

윤성우의 열혈 자료구조

: C언어를 이용한 자료구조 학습서



Chapter 02. 재귀(Recursion)

Introduction To Data Structures Using C

Chapter 02. 재귀(Recursion)

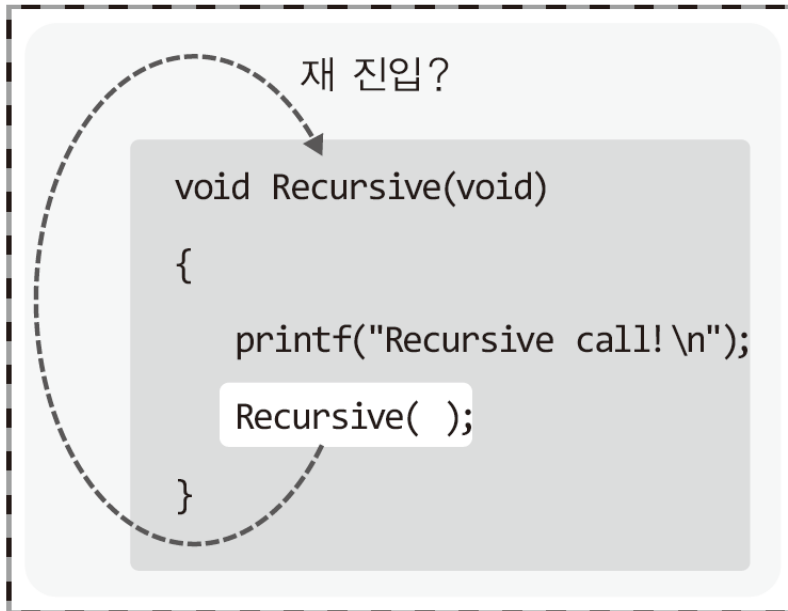


Chapter 02-1:

함수의 재귀적 호출의 이해

재귀함수의 기본적인 이해 1

재귀함수: 함수 내에서 함수 자신을 호출하는 형태의 함수를 말함.



재귀함수의 기본적인 이해 2

재귀함수의 간단한 예

```
void Recursive(int num)
{
    if(num <= 0)        // 재귀의 탈출조건
        return;        //재귀의 탈출!
    printf("Recursive call! %d \n", num);
    Recursive(num-1);
}

int main(void)
{
    Recursive(3);
    return 0;
}
```

사용가능한 재귀함수를 만들려면

- ① 재귀의 탈출조건이 있음!!
- ② 매개변수를 바꿔서 재귀 호출함!!

실행결과

```
Recursive call! 3
Recursive call! 2
Recursive call! 1
```

재귀함수의 디자인 사례

- 팩토리얼(factorial)

$$n! = n \times (n-1) \times (n-2) \times (n-3) \times \dots \times 2 \times 1$$

$(n-1)!$



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

팩토리얼의 수학적 표현

$$f(n) = \begin{cases} n \times f(n-1) & \dots n \geq 1 \\ 1 & \dots n = 0 \end{cases}$$



팩토리얼의 코드

```
if(n == 0)
    return 1;
else
    return n * Factorial(n-1);
```

팩토리얼의 재귀적 구현

팩토리얼 구현 결과

```
int Factorial(int n)
{
    if(n == 0)
        return 1;
    else
        return n * Factorial(n-1);
}

int main(void)
{
    printf("1! = %d \n", Factorial(1));
    printf("2! = %d \n", Factorial(2));
    printf("3! = %d \n", Factorial(3));
    printf("4! = %d \n", Factorial(4));
    printf("9! = %d \n", Factorial(9));
    return 0;
}
```

사용가능한 재귀함수를 만들려면

① 재귀의 탈출조건이 있음!!

② 매개변수를 바꿔서 재귀 호출함!!

$$f(n) = \begin{cases} n \times f(n-1) & \dots n \geq 1 \\ 1 & \dots n = 0 \end{cases}$$

실행결과

```
1! = 1
2! = 2
3! = 6
4! = 24
9! = 362880
```

Chapter 02. 재귀 (Recursion)



Chapter 02-2:

재귀의 활용

피보나치 수열 1: 이해

피보나치 수열

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55

피보나치 수열의 구성

수열의 n 번째 값 = 수열의 $n-1$ 번째 값 + 수열의 $n-2$ 번째 값

피보나치 수열의 수학적 표현

$$fib(n) = \begin{cases} 0 & \dots n=1 \\ 1 & \dots n=2 \\ fib(n-1) + fib(n-2) & \dots \text{otherwise} \end{cases}$$

피보나치 수열 2: 코드의 구현

피보나치 수열의 수학적 표현

$$fib(n) = \begin{cases} 0 & \dots n=1 \\ 1 & \dots n=2 \\ fib(n-1) + fib(n-2) & \dots \text{otherwise} \end{cases}$$



피보나치 수열의 코드

```
int Fibo(int n)
{
    if(n == 1)
        return 0;
    else if(n == 2)
        return 1;
    else
        return Fibo(n-1) + Fibo(n-2);
}
```

피보나치 수열 3: 완성된 예제

```
int Fibo(int n)
{
    if(n == 1)
        return 0;
    else if(n == 2)
        return 1;
    else
        return Fibo(n-1) + Fibo(n-2);
}

int main(void)
{
    int i;
    for(i=1; i<15; i++)
        printf("%d ", Fibo(i));

    return 0;
}
```

사용가능한 재귀함수를
만들려면

① 재귀의 탈출조건이 있음!!

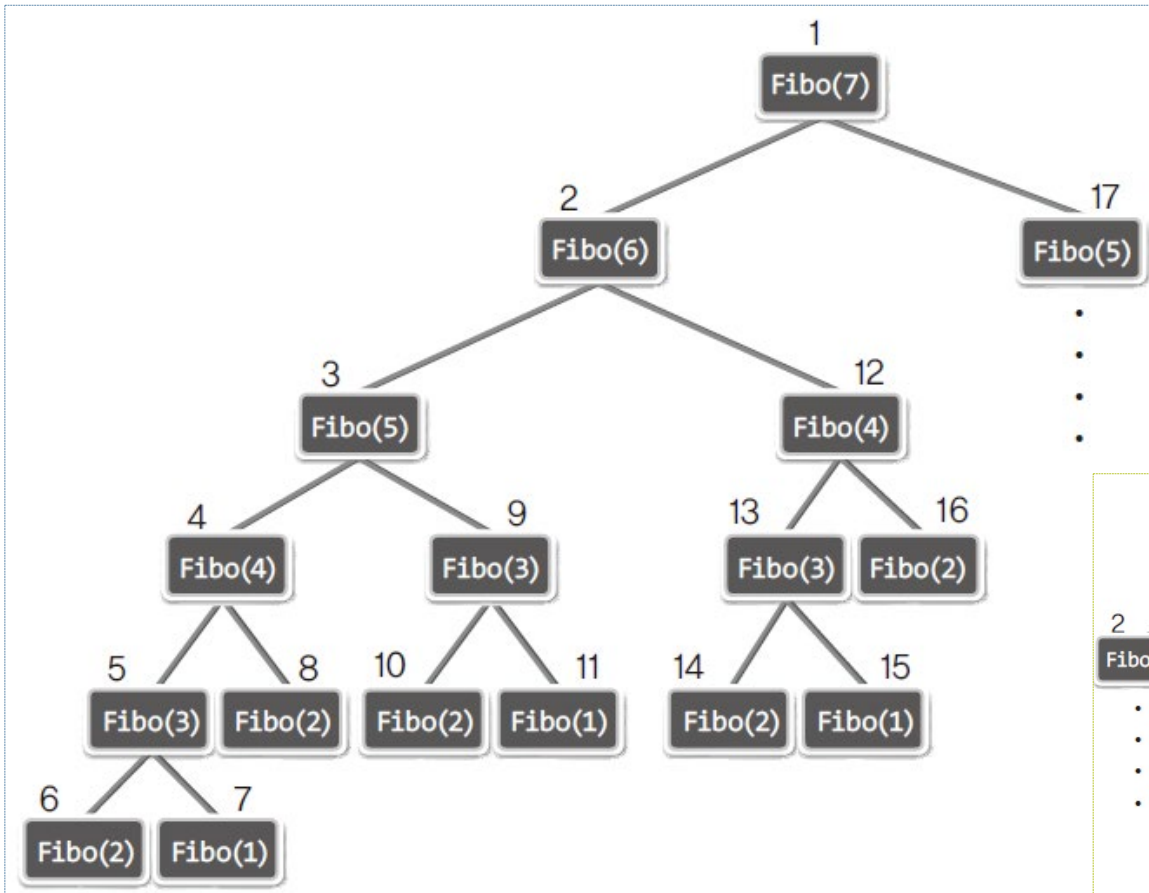
② 매개변수를 바꿔서 재귀 호출함!!

$$fib(n) = \begin{cases} 0 & \dots n=1 \\ 1 & \dots n=2 \\ fib(n-1) + fib(n-2) & \dots \text{otherwise} \end{cases}$$

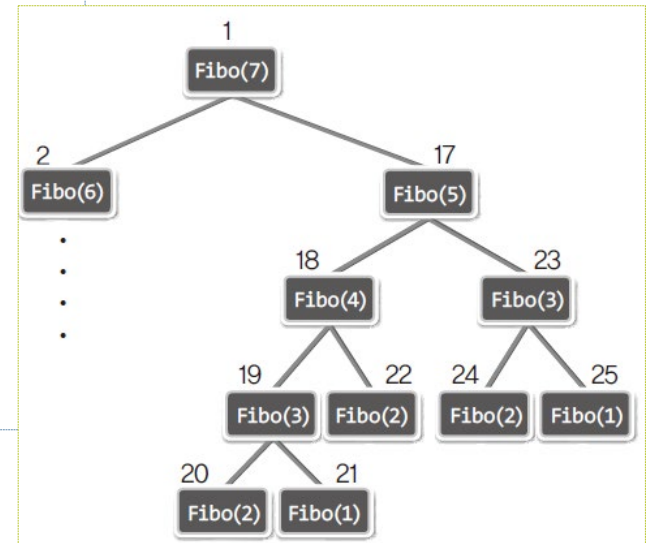
실행결과

0	1	1	2	3	5	8	13	21	34	55	89	144	233
---	---	---	---	---	---	---	----	----	----	----	----	-----	-----

피보나치 수열 4: 함수의 흐름



- 재귀 함수의 호출 순서를 정리하는 것은 큰 의미가 없음.
- 따라서 재귀함수 자체를 정확히 이해하고 사용하는 것이 중요함.



이진 탐색 알고리즘의 재귀구현 1

이진 탐색의 알고리즘의 핵심

1. 탐색 범위의 중앙에 목표 값이 저장되었는지 확인
2. 저장되지 않았다면 탐색 범위를 반으로 줄여서 다시 탐색 시작

이진 탐색의 종료에 대한 논의

```
int BSearchRecur(int ar[], int first, int last, int target)
{
    if(first > last)
        return -1;
    . . .
}
```

first와 last는 각각 탐색의 시작과 끝을 나타내는 변수

// -1의 반환은 탐색의 실패를 의미

1장에서는 <= 였음. if(first <= last)

이진 탐색 알고리즘의 재귀구현 2

탐색 대상의 확인!

```
int BSearchRecur(int ar[], int first, int last, int target)
{
    int mid;
    if(first > last)
        return -1;

    탐색의 대상에서 중심에 해당하는 인덱스 값 계산
    mid = (first+last) / 2;
    if(ar[mid] == target) 타겟이 맞는지 확인!
        return mid;
    . . . .
}
```

이진 탐색 알고리즘의 재귀구현 3

계속되는 탐색

```
int BSearchRecur(int ar[], int first, int last, int target)
{
    int mid;
    if(first > last)
        return -1;
    mid = (first+last) / 2;

    if(ar[mid] == target)
        return mid;
    else if(target < ar[mid])
        return BSearchRecur(ar, first, mid-1, target);
    else
        return BSearchRecur(ar, mid+1, last, target);
}
```

앞부분을 대상으로 재 탐색

뒷부분을 대상으로 재 탐색

Chapter 02. 재귀(Recursion)

Chapter 02-3:

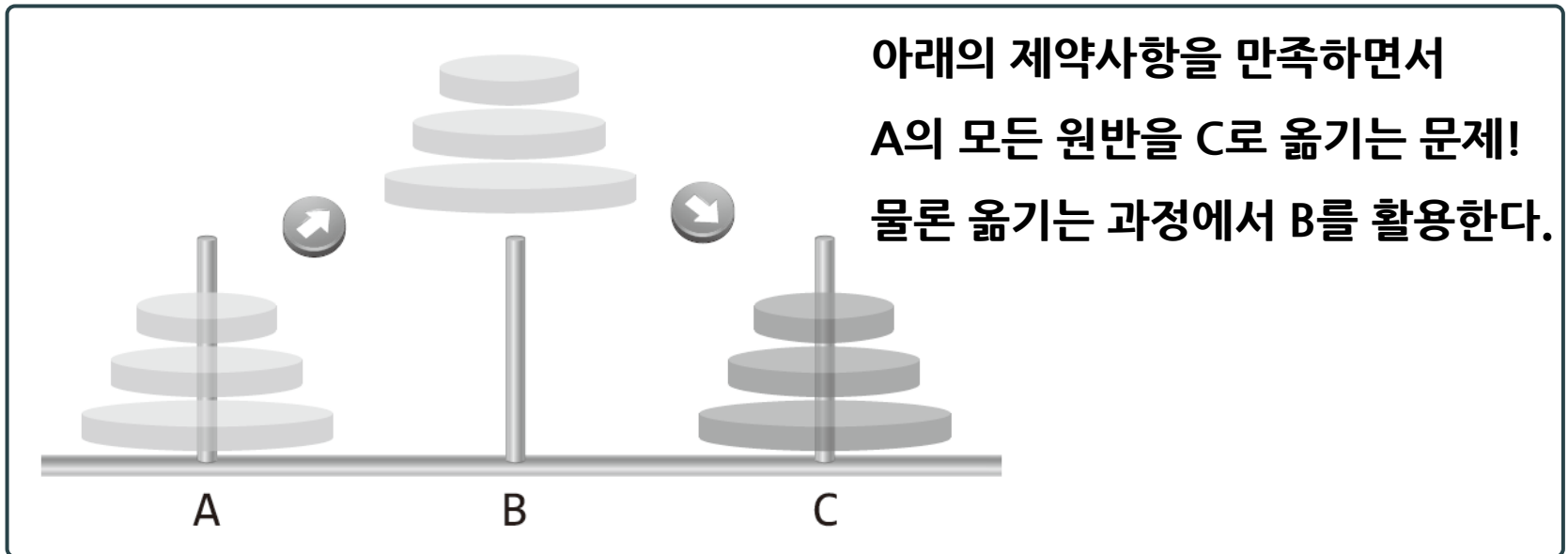
하노이 타워

하노이: 베트남의 수도



하노이 타워 문제의 이해

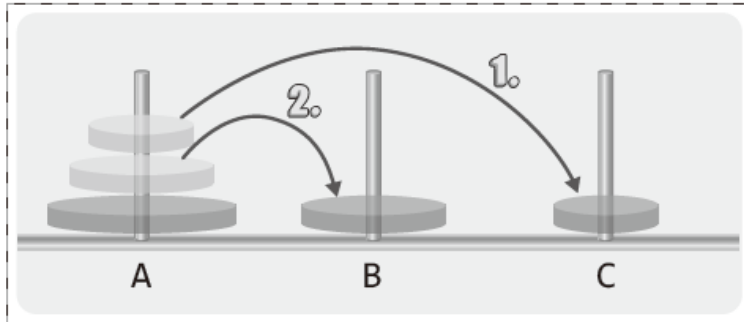
원반을 A에서 C로 이동하기



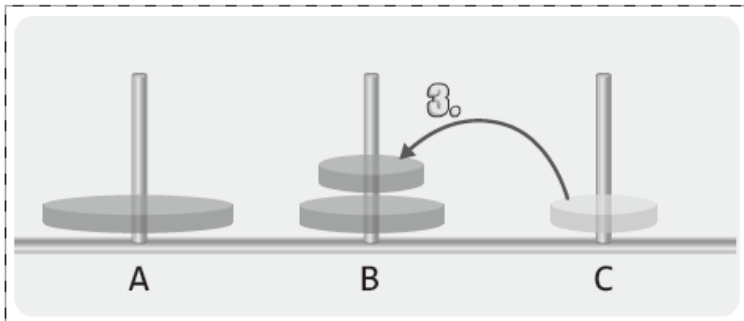
제약사항

- 원반은 한 번에 하나씩만 옮길 수 있다.
- 옮기는 과정에서 작은 원반의 위에 큰 원반이 올려져서는 안된다.

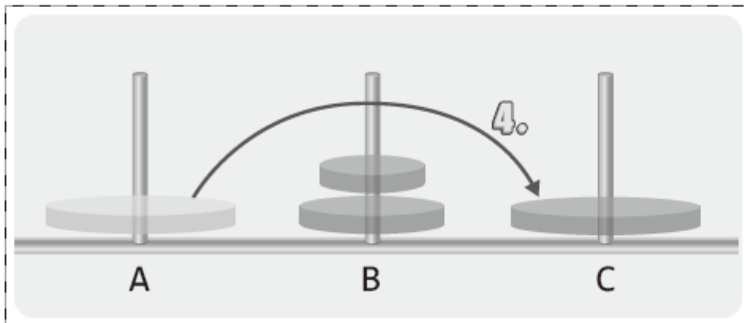
하노이 타워 문제 해결의 예 (1/2)



▶ [그림 02-8: 문제해결 1/5]

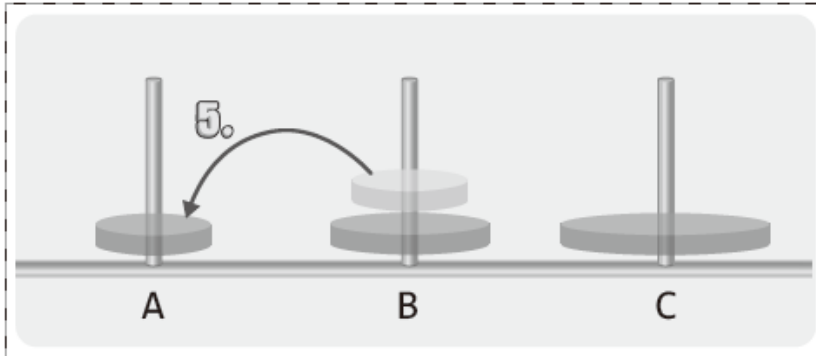


▶ [그림 02-9: 문제해결 2/5]

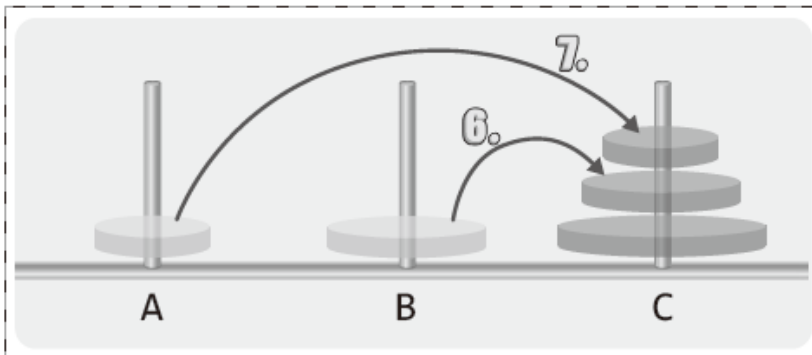


▶ [그림 02-10: 문제해결 3/5]

하노이 타워 문제 해결의 예 (2/2)



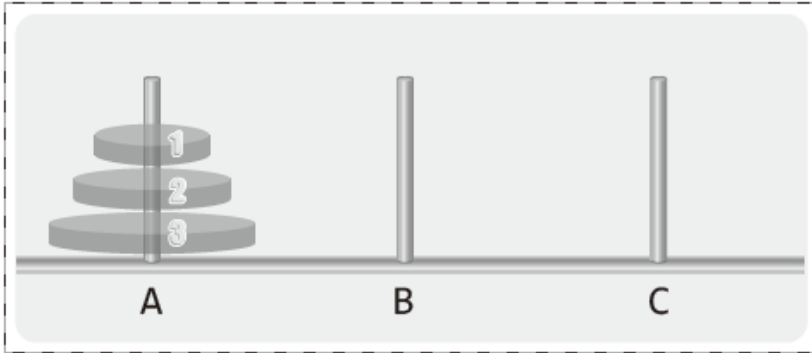
▶ [그림 02-11: 문제해결 4/5]



▶ [그림 02-12: 문제해결 5/5]

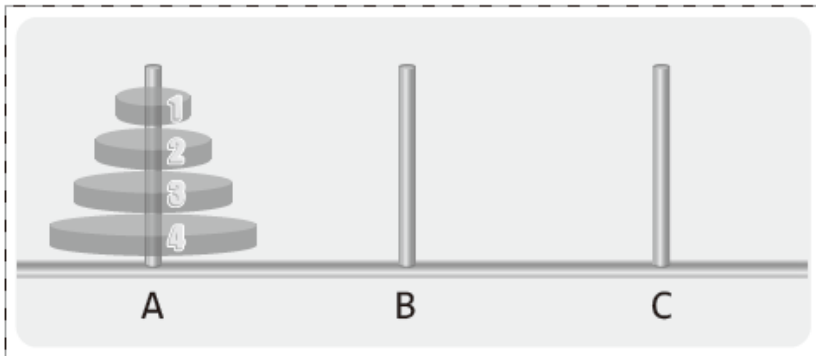
지금 보인 과정에서 **반복의 패턴**을 찾아야 문제의 해결을 위한 코드를 작성할 수 있다!

반복되는 일련의 과정을 찾기 위한 힌트



A의 세 원반을 C로 옮기기 위해서는 **원반 3을 C로** 옮겨야 한다. 그리고 이를 위해서는 원반 1과 2를 우선 **원반 B로** 옮겨야 한다.

▶ [그림 02-13: 원반이 3개인 하노이 타워]

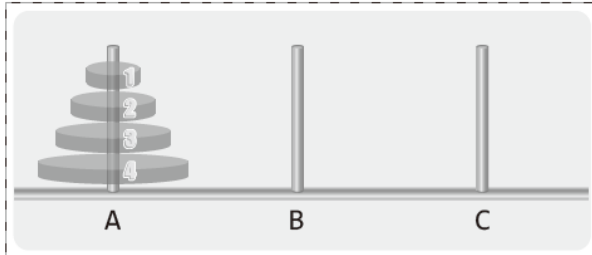


A의 네 원반을 C로 옮기기 위해서는 **원반 4를 C로** 옮겨야 한다. 그리고 이를 위해서는 원반 1과 2와 3을 우선 **원반 B로** 옮겨야 한다.

▶ [그림 02-14: 원반이 4개인 하노이 타워]

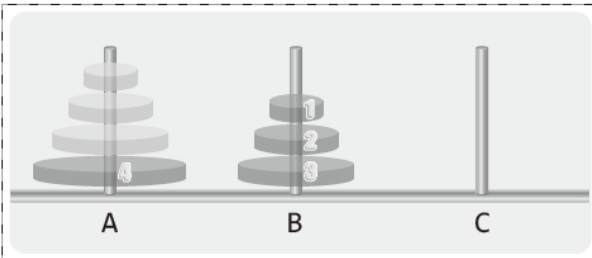
위의 두 예를 통해서 문제의 해결에 있어서 **반복이 되는 패턴**이 있음을 알 수 있다.

하노이 타워의 반복패턴 연구 (1/2)



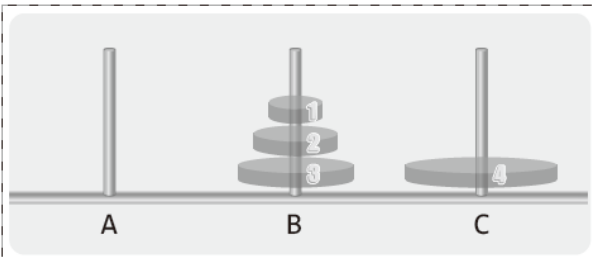
목적: 원반 4개를 A에서 C로 이동

▶ [그림 02-14: 원반이 4개인 하노이 타워]



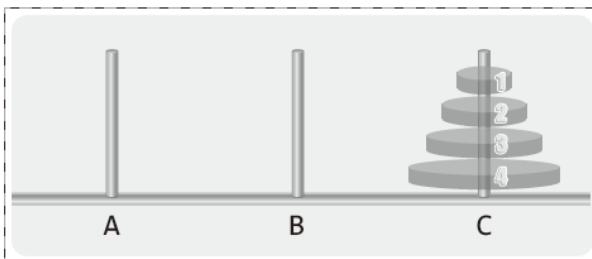
1. 작은 원반 3개를 A에서 B로 이동

▶ [그림 02-15: 반복패턴 1/3]



2. 큰 원반 1개를 A에서 C로 이동

▶ [그림 02-16: 반복패턴 2/3]



3. 작은 원반 3개를 B에서 C로 이동

▶ [그림 02-17: 반복패턴 3/3]

하노이 타워의 반복패턴 연구 (2/2)

목적. 원반 4개를 A에서 C로 이동

1. 작은 원반 3개를 A에서 B로 이동

2. 큰 원반 1개를 A에서 C로 이동

3. 작은 원반 3개를 B에서 C로 이동

목적: 큰 원반 n 개를 A에서 C로 이동

1. 작은 원반 $n-1$ 개를 A에서 B로 이동

2. 큰 원반 1개를 A에서 C로 이동

3. 작은 원반 $n-1$ 개를 B에서 C로 이동

하노이 타워 문제의 해결 1

하노이 타워 함수의 기본 골격

```
void HanoiTowerMove(int num, char from, char by, char to)
{
    . . . . .
}
```

num개의 원반을 from에서 to로 이동을 시키되, 그 과정에서 by를 활용한다.

목적: 원반 n 개를 A에서 C로 이동

1. 작은 원반 $n-1$ 개를 A에서 B로 이동
2. 큰 원반 1개를 A에서 C로 이동
3. 작은 원반 $n-1$ 개를 B에서 C로 이동

```
HanoiTowerMove(num, from, by, to);
                        n   A   B   C
HanoiTowerMove(num-1, from, to, by);
                        n-1   A   C   B
printf( . . . . );
HanoiTowerMove(num-1, by, from, to);
                        n-1   B   A   C
```

하노이 타워 문제의 해결 2

목적: 원반 n 개를 A에서 C로 이동

1. 작은 원반 $n-1$ 개를 A에서 B로 이동
2. 큰 원반 1개를 A에서 C로 이동
3. 작은 원반 $n-1$ 개를 B에서 C로 이동

HanoiTowerMove(num, from, by, to);

HanoiTowerMove(num-1, from, to, by);

printf(. . .);

HanoiTowerMove(num-1, by, from, to);

```
void HanoiTowerMove(int num, char from, char by, char to)
```

```
{
```

```
    if(num == 1)        // 이동할 원반의 수가 1개라면
```

```
    {
```

```
        printf("원반1을 %c에서 %c로 이동 \n", from, to);
```

```
    }
```

```
    else
```

```
    {
```

```
        HanoiTowerMove(num-1, from, to, by);
```

```
        printf("원반%d을(를) %c에서 %c로 이동 \n", num, from, to);
```

```
        HanoiTowerMove(num-1, by, from, to);
```

```
    }
```

```
}
```

하노이 타워 문제의 해결 3

```
void HanoiTowerMove(int num, char from, char by, char to)
{
    if(num == 1)        // 이동할 원반의 수가 1개라면
    {
        printf("원반1을 %c에서 %c로 이동 \n", from, to);
    }
    else
    {
        HanoiTowerMove(num-1, from, to, by);
        printf("원반%d을(를) %c에서 %c로 이동 \n", num, from, to);
        HanoiTowerMove(num-1, by, from, to);
    }
}

int main(void)
{
    // 막대A의 원반 3개를 막대B를 경유하여 막대C로 옮기기
    HanoiTowerMove(3, 'A', 'B', 'C');
    return 0;
}
```

실행결과

원반1을 A에서 C로 이동
원반2을(를) A에서 B로 이동
원반1을 C에서 B로 이동
원반3을(를) A에서 C로 이동
원반1을 B에서 A로 이동
원반2을(를) B에서 C로 이동
원반1을 A에서 C로 이동

연습문제1

▶ 재귀 함수에 대한 설명으로 **잘못된** 것은?

- ① 재귀 함수는 함수의 내부에서 자기 자신을 다시 호출한다.
- ② 일반적으로 재귀 함수에는 탈출 조건을 작성한다.
- ③ 일반적으로 재귀 함수 내부에서 자기 자신을 다시 호출할 때, 매개변수를 바꿔 호출한다.
- ④ 재귀적 수학 표현을 재귀 함수로 작성하는 것은 매우 어렵다.

연습문제2

- ▶ 하노이 타워 문제에 대한 설명으로 **잘못된** 것은?
 - ① 하노이 타워 문제는 재귀 함수를 이용해 쉽게 풀 수 있다.
 - ② 하노이 타워 문제를 해결하기 위한 재귀 함수 코드는 약 한 페이지 분량으로 길다.
 - ③ 하노이 타워의 원반은 한 번에 하나씩만 옮길 수 있다.
 - ④ 원반을 옮기는 과정에서 작은 원반 위에 큰 원반이 올려져서는 안된다.

수고하셨습니다~



Chapter 02에 대한 강의를 마칩니다!