
포인터

- Chapter 07 -

학습목차

- I. 운영체제의 메모리 관리 방식
- II. 포인터 개요
- III. 포인터 변수의 선언과 사용
- IV. 다차원 포인터 변수의 선언과 사용
- V. 포인터가 필요한 경우

운영체제의 메모리 관리 방식

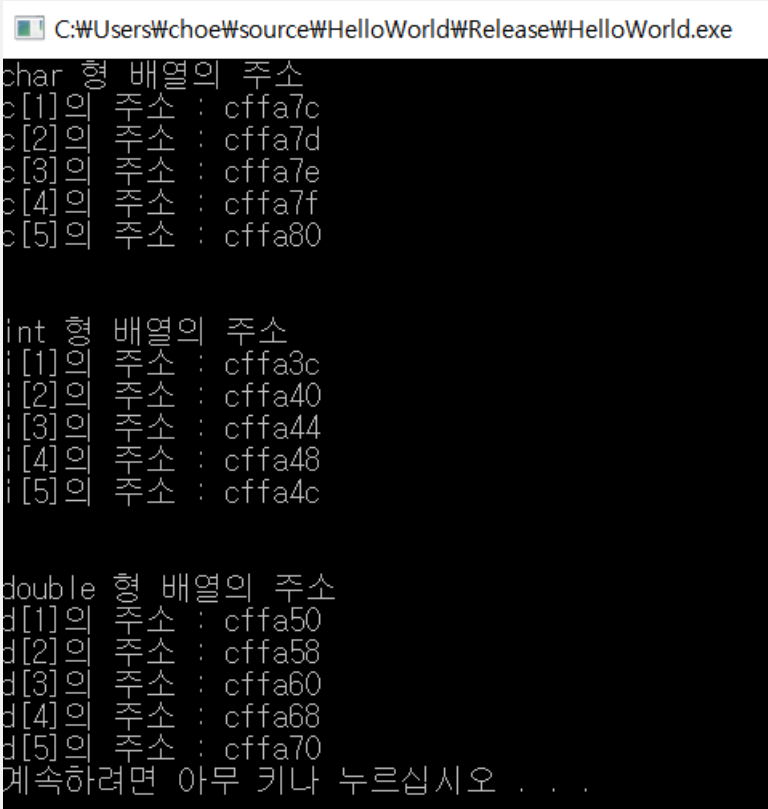
▶ 배열의 주소 출력하기

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    char c[5];
    int i[5];
    double d[5];
    int j;
    printf("char 형 배열의 주소\n\n\n");

    for (j = 0; j < 5; j++) {
        printf("c[%d]의 주소 : %x\n", j + 1, &c[j]);
    }
    printf("int 형 배열의 주소\n\n\n");

    for (j = 0; j < 5; j++) {
        printf("i[%d]의 주소 : %x\n", j + 1, &i[j]);
    }
    printf("double 형 배열의 주소\n\n\n");

    for (j = 0; j < 5; j++) {
        printf("d[%d]의 주소 : %x\n", j + 1, &d[j]);
    }
    system("pause");
}
```



```
C:\Users\choe\source\HelloWorld\Release\HelloWorld.exe
char 형 배열의 주소
c[1]의 주소 : cffa7c
c[2]의 주소 : cffa7d
c[3]의 주소 : cffa7e
c[4]의 주소 : cffa7f
c[5]의 주소 : cffa80

int 형 배열의 주소
i[1]의 주소 : cffa3c
i[2]의 주소 : cffa40
i[3]의 주소 : cffa44
i[4]의 주소 : cffa48
i[5]의 주소 : cffa4c

double 형 배열의 주소
d[1]의 주소 : cffa50
d[2]의 주소 : cffa58
d[3]의 주소 : cffa60
d[4]의 주소 : cffa68
d[5]의 주소 : cffa70
계속하려면 아무 키나 누르십시오 . . .
```

운영체제의 메모리 관리 방식

▶ 변수명과 주소

- ▶ 현재까지 학습한 프로그래밍 작성방법에서는 변수를 이용하여 대입하고 참조하는 방식으로 메모리에 데이터를 저장하거나 읽음
 - ▷ 복잡한 메모리 주소에 이름을 붙여서 사용하는 것이 프로그래밍을 할 경우 효율적임
 - ▷ 기계어에서는 변수 이름보다 변수가 위치한 메모리의 주소가 중요
 - ▷ 변수의 이름을 사용하지 않더라도 주소만 알고 있으면 값을 읽거나 변경 가능



운영체제의 메모리 관리 방식

▶ 운영체제(OS : Operating System)

- ▶ 컴퓨터 시스템을 효과적으로 관리해 주고 시스템이 가지고 있는 자원을 사용자나 프로그램이 활용할 수 있도록 서비스를 제공
 - ▷ 예) 윈도우, 리눅스, 맥OS, 안드로이드 등 ...
- ▶ 컴퓨터 시스템의 메모리는 운영체제가 관리함
- ▶ 하드웨어의 발전으로 64비트 시스템의 사용이 늘고 있으며 이에 따라 32비트 프로그램과 64비트 프로그램으로 나누어서 개발 가능
 - ▷ 비주얼 스튜디오도 32/64비트 용으로 구분
 - ▷ 그러나 특별한 경우가 아니고서는 32비트 프로그램을 개발
 - ▷ 32비트 프로그램은 64비트 운영체제에서도 동작함
 - ▷ 비트는 메모리와 관계가 있음
 - ▷ 32비트 운영체제는 최대 4GB, 64비트 운영체제는 최대 16GB까지 메모리를 인식
 - 32비트로 나타낼 수 있는 양수 데이터의 범위 $0 \sim 4,294,967,295$ ($4,294,967,296$ 개 = 2^{32})
 - 운영체제는 메모리의 주소를 1byte 단위로 관리($4GB = 4,294,967,296\text{Byte}$)

운영체제의 메모리 관리 방식

▶ CPU의 비트 수

- ▶ 64비트 컴퓨터는 32비트 운영체제에서도 동작할 수 있으며 이 경우 32비트 컴퓨터처럼 움직임
- ▶ CPU가 한 번에 처리하는 메모리의 2진수 자릿수가 비트 수
 - ▷ CPU가 몇 비트? 라고 말하는 것은 대부분 이런 의미
 - ▷ 특수한 명령으로 그 이상의 비트 수를 한꺼번에 계산하는 CPU도 존재

운영체제의 메모리 관리 방식

▶ 메모리 상의 주소 지정 방식

▶ 메모리는 각 자리에 주소(address)가 할당되어 있음

▷ 아파트의 집집마다 각 동과 호수가 존재하는 것과 같은 개념

▷ 메모리는 사용하려면 사용할 주소를 지정

▶ OS는 메모리 주소를 1바이트 단위로 관리

▷ 윈도우 NT 운영체제의 경우 **0 ~ 4,294,967,295($2^{32} - 1$)번지**까지 1바이트 단위로 주소가 매겨짐

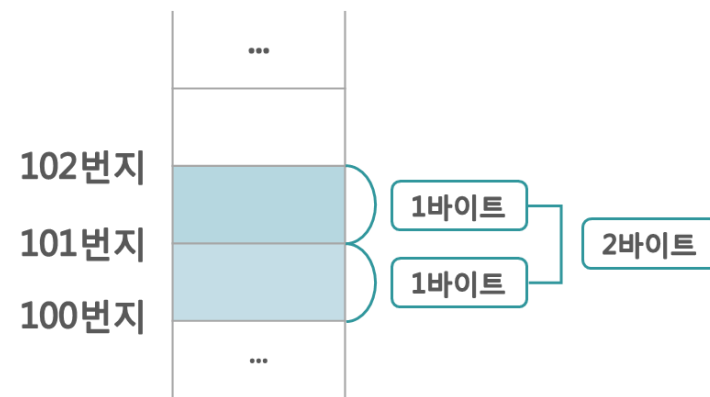
▷ 운영체제마다 메모리 관리 방식이 다름

▶ 주소를 통하여 메모리를 참조하거나 저장할 경우에 사용할 크기를 명시

▷ 다양한 타입의 저장공간이 필요하기 때문

▷ char는 1바이트, int는 4바이트

▷ 예) 100번지, 101번지를 사용하고 싶으면 100번지 부터 2개의 바이트를 사용한다고 명시



운영체제의 메모리 관리 방식

▶ 직접 주소 지정 방식

▶ 프로그래머가 메모리를 사용할 경우, 메모리 주소를 직접 적는 방식

▶ 예) “102번지에 1042라는 값을 2바이트 크기로 지정하겠다.”

▷ 102번지라는 주소를 직접적으로 표기

▷ 1042는 두 개의 바이트에 4와 12로 각각 나뉘어 저장됨

▷ 10진수 1042는 16진수로 412

▷ $1042 = 0000\ 0100\ 0001\ 0010$

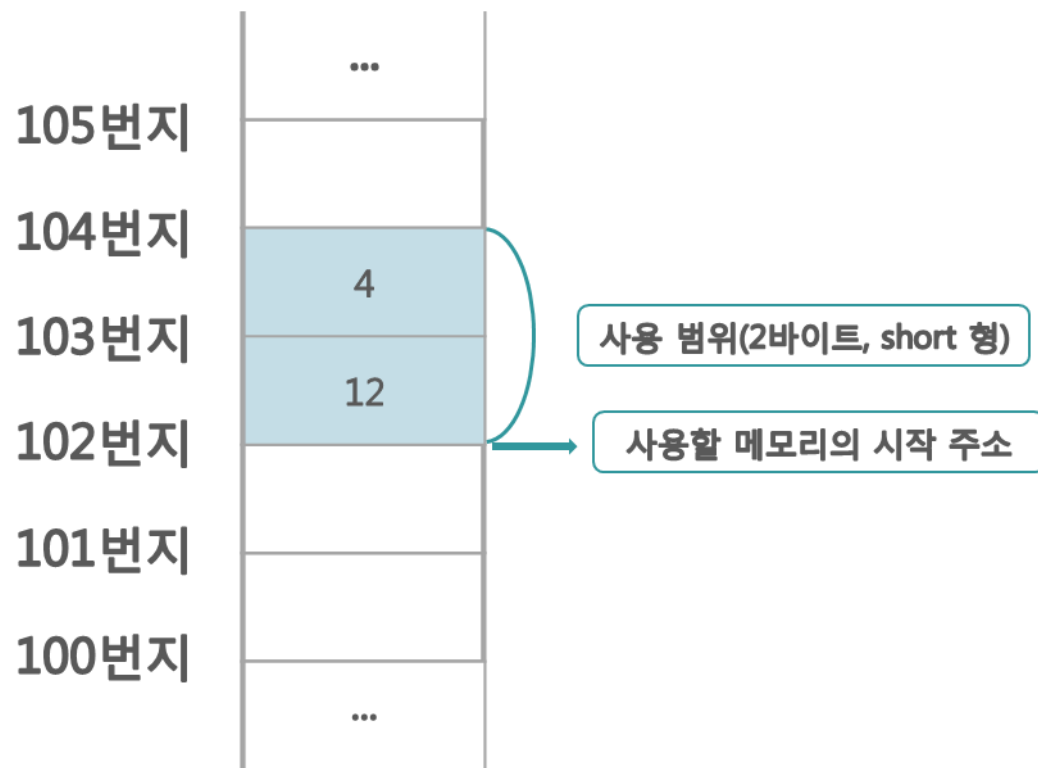
▷ C언어에서는 2진수를 직접 사용하는 방법을 제공하지 않음

▷ 2진수에 가장 가까운 표현법인 16진수를 사용

▷ 윈도우를 비롯한 대부분 운영체제는 리틀엔디언(Little Endian)방식으로 정렬

- 작은 단위가 앞으로 옴

▷ 리틀 앤디언은 시작 주소부터 하위 바이트를 기록



운영체제의 메모리 관리 방식

▶ 직접 주소 지정 방식

- ▶ 앞의 예제 “102번지에 1042라는 값을 2바이트 크기로 지정하겠다.”는
- ▶ 컴파일 후에 “0x00000066번지에 0x0412값을 2바이트 크기로 지정하겠다.”로 변경
 - ▷ 주소도 16진수로 표기하면 효율적임 - 10진수 102는 16진수로 66
 - ▷ $4,294,967,295 = 0xffffffff$
- ▶ C언어는 변수라는 개념을 통한 직접 주소 지정 방식을 사용
 - ▷ 컴파일러에 의해 변수가 주소로 변환되어 결과적으로는 직접 주소 지정 방식을 사용
 - ▷ 주소를 직접 사용하는 것 보다 가독성이 높고 실수를 줄일 수 있음

운영체제의 메모리 관리 방식

▶ 직접 주소 지정 방식의 한계

- ▶ 함수 내부에 선언된 변수는 해당 함수에서만 사용 가능
- ▶ 다른 함수에 선언한 변수가 메모리 내부에 존재해도 문법적으로 접근 불가

```
#include <stdio.h>
void Test( )
{
    short soft = 0x0000;
    soft = tips; /* 오류 */
}

void main( )
{
    short tips = 0x0005;
    Test( );
}
```

운영체제의 메모리 관리 방식

▶ 간접 주소 지정 방식

▶ 메모리의 특정 공간에 주소를 저장하고 있다가 해당 주소로 데이터를 전달하거나 참조하는 방식

▶ 현실 세계의 간접 주소 지정 방식

▷ 택배 배송원이 하는 일

▷ 배송 시 : 고객의 주소를 저장하고 있다가 해당 주소로 물건을 전달하는 일

▷ 반송 시 : 고객의 주소를 저장하고 있다가 해당 주소로 물건을 받아가는 일

▷ 우리에게 '간접 접근' 서비스를 제공

▷ 택배 배송원이 있어서 우리가 편리한 점

▷ 물품을 수령하기 위해 직접 구매처를 방문 하지 않아도 됨

▷ 반품을 위해 직접 구매처를 방문 하지 않아도 됨

▷ 컴퓨팅 세계에서 변수는 택배 배송원과 같은 역할

▷ 포인터 변수(포인터라고도 부름)

포인터 개요

▶ 포인터 변수(또는 포인터)

- ▶ Pointer는 가리킨다는 뜻의 Point에 er을 붙인 것으로 ‘가리키는 것’이라는 뜻
- ▶ 변수의 주소를 저장하는 변수
 - ▷ 기존의 변수는 데이터를 저장하지만 포인터는 메모리 상의 다른 변수의 주소를 저장
- ▶ 포인터에 변수의 주소를 저장하려면 해당 변수명 앞에 & 연산자를 사용(주소 연산자)
- ▶ 주소에 저장되어 있는 값을 참조하려면 주소 앞에 * 연산자를 사용(간접 참조 연산자)

```
int main(void) {  
    int a = 10;  
    int* pa = &a;  
  
    printf("a의 주소 : %p\n", &a);  
    printf("pa에 저장된 주소에 위치한 값 : %d\n", *pa);  
  
    return 0;  
}
```

a의 주소 : 010FFEAC
pa에 저장된 주소에 위치한 값 : 10

포인터 변수의 선언과 사용

▶ 포인터 변수의 선언

- ▶ 포인터 변수도 지금까지 배운 변수와 동일하게 선언하고 사용
- ▶ 자료형: 포인터 변수의 자료형을 지정, 자료형 다음에 * 연산자를 붙임
- ▶ 포인터 변수 이름: 주소를 저장할 변수의 이름 지정
- ▶ NULL 포인터 설정: 포인터 변수 선언 시 NULL로 초기화
 - ▶ NULL 포인터는 포인터 변수에 아무 주소도 저장하지 않을 경우(NULL을 대문자로 사용)

①
자료형
↓
int*

②
포인터 변수 이름
↓
pointer

③
NULL 포인터 설정
↓
=NULL;

▶ 예)

`int* p1 = NULL;` // int형 주소를 저장하는 포인터 변수

`char* p2 = NULL;` // char형 주소를 저장하는 포인터 변수

`double* p3 = NULL;` // double형 주소를 저장하는 포인터 변수

다차원 포인터 변수의 선언과 사용

▶ 주소 연산자(&)

▶ 프로그램이 사용하는 메모리에는 바이트 별로 주소 값 존재

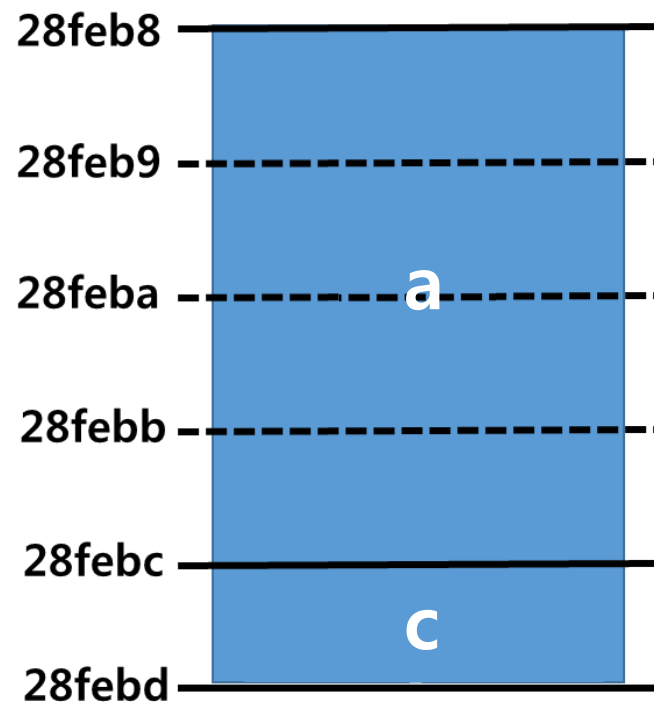
▷ 0부터 시작하고 바이트 단위로 1씩 증가

▷ 2바이트 이상의 크기를 갖는 변수는 여러 개의 주소 값에 걸쳐 할당

▷ int형 변수 a가 메모리 100번지부터 할당된다면 100번지부터 103번지까지 4바이트에 걸쳐 할당된다는 의미

```
int a;  
char c;  
int *pa = &a;  
int *pc = &c;  
printf("a의 주소 : %x %x\\n", &a, pa);  
printf("c의 주소 : %x %x\\n", &c, pc);
```

```
a의 주소 : 28feb8 28feb8  
c의 주소 : 28febc 28febc
```



포인터 변수의 선언과 사용

▶ 포인터 변수의 초기화

▶ 포인터 변수의 선언과 초기화를 개별적으로 수행

```
int main(void)
{
    int num=10;
    int* ip=NULL;
    ip=&num;
    return 0;
}
```

▶ 포인터 변수의 선언과 초기화를 동시에 수행

```
int main(void)
{
    int num=10;
    int* ip=&num;
    return 0;
}
```

포인터 변수의 선언과 사용

▶ 포인터 변수 실습

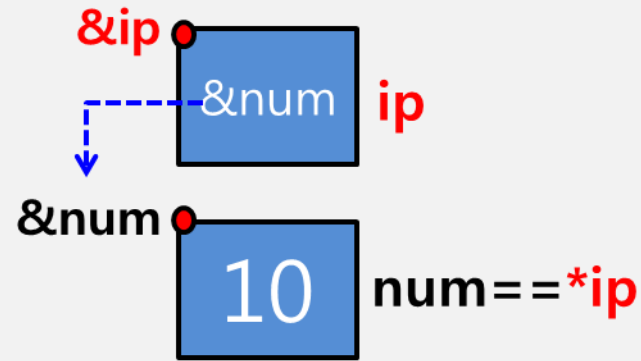
▶ 주소 연산자(&)와 간접 참조 연산자(*)

▶ *와 &는 서로 상쇄

```
#include <stdio.h>
int main( )
{
    int num=10;
    int* ip=NULL;    // 포인터 변수 선언

    ip=&num;         // 주소 저장

    printf("%x %x %d \n", &ip, ip, *ip);
    printf("%x %x %d \n", &*ip, *ip, **&ip);
    return 0;
}
```



```
28feb8 28febc 10
28feb8 28febc 10
```


포인터 변수의 선언과 사용

▶ 포인터 변수 실습

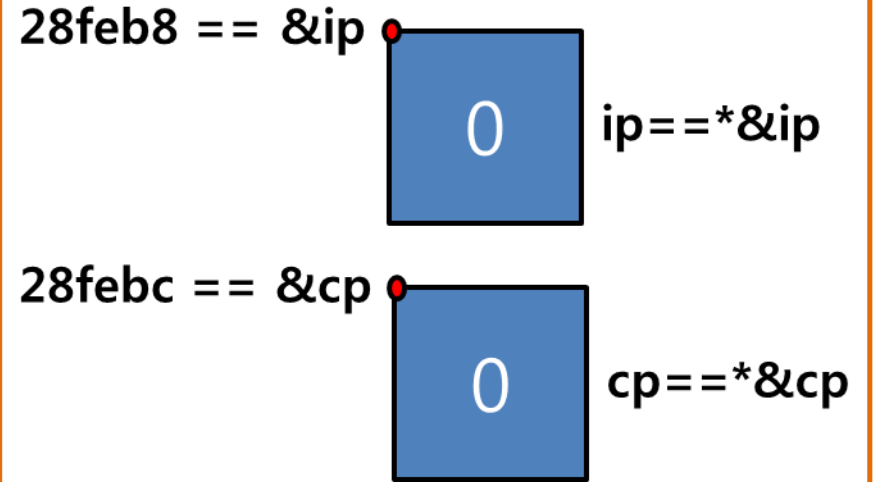
- ▶ 포인터 변수의 주소 확인 및 접근
- ▶ 각 타입별 포인터 변수의 크기 확인
- ▶ 32비트 프로그램에서는 4바이트

```
#include <stdio.h>
int main(void)
{
    // 포인터 변수 선언
    char* cp=NULL;
    int* ip=NULL;

    printf("%x %x %x\n", &cp, cp, *&cp);
    printf("%x %x %x\n", &ip, ip, *&ip);

    printf("%d %d\n", sizeof(char*), sizeof(int*));
    printf("%d %d\n", sizeof(cp), sizeof(ip));
    return 0;
}
```

```
// 4, 4 출력
// 4, 4 출력
```



```
28feb8c 0 0
28feb8 0 0
4 4
4 4
```

포인터 변수의 선언과 사용

▶ 앞의 예제 분석

- ▶ 포인터 변수도 변수이므로 메모리 어딘가에 위치하며 주소를 갖고 있음
- ▶ NULL은 ASCII값으로 0
 - ▷ NULL로 초기화 했기 때문에 0이 출력
- ▶ 대부분의 시스템에서 모든 포인터 변수의 크기는 4byte
 - ▷ 64비트 프로그램보다는 32비트 프로그램이 더 많이 개발되고 사용됨
- ▶ 다양한 타입의 포인터가 있는 이유
 - ▷ 포인터에 저장된 주소로 변수를 참조하는 경우, 몇 바이트씩 참조해야 하는지 알려주기 위함
 - ▷ char*형 포인터 변수 cp는 1바이트씩 참조
 - ▷ int*형 포인터 변수 ip는 4바이트씩 주소를 참조



포인터 변수의 선언과 사용

▶ 포인터 변수 실습

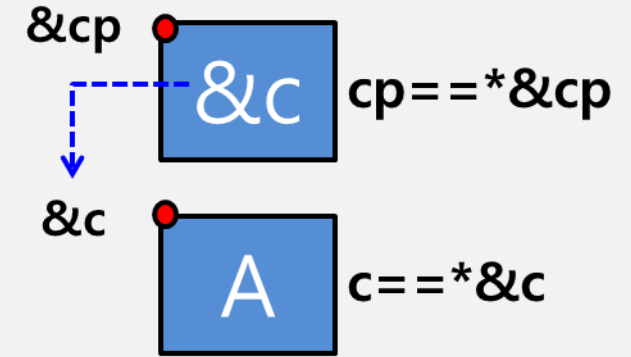
▶ 포인터 변수에 변수의 주소를 저장

```
#include <stdio.h>
int main( )
{
    char c='A';
    char* cp=NULL;           //선언과 동시에 주소 저장 가능

    cp=&c; // 주소 저장

    printf("%x %c %c \n", &c, c, *&c);
    printf("%x %x %x \n", &cp, cp, *&cp);

    printf("%c \n", c);       // 직접 접근
    printf("%c \n", *cp);     // 간접 접근
    return 0;
}
```



같은 메모리 공간의 이름
`c == *&c == *cp`

```
28feb8  A  A
28feb8  28feb8  28feb8
A
A
```

포인터 변수의 선언과 사용

▶ 포인터 변수 실습

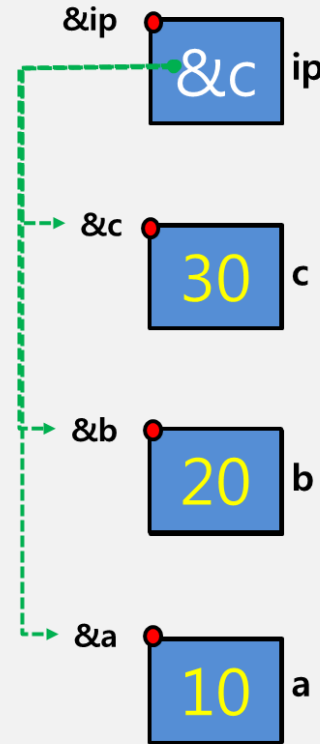
▶ 포인터 변수를 이용하여 변수 값 변경하기

```
int a=0, b=0, c=0;
int* ip=NULL;    // 포인터 변수 선언

ip=&a;            // 주소 저장
*ip=10;
printf("%d %d %d %d\n", a, b, c, *ip);

ip=&b;            // 주소 저장 변경
*ip=20;
printf("%d %d %d %d\n", a, b, c, *ip);

ip=&c;            // 주소 저장 변경
*ip=30;
printf("%d %d %d %d\n", a, b, c, *ip);
```



10	00	00	10
10	20	00	20
10	20	30	30

포인터 변수의 선언과 사용

▶ 잘못 사용된 포인터

▶ 포인터 변수에 주소를 저장하지 않고 연산하는 경우

```
#include <stdio.h>
int main(void)
{
    int* ip=NULL;
    *ip=10000;
    return 0;
}
```

▶ 포인터 변수에 이상한 주소 저장

```
#include <stdio.h>
int main(void)
{
    int* ip=14592343;
    *ip=1020;
    return 0;
}
```

다차원 포인터 변수의 선언과 사용

▶ 다차원 포인터 변수

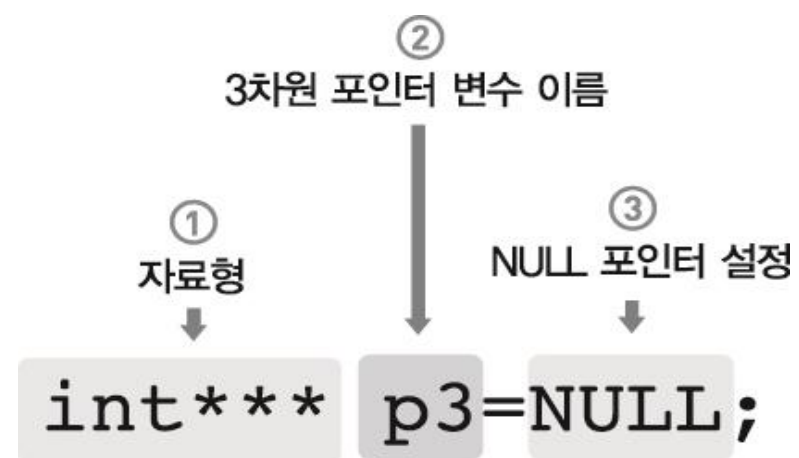
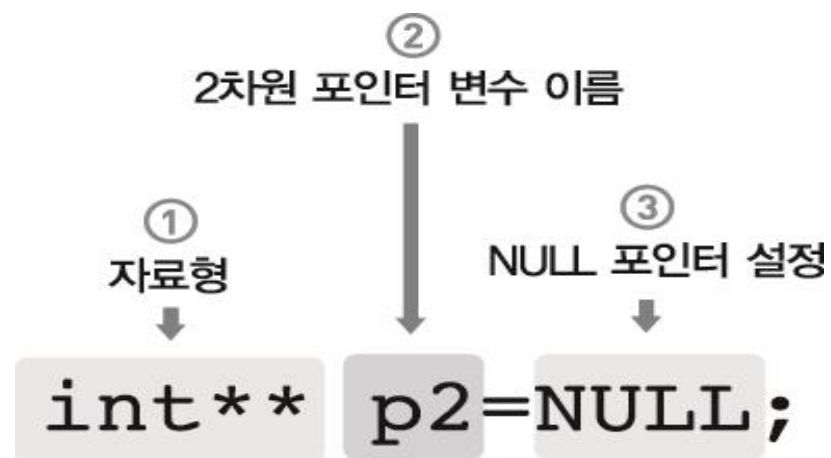
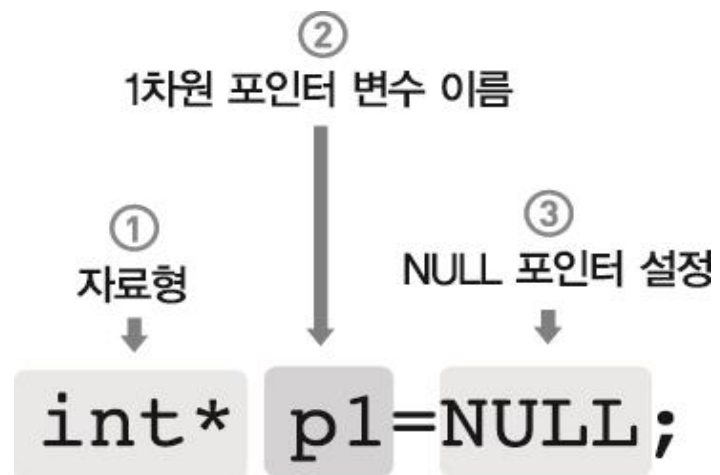
▶ 포인터 변수의 주소를 저장하는 변수

▶ 2차원 이상의 포인터 변수를 의미

▷ 1차원 포인터 변수 : 일반 변수의 주소를 저장 및 참조

▷ 2차원 포인터 변수 : 1차원 포인터의 주소를 저장 및 참조

▷ 3차원 포인터 변수 : 2차원 포인터의 주소를 저장 및 참조

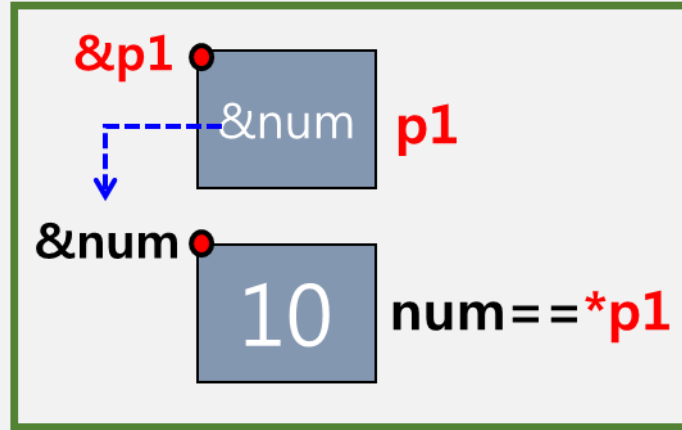


다차원 포인터 변수의 선언과 사용

▶ 1차원 포인터 변수의 역할

▶ 일반 변수의 주소를 저장

```
#include <stdio.h>
int main(void)
{
    int num=10;
    int* p1=NULL;
    p1=&num;
    return 0;
}
```



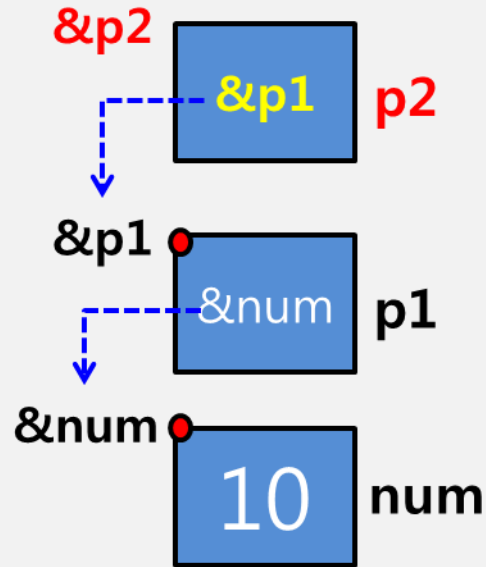
다차원 포인터 변수의 선언과 사용

▶ 2차원 포인터 변수의 역할

▶ 1차원 포인터 변수의 주소를 저장

```
#include <stdio.h>
int main(void)
{
    int num=10;
    int* p1=NULL;
    int** p2=NULL;

    p1=&num;
    p2=&p1;
    return 0;
}
```



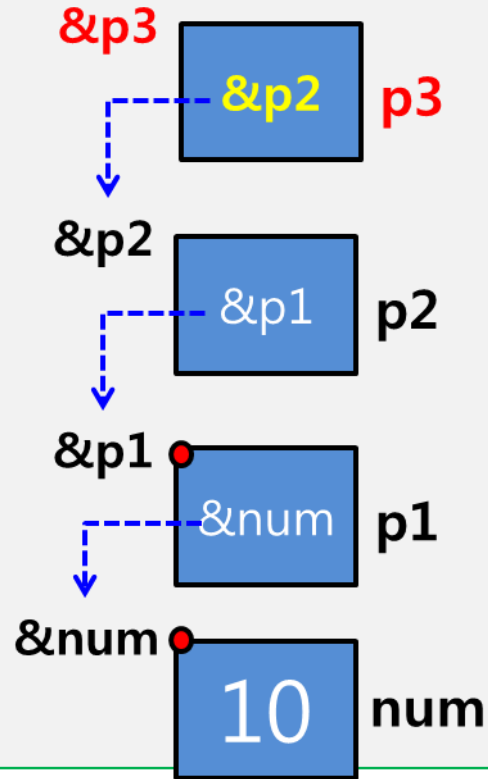
다차원 포인터 변수의 선언과 사용

▶ 3차원 포인터 변수의 역할

▶ 2차원 포인터 변수의 주소를 저장

```
#include <stdio.h>
int main(void)
{
    int num=10;
    int*  p1=NULL;
    int** p2=NULL;
    int*** p3=NULL;

    p1=&num;
    p2=&p1;
    p3=&p2;
    return 0;
}
```



다차원 포인터 변수의 선언과 사용

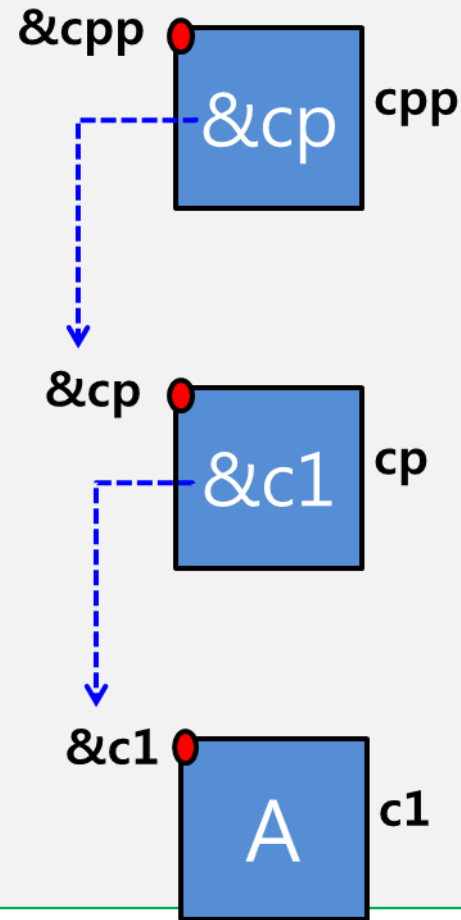
▶ 다차원 포인터 실습

```
#include <stdio.h>
int main( )
{
    char c1='A';
    char* cp=NULL;
    char** cpp=NULL;

    cp=&c1;
    cpp=&cp;

    printf("%c %x %x %n", c1, cp, cpp);
    printf("%x %x %x %n", &c1, &cp, &cpp);
    printf("%c %c %c %n", c1, *cp,**cpp);

    return 0;
}
```



다차원 포인터 변수의 선언과 사용

▶ 다차원 포인터 실습

▶ 직접 확인하기

```
int num1=10;
int* ip1=NULL;
int** ip2=NULL;
int*** ip3=NULL;
ip1=&num1;
ip2=&ip1;
ip3=&ip2;
printf("%d %d %d %d \n", num1, *ip1, **ip2, ***ip3);
printf("%x %x %x %x \n", &num1, ip1, *ip2, **ip3);
printf("%x %x %x \n", &ip1, ip2, *ip3);
printf("%x %x \n", &ip2, ip3);
```

```
printf("%d %d \n", sizeof(int), sizeof(int*));
printf("%d %d \n", sizeof(int**), sizeof(int***));
```

```
printf("%d %d \n", sizeof(num1), sizeof(ip1));
printf("%d %d \n", sizeof(ip2), sizeof(ip3));
```

```
10 10 10 10
affce0 affce0 affce0 affce0
affce4 affce4 affce4
affce8 affce8
4 4
4 4
4 4
4 4
```

포인터가 필요한 경우

▶ 직접 주소 지정 방식의 한계

- ▶ 함수 내부에 선언된 변수는 해당 함수에서만 사용 가능
- ▶ 다른 함수에 선언한 변수가 메모리 내부에 존재해도 문법적으로 접근 불가

```
#include <stdio.h>
void Test( )
{
    short soft = 0x0000;
    soft = tips; /* 오류 */
}

void main( )
{
    short tips = 0x0005;
    Test( );
}
```

포인터가 필요한 경우

▶ 포인터가 필요한 경우

- ▶ 전역 변수와 정적 변수를 사용하지 않고 서로 다른 함수에서 동일한 변수에 접근

```
void swap(int* pa, int* pb);
int main(void)
{
    int a = 10, b = 20;
    printf("a : %d, b : %d\n", a, b);
    swap(&a, &b);
    printf("a : %d, b : %d\n", a, b);
    return 0;
}
void swap(int* pa, int* pb){
    int temp;

    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```

```
a : 10, b : 20
a : 20, b : 10
```

실습예제 01

▶ 2개의 정수를 받아서 합과 곱을 동시에 연산하는 함수를 작성하고 메인 함수에서 호출하시오.

▷ 해당 함수의 프로토타입은 아래와 같다.

▷ `void get_sum_mul(int x, int y, int *p_sum, int *p_mul);`

Q & A
