Data Structure

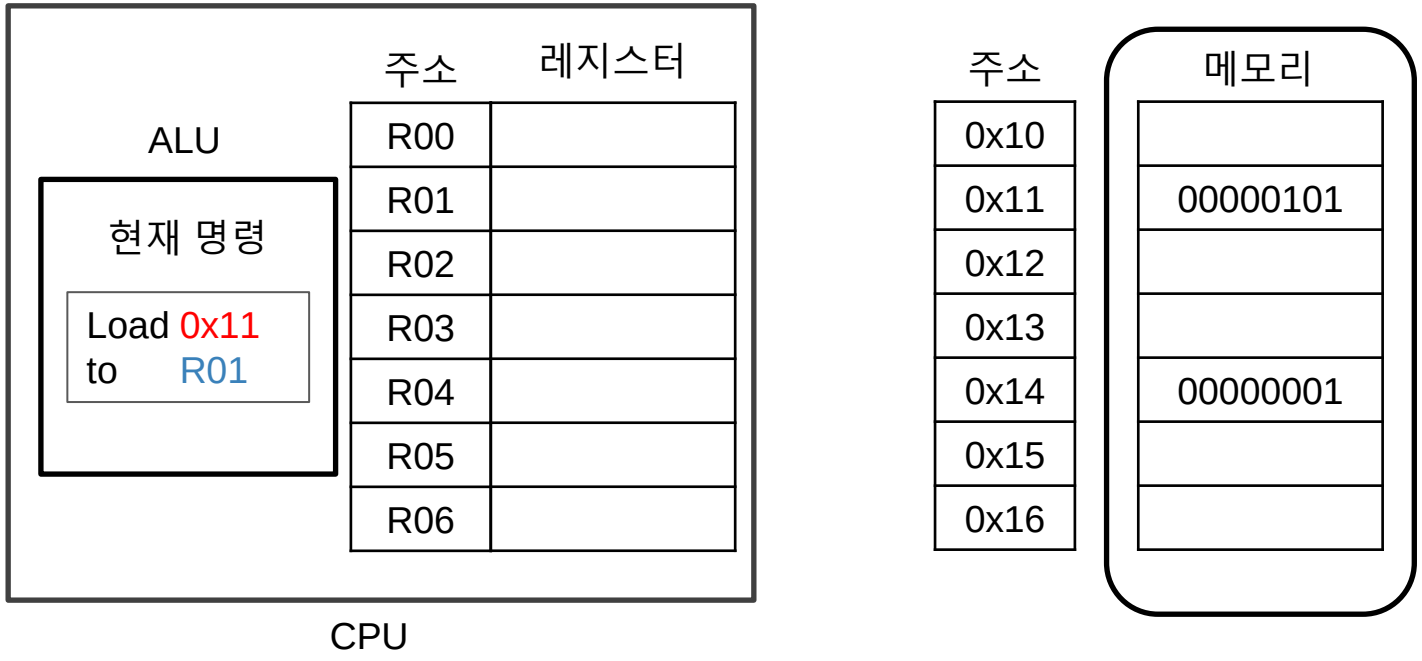
실습 1

0. 이번 주 실습 내용

- ‘C’ Review
  - 포인터(pointer)
  - 메모리(memory)
  - 구조체(structure)
  - 재귀함수(recursive function)
- 실습 과제
  - 하노이 탑(Hanoi tower)

1. 포인터 (Pointer)

- 컴퓨터의 동작구조(간략화, 참고만 하세요)



C 프로그램

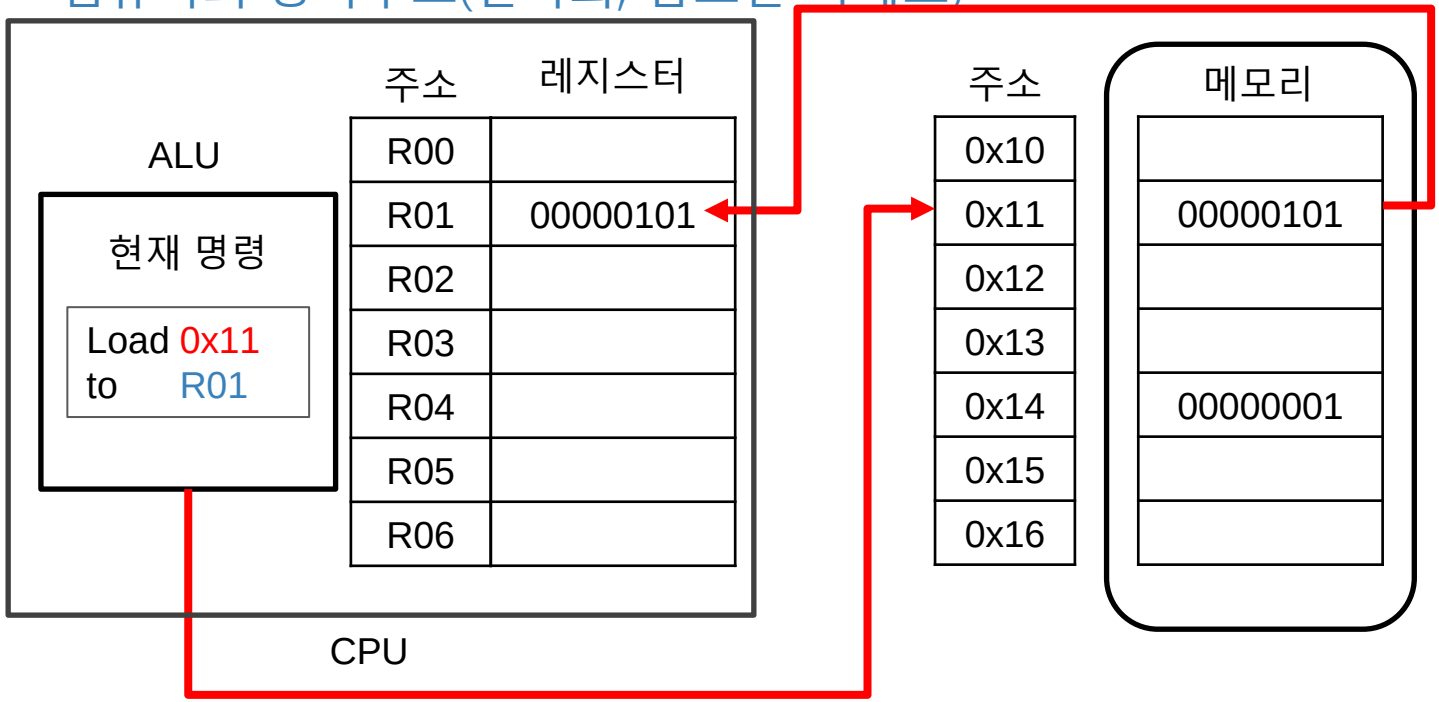
```
int a = 5;
int b = 1;
a = a+b;
```

변수주소 테이블

a: 0x11
b: 0x14

1. 포인터 (Pointer)

- 컴퓨터의 동작구조(간략화, 참고만 하세요)



C 프로그램

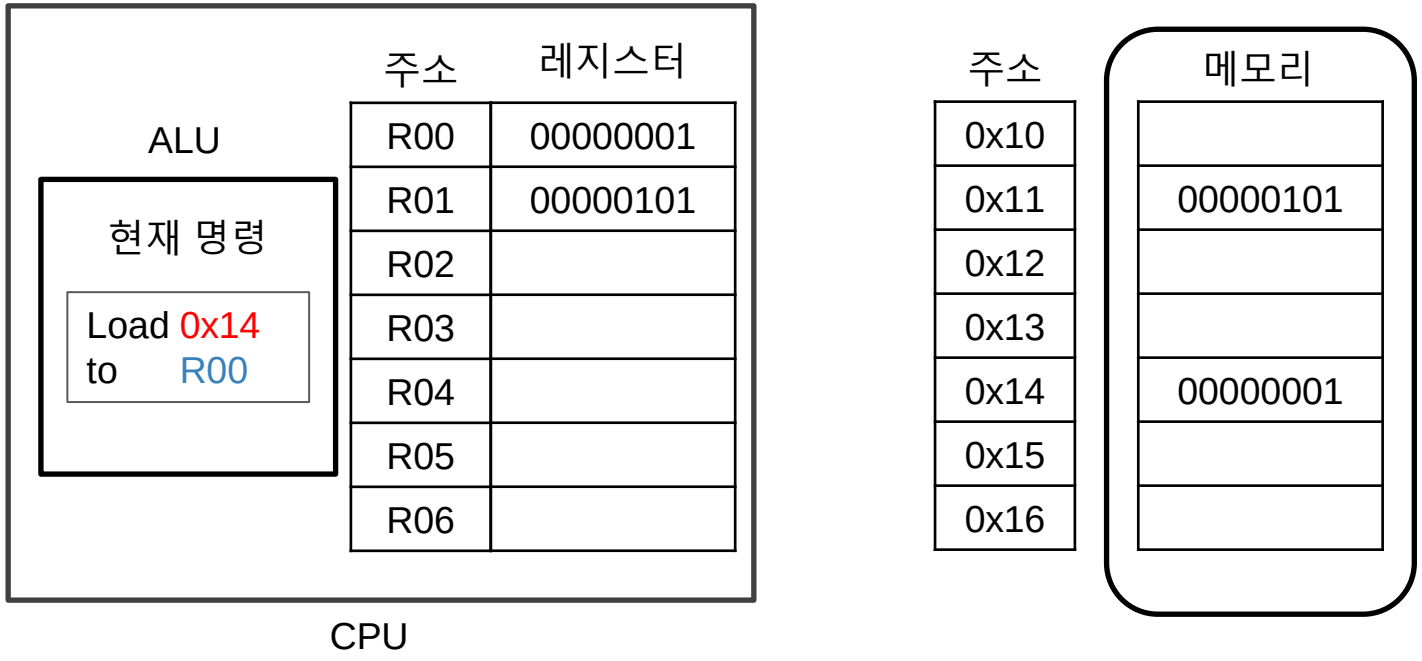
```
int a = 5;
int b = 1;
a = a+b;
```

변수주소 테이블

a: 0x11
b: 0x14

1. 포인터 (Pointer)

- 컴퓨터의 동작구조(간략화, 참고만 하세요)



C 프로그램

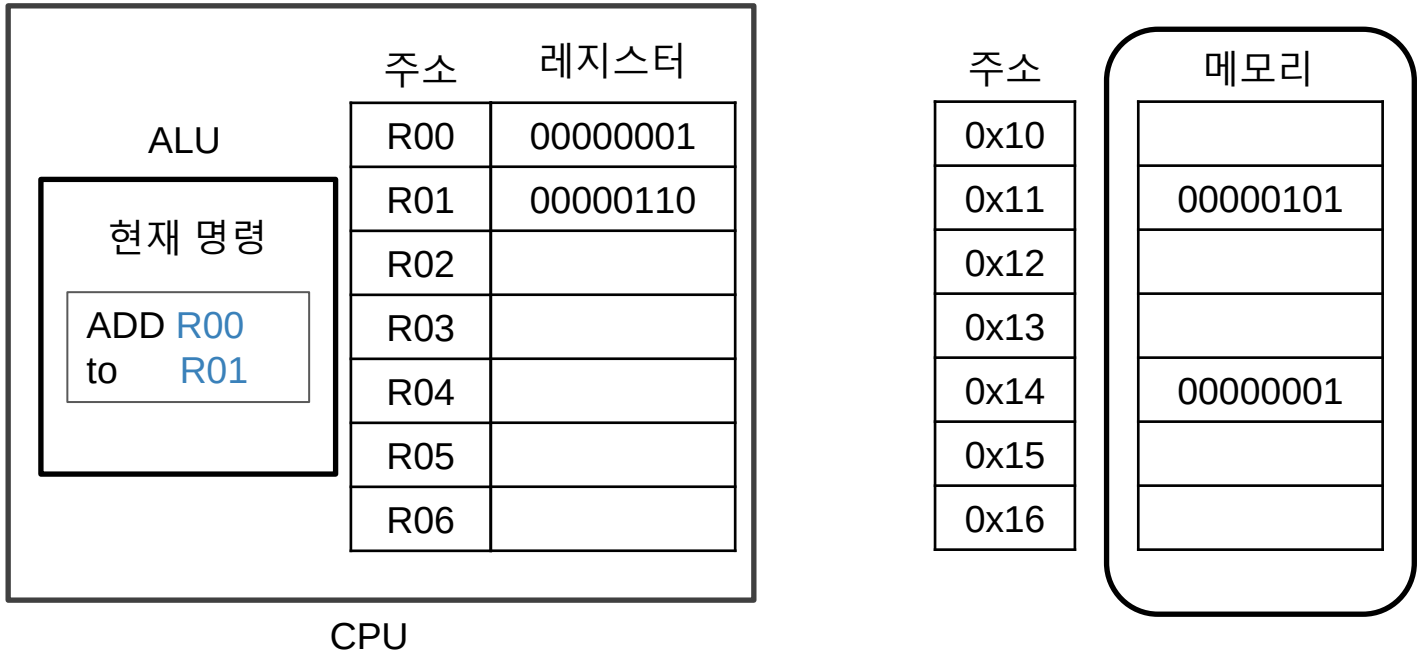
```
int a = 5;
int b = 1;
a = a+b;
```

변수주소 테이블

a: 0x11
b: 0x14

1. 포인터 (Pointer)

- 컴퓨터의 동작구조(간략화, 참고만 하세요)



C 프로그램

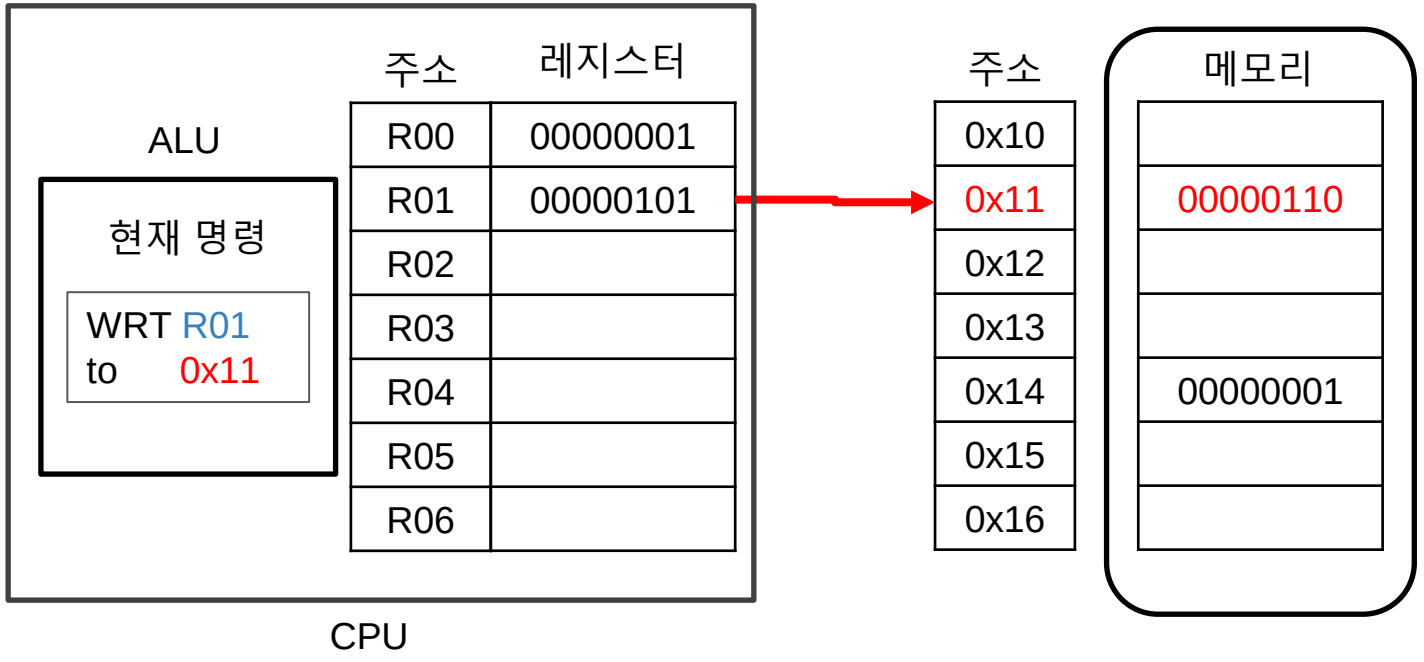
```
int a = 5;
int b = 1;
a = a+b;
```

변수주소 테이블

a: 0x11
b: 0x14

1. 포인터 (Pointer)

- 컴퓨터의 동작구조(간략화, 참고만 하세요)



C 프로그램

```
int a = 5;
int b = 1;
a = a+b;
```

변수주소 테이블

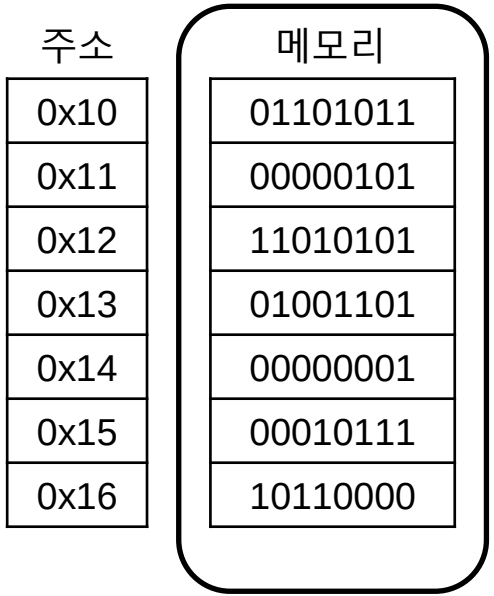
a: 0x11
b: 0x14

- 관련된 정확한 이론은 '컴퓨터구조론', '오토마타 및 계산이론', '컴파일러' 수업을 수강바랍니다.

1. 포인터 (Pointer)

- **정의: 메모리 상의 임의의 주소를 저장하는 변수**
- 역할: 변수들이 저장된 위치(메모리 주소)정보를 이용하여 직접 접근
- * 기호로 선언
  - EX) int *ptr; //int형 변수의 주소값을 저장하는 변수 ptr
  - EX) char *cp; //char형 변수의 주소값을 저장하는 변수 cp

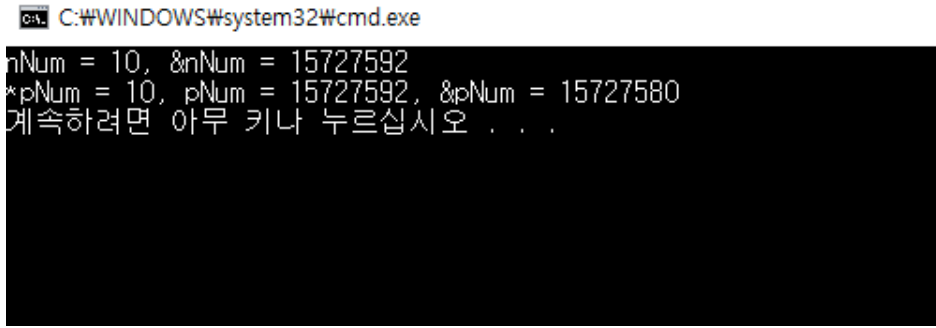
- 포인터 관련 연산자
  - **연산자 &** : 변수에 할당된 메모리의 시작 **주소**
  - **연산자 *** : 포인터 변수가 가리키는 곳의 **내용**



1. 포인터 (Pointer)

Example

```c
int main( )
{
    int num, *ptr;
    num = 10;
    ptr = &num;

    printf("nNum = %d, &nNum = %d ₩n", num, &num);
    printf("*pNum = %d, pNum = %d, &pNum = %d₩n", *ptr, ptr, &ptr);

    return 0;
}
```



1. 포인터 (Pointer)

Example

```
num = 10;
```

…	…
…	…
…	…
…	…
…	…

1. 포인터 (Pointer)

Example

```
num = 10;
```

…	…
num	10
…	…
…	…
…	…

1. 포인터 (Pointer)

Example

ptr = #

변수에 할당된 메모리의 시작 주소값을 나타내는 연산자

0x1013

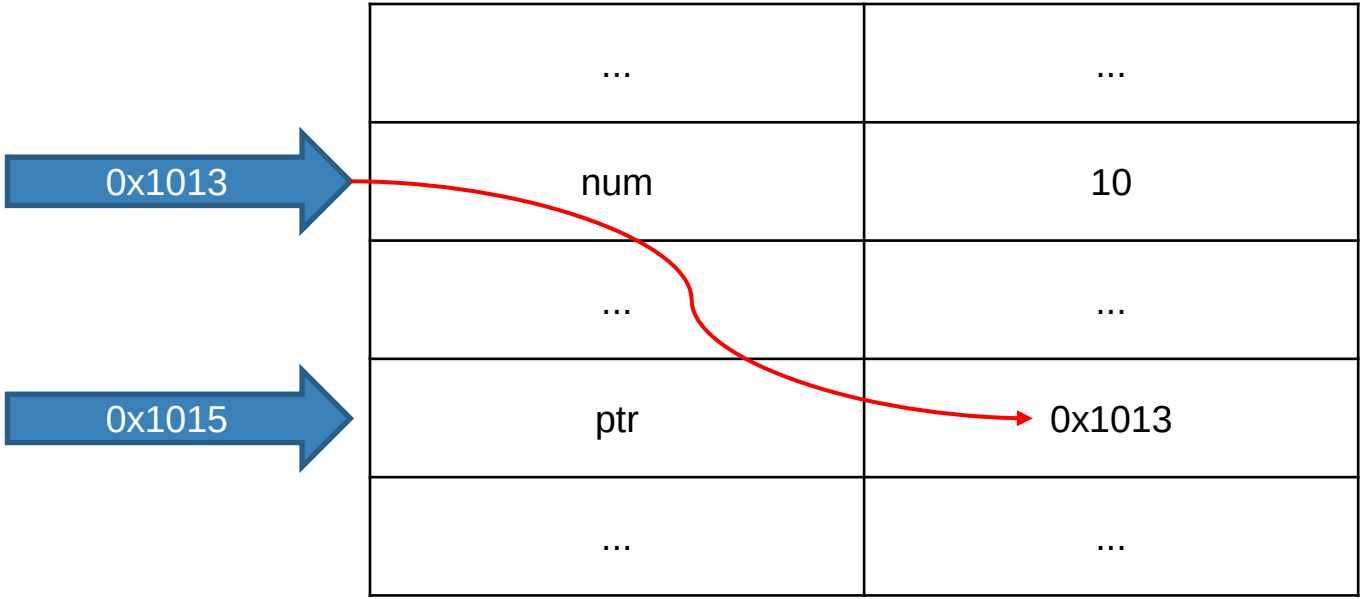
…	…
num	10
…	…
…	…
…	…

1. 포인터 (Pointer)

Example

ptr = #

변수에 할당된 메모리의 시작 주소값을 나타내는 연산자

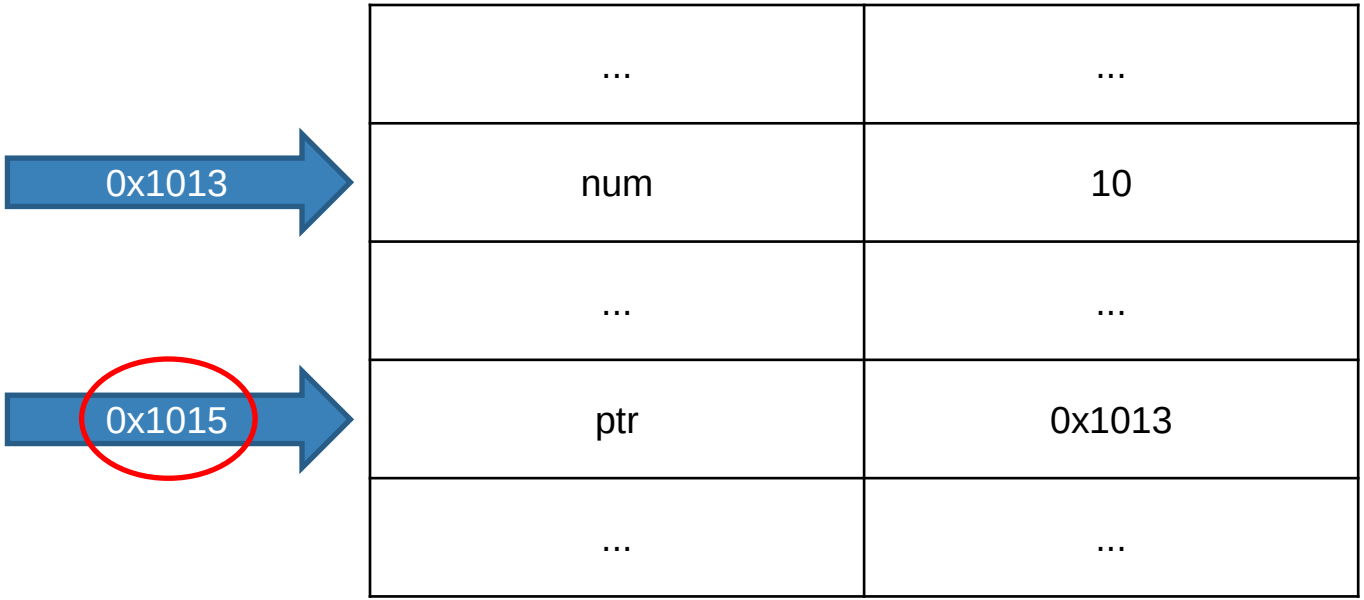


1. 포인터 (Pointer)

Example

&ptr

변수에 할당된 메모리의 시작 주소값을 나타내는 연산자

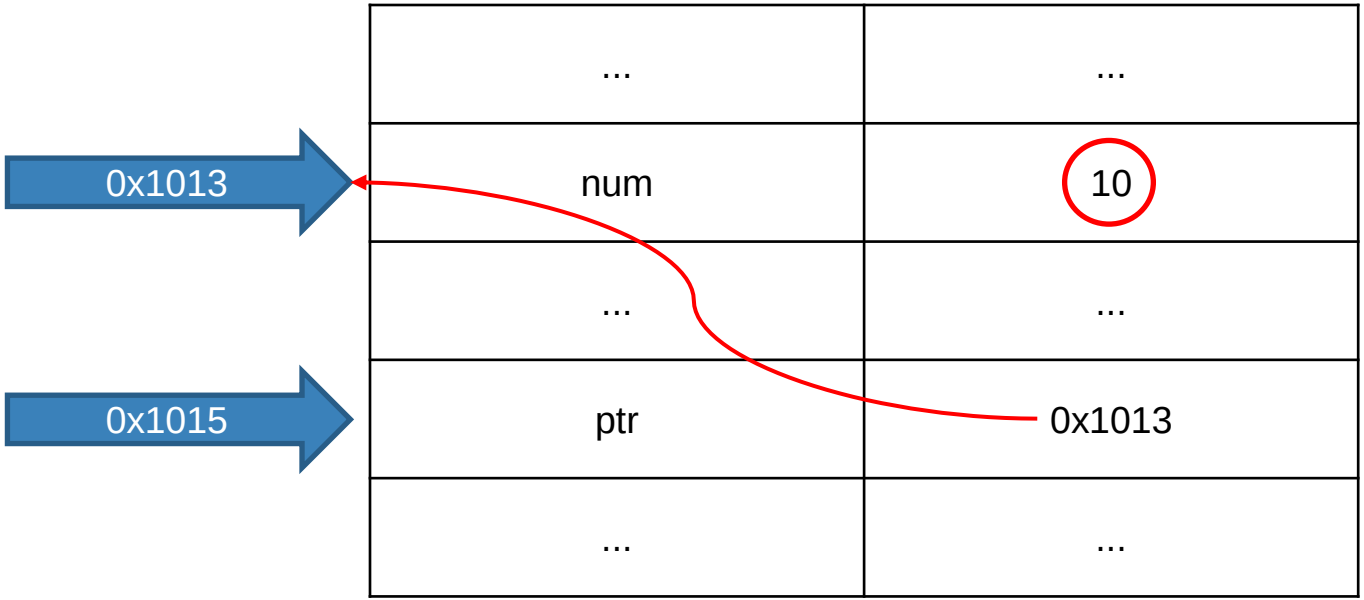


1. 포인터 (Pointer)

Example

*ptr

포인터 변수가 가리키는 곳의 내용(변수값)



1. 포인터 (Pointer)

코딩 시 주의할 몇 가지 팁

```
int *ptr;
*ptr = 100;              // 포인터 변수가 초기화되어 있지 않아 초기화 할 수 없다.
ptr = 100;               // 상수는 주소를 갖지 않기 때문에 포인터 변수로 참조할 수 없다.
```

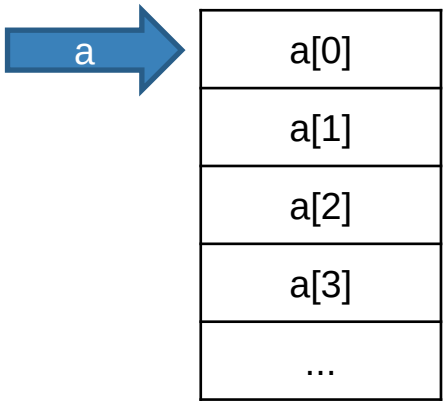
```
double dNum = 15.55;
int *pNum = &dNum;    // 포인터 자료형이 일치해야 한다.(권장)
```

```
int num;
*num = 100;              // 포인터 변수가 아닌 경우 *(메모리 접근 연산자)를 사용할 수 없다.
```

```
int *pNum = NULL;      //포인터 변수를 사용할 때는 NULL('\0')로 초기화 하는 것이 좋다.
int num;               // NULL: 아무 것도 가리키지 않는 포인터
if (pNum == '\0') pNum = &num;
else *pNum = 100;
```

1. 포인터 (Pointer)

- 포인터 연산
  - 포인터 값, 즉 주소 값을 대상으로 하는 연산
  - +,-,++,--를 이용하여 연산이 가능함
  - 포인터 연산을 통해 배열의 각 값에 접근할 수 있음



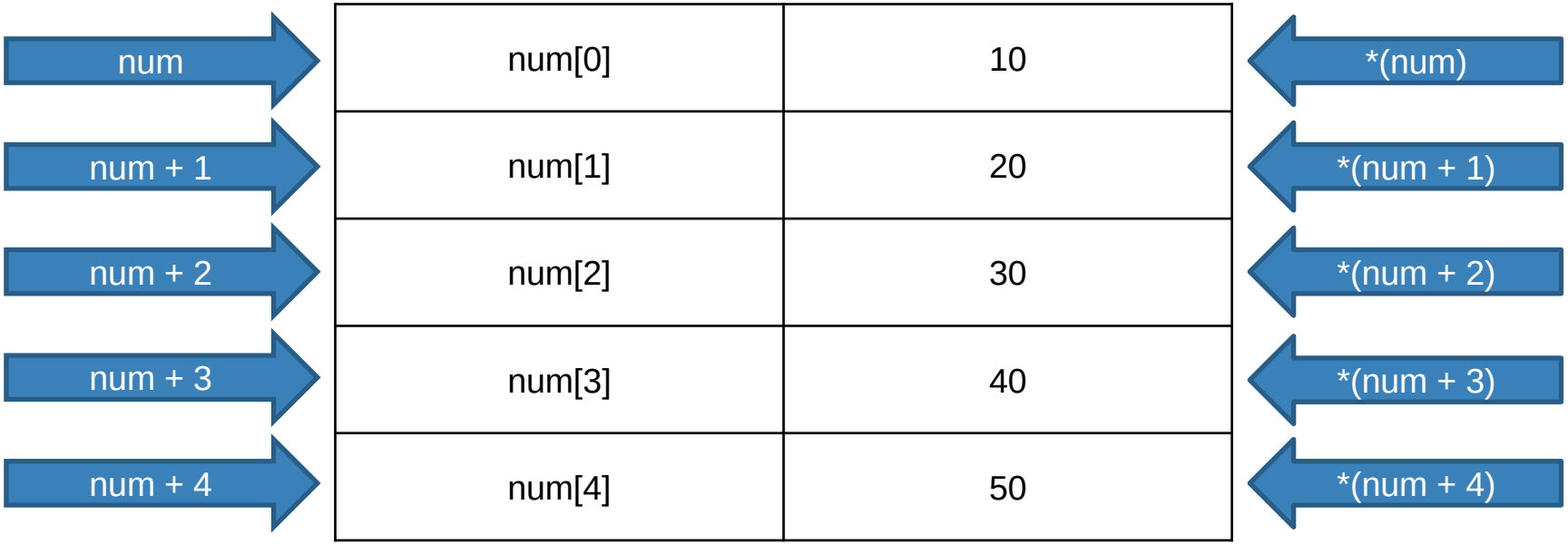
배열의 이름은 배열의 시작 주소와 같다.

- a = &a[0]
- *a = *&a[0] = a[0]

1. 포인터 (Pointer)

- 포인터 연산

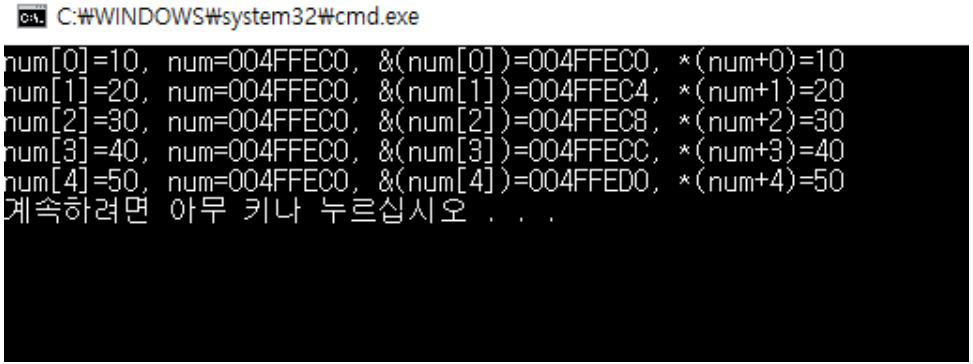
```
int num[5] = {10,20,30,40,50};
```



1. 포인터 (Pointer)

- 포인터 연산

```
int num[5] = {10,20,30,40,50};
```



2. 메모리 (Memory)

- 메모리 관련 함수 (#include <stdlib.h>)
  - void* **malloc** (size_t size);
    - size 만큼 메모리 블록을 할당한다.
    - return: a pointer to the beginning of the block
  - void* **free** (void* ptr);
    - 할당한 메모리를 해제한다.
  - void* **memset** (void* ptr, int val, size_t num);
    - ptr의 시작부터 num만큼 val로 메모리를 설정한다.
  - void* **memcpy** (void* dest, void* src, size_t size);
    - src의 메모리를 dest로 size만큼 복사

```
int *buffer = 0;

buffer = (int*)malloc(100 * sizeof(int));

free(buffer);
```

0x1010	...	
0x1011	buffer	0x1013
0x1012		
0x1013	(buffer[0])	
0x1014	(buffer[1])	
0x1015	(buffer[2])	
0x1016	(buffer[3])	
0x1017	...	

100개의 블록

2. 메모리 (Memory)

- 메모리 관련 함수를 이용한 동적 할당의 예

```
void main( )
{
    int a = 10;
    int numbers[a];
    //int numbers[10];
}
```

Error

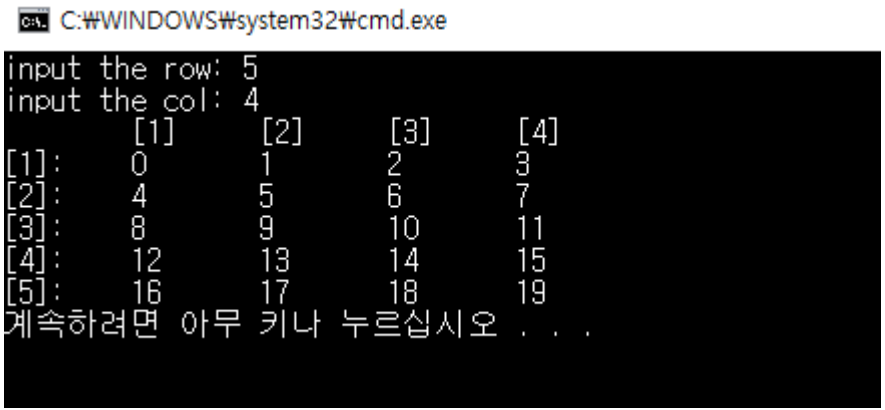
```
#include <stdio.h>
#include <stdlib.h>

int main( )
{
    int a,i;
    int * numbers;
    printf("input a: ");
    scanf("%d", &a);
    numbers = malloc(a * sizeof(int));

    for (i = 0; i < 10; i++)
        numbers[i] = i;

    for (i = 0; i < 10; i++)
        printf("numbers[%d]: %d₩n", i, numbers[i]);

    free(numbers);

    return 0;
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
input a: 10
numbers[0]: 0
numbers[1]: 1
numbers[2]: 2
numbers[3]: 3
numbers[4]: 4
numbers[5]: 5
numbers[6]: 6
numbers[7]: 7
numbers[8]: 8
numbers[9]: 9
계속하려면 아무 키나 누르십시오 . . .
```

2. 메모리 (Memory)

- 이중 포인터와 동적 할당을 이용한 2차원 배열 만들기




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

int main()
{
    int** matrix;
    int row, col, i, j;
    printf("input the row: ");
    scanf("%d", &row);
    printf("input the col: ");
    scanf("%d", &col);

    matrix = (int **)malloc(row * sizeof(int*));
    for (i = 0; i < row; i++)
        matrix[i] = (int*)malloc(col * sizeof(int));

    for (i = 0; i < row; i++)
        for (j = 0; j < col; j++)
            matrix[i][j] = i*col + j;

    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
            printf("%d₩t", matrix[i][j]);
        printf("₩n");
    }

    for (i = 0; i < row; i++)
        free(matrix[i]);
    free(matrix);

    return 0;
}
```

3. 구조체 (Structure)

- **정의: 하나 이상의 변수(포인터 포함)를 묶어서 새로운 자료형을 정의하는 도구**
- 구조체를 선언 및 정의하는 3가지 방법

```c
struct student {
    int id;
    char* name;
};


int main()
{
    struct student mike;

    return 0;
}
```

```c
struct student {
    int id;
    char* name;
};

typedef struct student STU;

int main()
{
    STU mike;

    return 0;
}
```

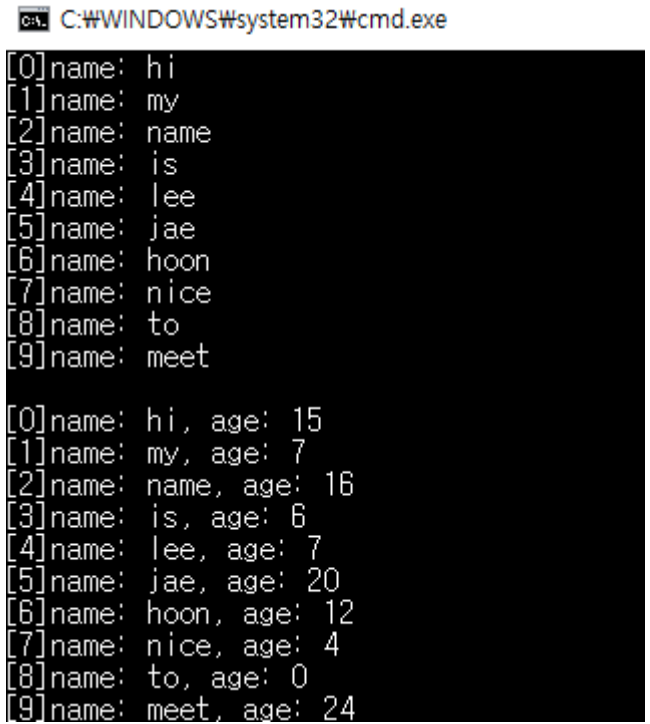
```c
typedef struct student {
    int id;
    char* name;
}STU;


int main()
{
    STU mike;

    return 0;
}
```

typedef : 사용자가 새로운 자료형을 정의

```c
typedef int INTEGER;

INTEGER num;
```

3. 구조체 (Structure)

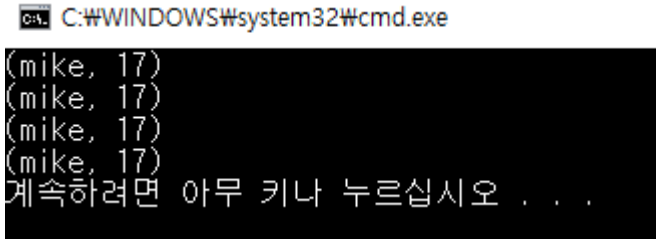
- 구조체를 활용한 동적할당의 예




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

typedef struct person {
    char* name;
    int age;
}PERSON;

int main()
{
    int size, k;
    PERSON *pPerson = NULL;
    char temp[256];
    size = 10;

    pPerson = (PERSON*)malloc(sizeof(PERSON)*size);

    for (k = 0; k < size; k++)
    {
        printf("[%d]name: ", k);
        scanf("%s", temp);

        pPerson[k].name = (char*)malloc(sizeof(temp));
        strcpy(pPerson[k].name, temp);

        pPerson[k].age = rand() % 26;
    }
    printf("₩n");
    for (k = 0; k < size; k++)
    {
        printf("[%d]name: %s, age: %d₩n", k, pPerson[k].name, pPerson[k].age);
    }
    for (k = 0; k < size; k++)
        free(pPerson[k].name);
    free(pPerson);
    return 0;
}
```

3. 구조체 (Structure)

- 구조체 멤버 변수에 접근하는 방법
  - `.` 연산자: 이름으로 구조체의 멤버 변수에 접근할 때
  - `->` 연산자: 주소로 구조체의 멤버 변수에 접근할 때

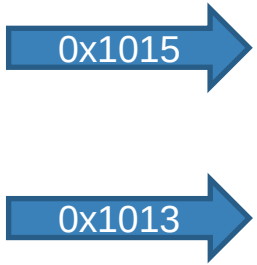



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct person {
    char* name;
    int age;
}PERSON;

int main()
{
    PERSON p1 = { "mike",17 };
    PERSON *p2 = &p1;
    PERSON **p3 = &p2;

    printf("(%s, %d)₩n", p1.name, p1.age);
    printf("(%s, %d)₩n", p2->name, p2->age);
    printf("(%s, %d)₩n", (*p2).name, (*p2).age);
    printf("(%s, %d)₩n", (*p3)->name, (*p3)->age);
}
```

3. 구조체 (Structure)

- 구조체 멤버 변수에 접근하는 방법

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

typedef struct person {
    char* name;
    int age;
}PERSON;

int main( )
{
    PERSON p1 = { "mike",17 };
    PERSON *p2 = &p1;
    PERSON **p3 = &p2;

    printf("(%s, %d)₩n", p1.name, p1.age);
    printf("(%s, %d)₩n", p2->name, p2->age);
    printf("(%s, %d)₩n", (*p2).name, (*p2).age);
    printf("(%s, %d)₩n", (*p3)->name, (*p3)->age);
}
```



...		...
p3		0x1015
...		...
p2		0x1013
...		...
p1	p1.name	“mike”
	p1.age	17

4. 재귀함수 (recursive function)

- 정의: 자기 자신을 호출하는 함수

```c
#include <stdio.h>

void Recursive()
{
    printf("Recersive call ₩n");
    //Recursive();
}

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

C:₩WINDOWS₩system32₩cmd.exe

```
Recersive call
계속하려면 아무 키나 누르십시오 . . .
```

```c
#include <stdio.h>

void Recursive()
{
    printf("Recersive call ₩n");
    Recursive();
}

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

C:₩WINDOWS₩system32₩cmd.exe

```
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
Recersive call
```

4. 재귀함수 (recursive function)

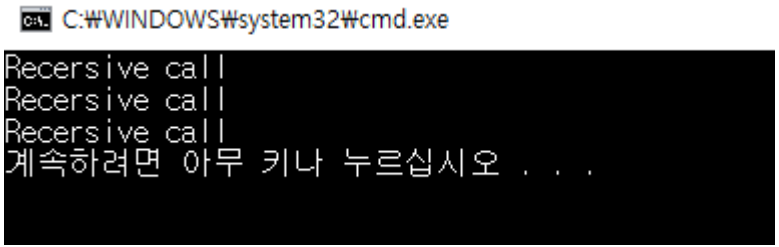
```
int main()
{
    Recursive(); ──→ void Recursive()
    return 0;        {
}                        printf("Recersive call ₩n");
                         Recursive(); ──→ void Recursive()
                     }                    {
                                              printf("Recersive call ₩n");
                                              Recursive(); ──→ void Recursive()
                                          }                    {
                                                                   printf("Recersive call ₩n");
                                                                   Recursive(); ──→ void Recursive()
                                                               }                    {
                                                                                        printf("Recersive call ₩n");
                                                                                        Recursive();
                                                                                    }
                                                                                    ↓
                                                                                void Recursive()
                                                                                {
                                                                                    printf("Recersive call ₩n");
                                                                                    Recursive();
                                                                                }
                                                                                ↓
                                                                                .....
```

infinite loop!

4. 재귀함수 (recursive function)

원하는 횟수 만큼만 실행 되는 재귀함수를 만들려면?

```c
#include <stdio.h>

void Recursive(int n)
{
    printf("Recersive call ₩n");
    if (n == 1)
        return;
    Recursive(n-1);
}

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



4. 재귀함수 (recursive function)

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

인자 3 전달

```
void Recursive(int n)
{
    printf("Recersive call \n");
    if (n == 1)
        return;
    Recursive(n-1);
}
```

인자 2 전달

```
void Recursive(int n)
{
    printf("Recersive call \n");
    if (n == 1)
        return;
    Recursive(n-1);
}
```

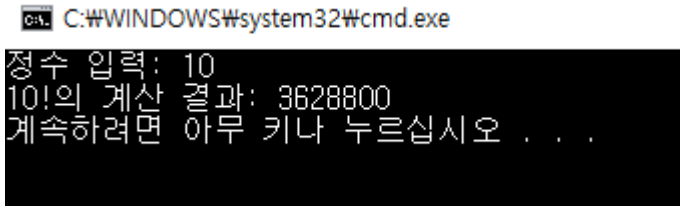
인자 1 전달

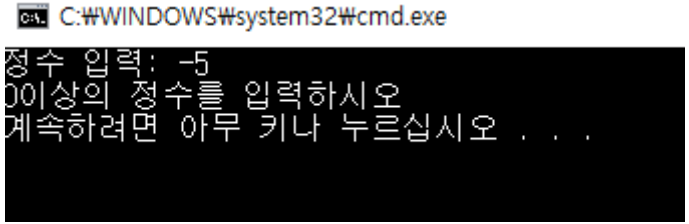
```
void Recursive(int n)
{
    printf("Recersive call \n");
    if (n == 1)
        return;
    Recursive(n-1);
}
```

Return

4. 재귀함수 (recursive function)

- 재귀함수를 이용한 팩토리얼(factorial) 계산



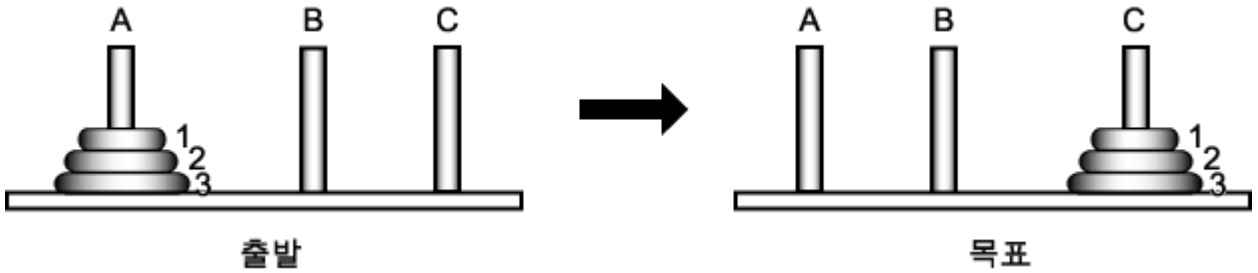



```c
#include <stdio.h>

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

int main()
{
    int val;
    int result;

    printf("정수 입력: ");
    scanf("%d", &val);
    if (val < 0)
    {
        printf("0이상의 정수를 입력하시오₩n");
        return -1;
    }

    result = factorial(val);
    printf("%d!의 계산 결과: %d₩n", val, result);
    return 0;
}
```

5. 하노이 탑(Hanoi tower)

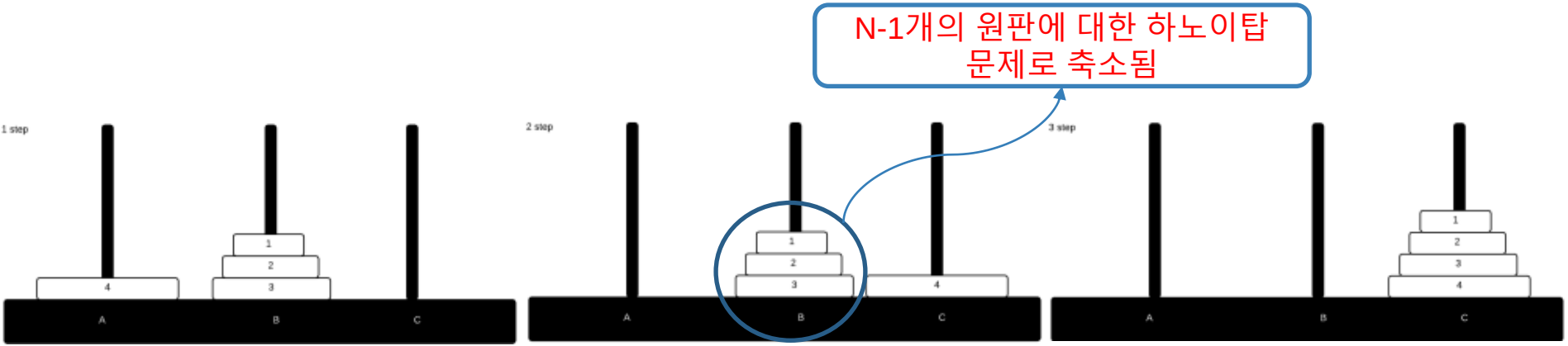
- 세 개의 기둥과 다양한 크기의 원판이 존재
- 초기 상태: 한 기둥(A)에 작은 원판들이 위로 오도록 순서대로 쌓여 있다.
- 목표 상태: 다른 기둥(C)로 원판들을 모두 옮겨야 하며 이 때 두 가지 조건이 만족되어야 한다.
  - **한번에 하나의 원판만 옮길 수 있다.**
  - **큰 원판은 항상 작은 원판 아래에 있어야 한다.**

⇒ Recursive를 이용해 풀 수 있는 대표적인 문제이다.



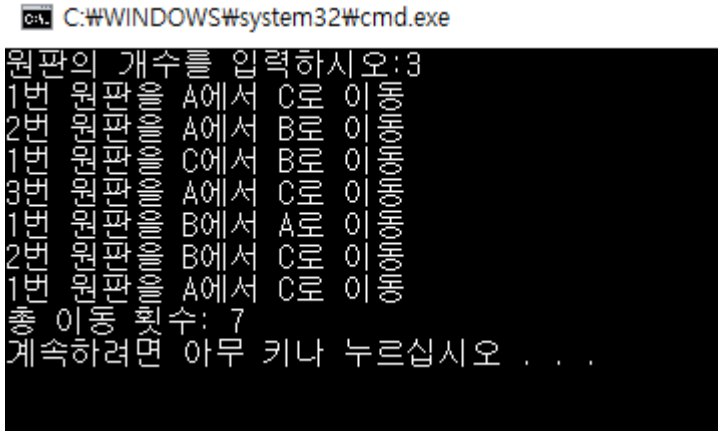
5. 하노이 탑(Hanoi tower)

- 기본 원리
  - A(출발 막대), B(중간 막대), C(목표 막대), 1~N개의 원판(1이 가장 작은 원판, N이 가장 큰 원판)
  - A에서 C로 모든 원판을 옮기기 위한 기본적인 방법
    - **A에서 B로 1~(N-1)번 원판을 옮긴다.**
    - **A에서 C로 N번 원판을 옮긴다.**
    - **B에서 C로 1~(N-1)번 원판을 옮긴다.**



5. 하노이 탑(Hanoi tower)

- 과제
  - 하노이의 탑 구현하기
    - hanoi 함수의 인자는 임의로 변경이 가능하나 아래의 출력은 동일해야 함




```
int main()
{
    int n;
    printf("원판의 개수를 입력하시오:");
    scanf("%d", &n);

    hanoi(n, 'A', 'B', 'C');
}

void hanoi(int n, char from, char middle, char to)
{

}
```

총 이동 횟수 출력

재귀함수를 활용하여 구현

왼쪽 출력 화면처럼
이동 상황 출력

감사합니다.

과제 제출 기한: 2025년 3월 25일 23:59분 (LMS 제출 시간 기준)

제출형식:

- 소스코드의 이름을 `hanoi_tower.c`로 작성하고 LMS 과제 탭에 제출
- 과제 제출 탭은 추후 생성 예정

궁금한 것이 생기면 언제든지 질문하시면 됩니다 ☺

- 공업센터본관 304호로 방문하시거나
- jeongiun@hanyang.ac.kr 로 연락 바랍니다.
- 담당조교: 이범기