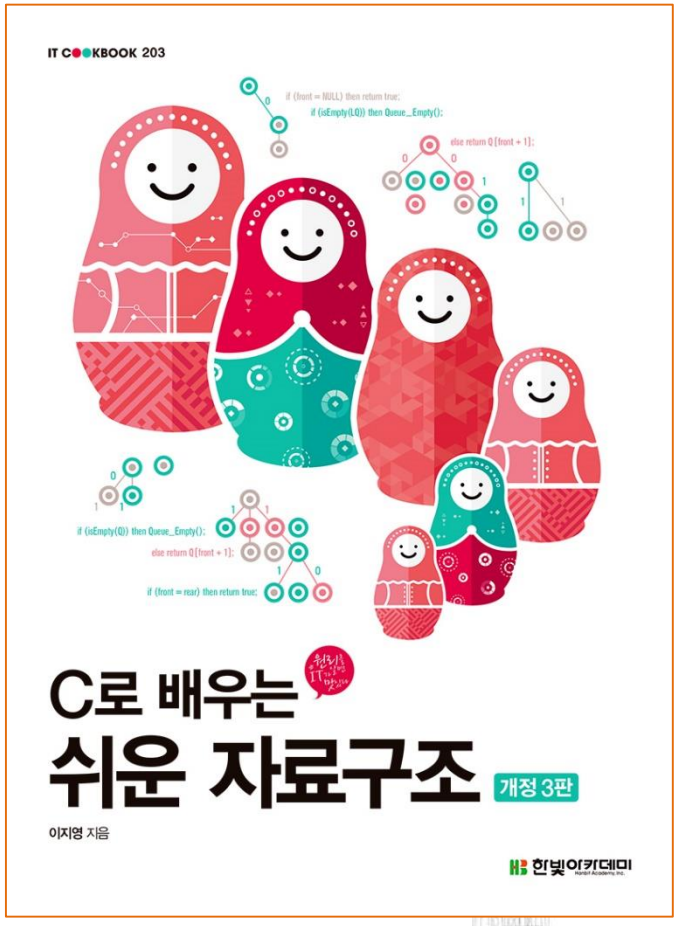
[강의교안 이용 안내]

Textbook

원리를 알면 IT가 맛있다
IT COOKBOOK

- 도서명 : C로 배우는 쉬운 자료구조, 개정 3판
- ISBN : 79-11-5664-269-5 93000
- 저자 : 이지영
- 출판사 : 한빛아카데미(주)
- 페이지 / 정가 : 596p / 27,000원
- 예제 소스 :

  http://www.hanbit.co.kr/exam/4269



2 자료구조 구현을 위한 C 프로그래밍 기법

IT CookBook, C로 배우는 쉬운 자료구조(개정 3판)

한빛아카데미 Hanbit Academy, Inc.

Contents

- **Learning programming techniques to implement various data structures (algorithms) in C programs**

- ***Arrays***
- ***Pointers***
- **Structure**
- **Recursive functions**

Structure

3. Structure: Definition

❖ Definition of structure

- Similar to arrays, grouping a number of data items, and make the group as a single type
  - Arrays can group only the same data type, but structures can group different data types, so it is useful for defining complex data types.
- Structures are used to create records that contain fields of multiple data types
  - Fields, records, files

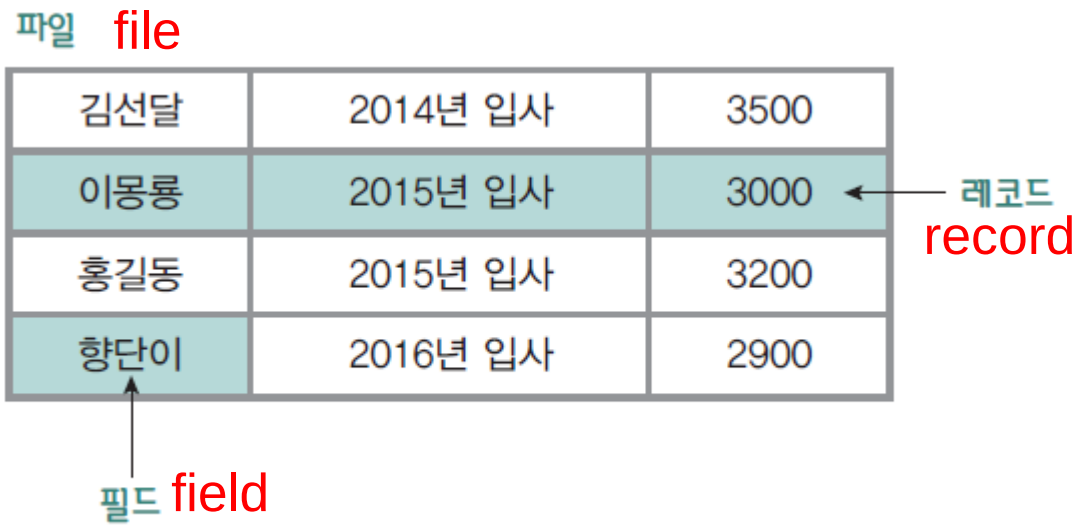


그림 2-30 필드, 레코드, 파일의 개념

3. Structure: declaration

❖ Structure name, data types, data fields

- Since all the array elements are of the same data type, it can be used without declaring the array elements. However, since each item can have a different data type in the structure, the data type and item name (variable name) must be declared for each item.

```
struct Name {
    type1 variable1;
    type2 variable2;
    …
    type_n variable_n;
}

struct Name struct_variable_name;
```

3. Structure: usage example


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

그림 2-32 구조체형 employee 선언 예

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

그림 2-33 구조체형 employee에 대한 구조체 변수 선언 예

3. Structure: declaration

표 2-2 구조체 변수의 선언 방법

방법	예

구조체형을 선언한 후에 구조체 변수 선언
Structure definition → variable declaration
	struct employee {
	  char name[10];
	  int year;
	  int pay;
	};
	struct employee Lee;

구조체형과 구조체 변수를 연결하여 선언
Structure definition and variable declaration at the same time
	struct employee {
	  char name[10];
	  int year;
	  int pay;
	} Lee;

구조체형 이름을 생략하고 구조체 변수 이름만 선언
Anonymous structure (one time)
	struct {
	  char name[10];
	  int year;
	  int pay;
	} Lee;

3. Structure: initialization

- To initialize a structure variable like normal variable initialization, specify the initial value of the variable while declaring the structure variable.
- Since a structure can have multiple internal items, specify the data type and number of internal items as a list of initial values in order and use braces ({ }).

```
struct employee { // 구조체형 선언
    char name[10];
    int year;
    int pay;
};
struct employee Lee = { "Ann", 2015, 4200 }; // 구조체 변수의 초기화
```

그림 2-34 구조체 변수를 초기화한 예

3. Structure: accessing data items

- Dot operator: .
  - Used to individually specify data items in a struct variable

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

For better compatibility,
`strcpy(Lee.name,"Ann");`
is recommended

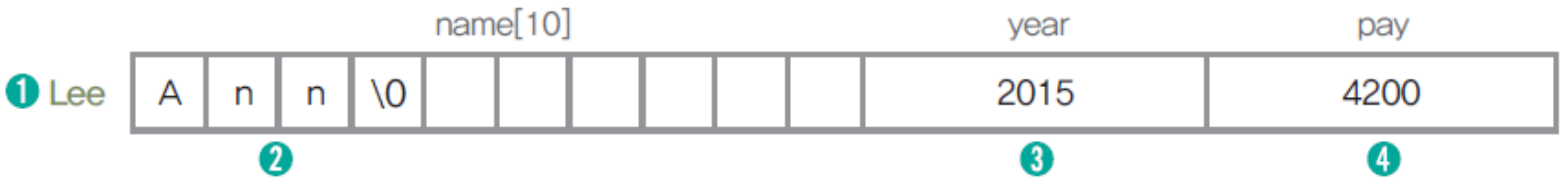


그림 2-35 점 연산자를 이용한 데이터 항목값 지정

3. Structure: accessing data items

ex2_12.c *

```c
#include <stdio.h>

struct employee{
  char name[20];
  int year;
  int pay;
};

int main()
{
  int i;
  struct employee Lee[4]={
    {"Jinho Lee", 2002, 3200},
    {"Hanyoung Lee", 2002, 3000},
    {"Sangwon Lee", 2004, 2500},
    {"Sangbeom Lee", 2003, 2900}
  };

  printf("Name\tYear\tSallary\n");
  printf("------------------------------\n");
  for(i=0; i<4; i++) {
    printf("%s\t%d\t%d\n",
        Lee[i].name, Lee[i].year, Lee[i].pay);
  }
}
```

```
$ gcc -W -Wall ex2_12.c -o
ex2_12.exe
$ ./ex2_12.exe
Name    Year    Sallary
------------------------------
Jinho Lee       2002    3200
Hanyoung Lee    2002    3000
Sangwon Lee     2004    2500
Sangbeom Lee    2003    2900
```

3. Structure: accessing data items

- Arrow operator: ->
  - In a structure pointer variable, the arrow operator is used to specify the data item of the structure variable pointed to by the pointer.

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

For better compatibility,
`strcpy(Sptr->name,"susan");`
is recommended

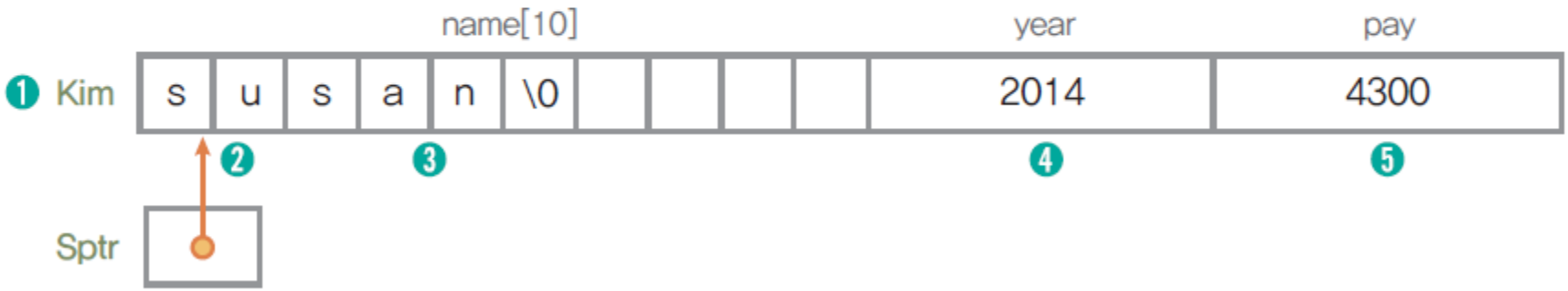


그림 2-36 화살표 연산자를 이용한 데이터 항목값 지정

3. Structure: accessing data items


```
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

그림 2-37 구조체 포인터를 이용한 데이터 항목 지정 방법

Error without parentheses

3. Structure: operations

❖ Reference operation for data fields

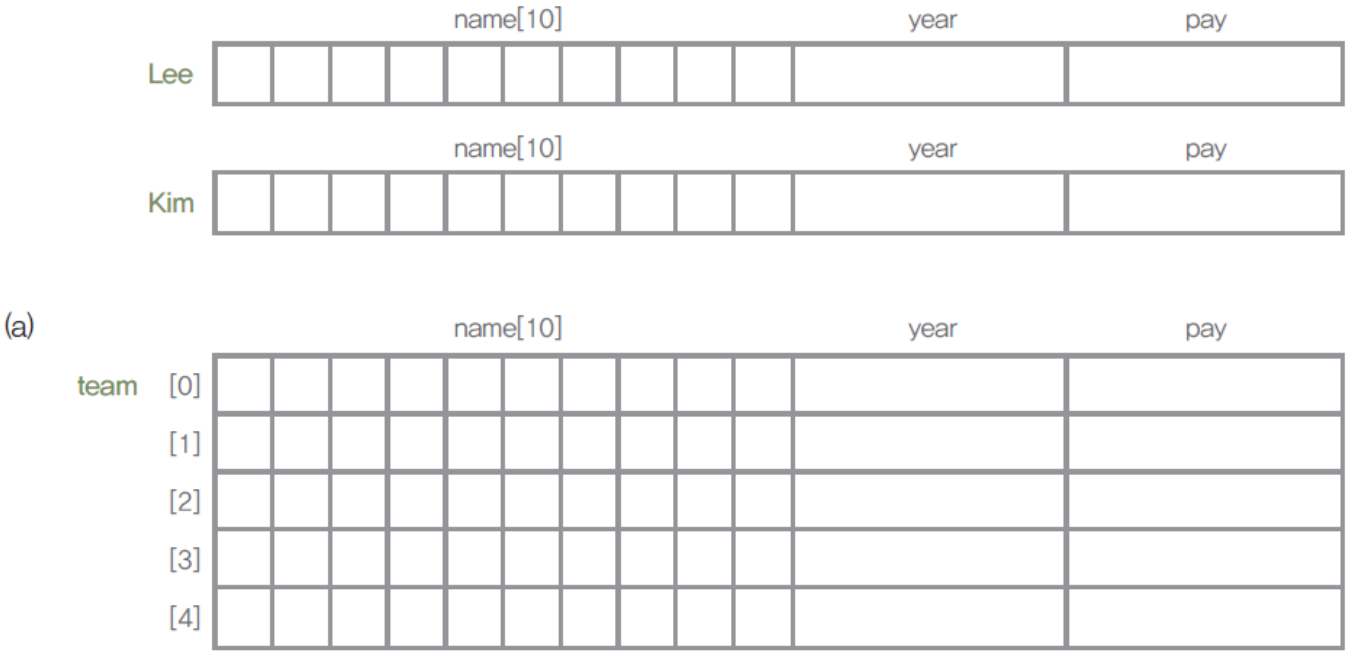
- Separate reference to struct data items using dot and arrow operators

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

그림 2-38 데이터 항목 참조 연산 예

3. Structure: copying struct variables


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



(b)

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

그림 2-39 구조체 변수 복사 연산 예

Note: may not be supported according to the OS/C versions.
It is recommended to copy each item, or use `memcpy()` function
ex) `memcpy(&Lee, &Kim, sizeof(struct employee));`

3. Structure: referencing

❖ How to obtain the memory address of a struct variable

- The address of a structure variable can be obtained by using the address operator of a pointer, or, if the structure variable is an array, the address can be obtained from the name of the structure array variable according to the characteristics of the array.

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

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

그림 2-40 구조체 변수의 주소 구하기 연산 예

[Example 2-13] Referencing data items using arrow operators

ex2_13.c *

```c
#include <stdio.h>
#include <string.h>     // strcpy(), memcpy()
#include <stdlib.h>     // malloc(), free()
struct employee{
  char name[20]; int year, pay;
};

int main() {
  int i;
  struct employee Lee[]={ {}, {}, {"Sangwon Lee", 2004, 2500} };
  struct employee *Sptr[3];

  // struct pointer assigned struct variable's address
  Sptr[0] = &(Lee[0]);
  strcpy(Sptr[0]->name, "Jinho Lee");
  Sptr[0]->year = 2002; Sptr[0]->pay = 3200;

  // struct pointer assigned by malloc
  Sptr[1] = (struct employee*)malloc(sizeof(struct employee));
  strcpy(Sptr[1]->name, "Hanyoung Lee");
  Sptr[1]->year = 2002; Sptr[1]->pay = 3000;

  // struct pointer assigned by malloc and memcpy
  Sptr[2] = (struct employee*)malloc(sizeof(struct employee));
  memcpy(Sptr[2], Lee+2, sizeof(struct employee));

  printf("Name\tYear\tSallary\n");
  printf("-----------------------------\n");
  for(i=0; i<3; i++) printf("%s\t%d\t%d\n",
      Sptr[i]->name, Sptr[i]->year, Sptr[i]->pay);

  /* note: struct allocated by malloc should be "free()'d" */
  free(Sptr[2]); printf("Sptr[2] is freed\n");
  free(Sptr[1]); printf("Sptr[1] is freed\n");

  /* but not the address of a local struct variable -- memory error
   * it is returned to memory automatically after the function ends */
  free(Sptr[0]); printf("Sptr[0] is freed\n");
}
```

```
$ gcc -W -Wall ex2_13.c -o
ex2_13.exe
$ ./ex2_13.exe
Name    Year    Sallary
-----------------------------
Jinho Lee       2002    3200
Hanyoung Lee    2002    3000
Sangwon Lee     2004    2500
Sptr[2] is freed
Sptr[1] is freed
double free or corruption (out)
terminated (core dumped)
```

Intentionally added to a erroneous case.

4. Recursion: definition

❖ Recursive call (circular call)

- Calling itself
- Depending on the algorithm, recursive calling methods can reduce the size of the program compared to the procedural calling
- Used when it is more efficient to solve a small problem by splitting it into subtasks of the same type rather than solving the whole problem all at once.
  - Sub-task: A sub-step of the current task, that is, a smaller unit task.
- Base case
  - Termination condition of the recursive call sequence
  - Needed to avoid infinite calling

4. Recursion: factorial function

❖ Factorial

- Multiply all integers from 1 to n

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

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

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

…

$2! = 2 \times 1!$

$1! = 1$ (베이스케이스) Base case

그림 2-43 n! 연산

4. Recursion: efficiency

❖ Comparison of recursive and iterative calls

Recursive factorial function implementation

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

(a) 재귀호출을 이용한 팩토리얼 함수

Iterative factorial function implementation

```
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

그림 2-44 팩토리얼 함수

4. Recursion: call stack

❖ Call stack of the recursive factorial function with n=4

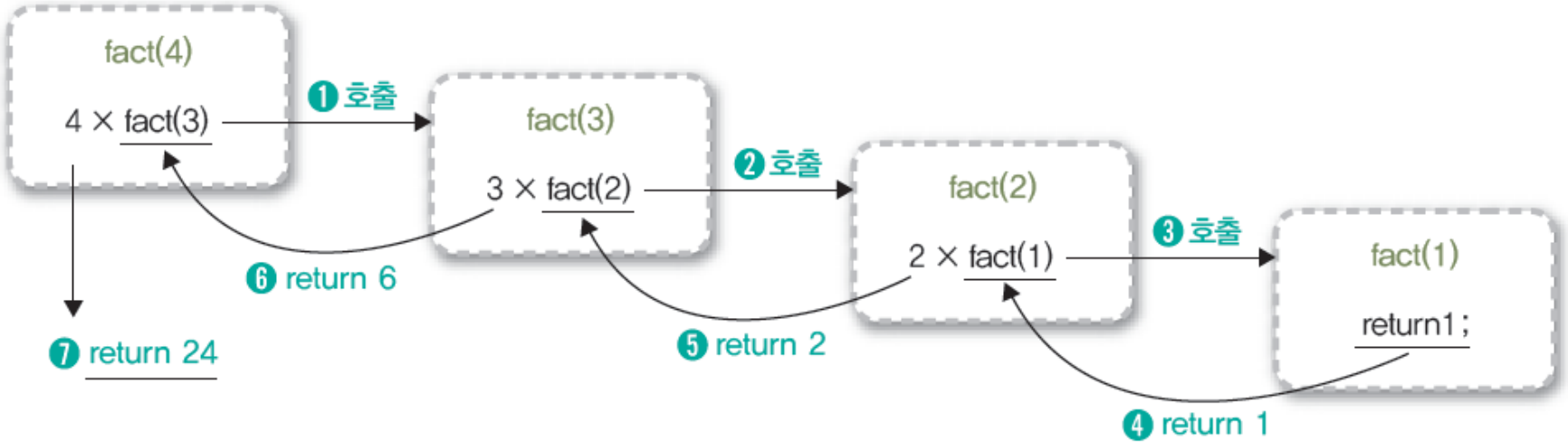


그림 2-45 n=4일 때 재귀호출과 반환

[Ex 2-14]: Recursive factorial function

ex2_14.c *

```
1  #include <stdio.h>
2  #include <stdlib.h>     // exit(), atoi()
3
4  long int fact(int n) {
5    long value;
6    if (n<=1){
7      printf("\n fact(1) called!");
8      printf("\n fact(1) returns 1 ");
9      return 1;
10   }
11   else{
12     printf("\n fact(%d) called!", n);
13     value = (n * fact(n-1));
14     printf("\n fact(%d) returns %ld ", n, value);
15     return value;
16   }
17 }
18
19 int main(int argc, char *argv[]) {
20   int n;
21   if ( argc < 2 ) {
22     fprintf(stderr, "usage: %s <integer (>=0)>\n",argv[0]);
23     exit(0);
24   }
25   else {
26     n = atoi(argv[1]);
27     printf("\n\n factorial %d is %ld\n", n, fact(n));
28   }
29 }
```

```
$ gcc -W -Wall ex2_14.c -o ex2_14.exe
$ ./ex2_14.exe
usage: ./ex2_14.exe <integer (>=0)>
$ ./ex2_14.exe 4

 fact(4) called!
 fact(3) called!
 fact(2) called!
 fact(1) called!
 fact(1) returns 1
 fact(2) returns 2
 fact(3) returns 6
 fact(4) returns 24

 factorial 4 is 24
```

4. Recursion: Tower of Hanoi

❖ Tower of Hanoi

(a) 하노이 탑 퍼즐 시작 상태

(b) 하노이 탑 퍼즐 종료 상태

그림 2-46 하노이 탑 퍼즐의 시작 상태와 종료 상태(원반 개수 n = 3)

4. Recursion: Tower of Hanoi

❖ Tower of Hanoi algorithm

1단계(1 ~ 3) : 시작봉 start에 있는 원반 1~원반 n-1을, 목적봉 target을 이용하여 중간 작업봉 work으로 옮긴다.

➜ `hanoi(n-1, start, target, work)`

2단계(4) : 시작봉 start에 있는 마지막 원반 n을 목적봉 target으로 옮긴다.

➜ `hanoi(1, start, work, target)`

3단계(5 ~ 7) : 중간 작업봉 work에 있는 원반 1~원반 n-1을, 시작봉 start을 이용하여 목적봉 target으로 옮긴다.

➜ `hanoi(n-1, work, start, target)`

그림 2-47 하노이 탑 작업 과정의 일반화

4. Recursion: Tower of Hanoi

- **The process of solving the Tower of Hanoi process using recursion when there are three disks**

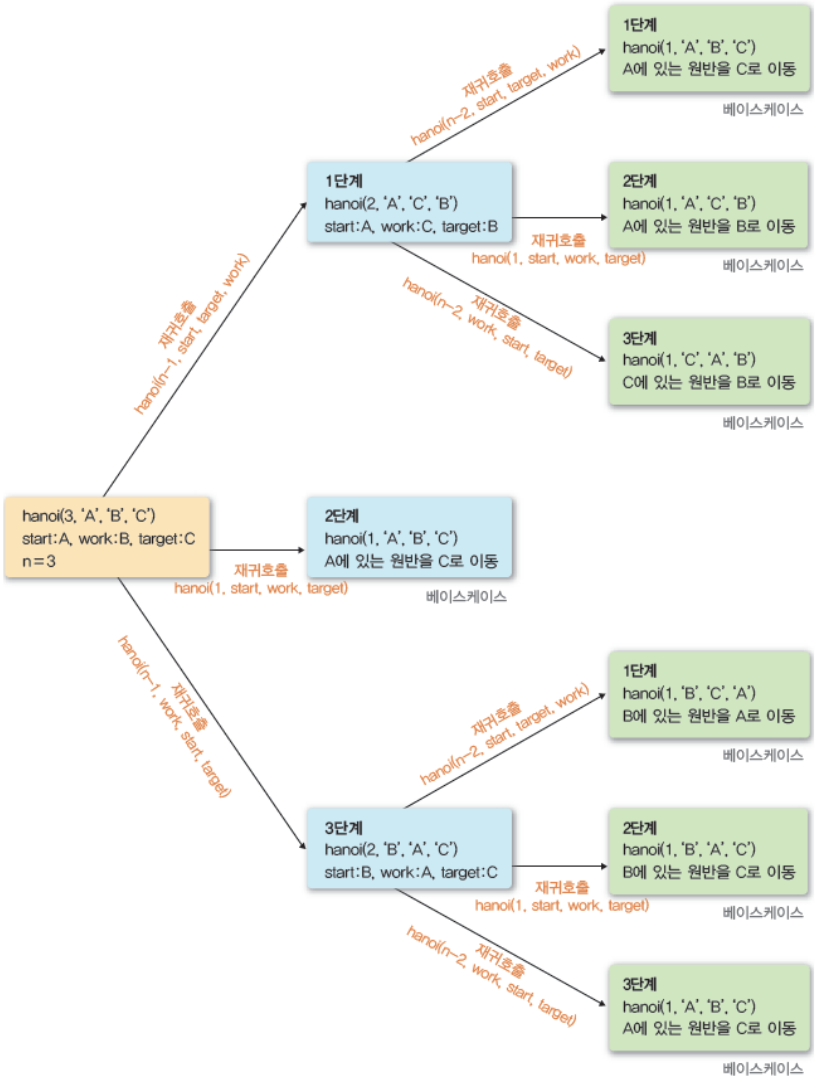


그림 2-48 원반이 세 개일 때 하노이 탑 과정을 재귀호출을 이용해 해결하는 과정

[Ex 2-15]: Recursive Hanoi tower

원리를 알면 IT가 맛있다

hanoi.c *

```c
/* Reference: https://nate9389.tistory.com/52 */

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

/* This is an algorithm to solve the Hanoi Tower problem */
// Step 1. transfer (n-1)th floor pyramid to other blank(A or B)
// Step 2. transfer the largest disk to C
// Step 3. repeat step 1 and step 2 (n is getting smaller)

void Hanoi(int n, char A, char B, char C) {
  if(n == 1) printf("%c --> %c\n", A, C);
  else{
    Hanoi(n-1, A, C, B);
    printf("%c --> %c\n", A, C);
    Hanoi(n-1, B, A, C);
  }
}

int main(int argc, char *argv[]) {
  int n;
  if ( argc < 2 ) {
    fprintf(stderr, "usage: %s <integer (>0)>\n",argv[0]);
    exit(0);
  }
  else {
    n = atoi(argv[1]);
    Hanoi(n, 'A', 'B', 'C');
  }
}
```

```
$ gcc -W -Wall hanoi.c -o hanoi.exe
$ ./hanoi.exe
usage: ./hanoi.exe <integer (>0)>
$ ./hanoi.exe 1
A --> C
$ ./hanoi.exe 2
A --> B
A --> C
B --> C
$ ./hanoi.exe 3
A --> C
A --> B
C --> B
A --> C
B --> A
B --> C
A --> C
$ ./hanoi.exe 4
A --> B
A --> C
B --> C
A --> B
C --> A
C --> B
A --> B
A --> C
B --> C
B --> A
C --> A
B --> C
A --> B
A --> C
B --> C
```

Thank You !

IT CookBook, C로 배우는 쉬운 자료구조(개정판)

한빛미디어 Hanbit Media, Inc.