[3주차 1강]
포인터(5)

학습 내용

- 9.5 포인터 인자와 주소 반환(2)

학습 목표

- 9.5 포인터 개념이 추가된 함수 호출 과정을 이해한다.

C 프로그래밍 및 실습

1. 포인터 개요
2. 포인터 선언과 사용
3. 배열과 포인터
4. 포인터 연산
5. **포인터 인자와 주소 반환**
6. 포인터 배열
7. 다중 포인터

5. 포인터 인자와 주소 반환(2)

주소 인자 추가 예시

- 함수의 실 인자가 포인터 변수인 경우

```
void main()
{  int a = 5;
   int *pa = &a;

   change(pa);

   printf("a=%d\n", a);
}
```

```
void change(int *p)
{
   *p = 10;
}
```

✓ 메모리 그림을 그리면서 수행과정을 따져보자

5. 포인터 인자와 주소 반환(2)

포인터 인자 활용 예제 : 두 변수의 값을 교환하는 함수

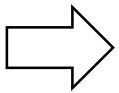
- 아래 swap 함수를 이용하면 main의 값이 바뀌는가? NO!!

```
void swap(int x, int y){
    int tmp = x;
    x = y;
    y = tmp;
    printf("%d %d\n", x, y);
}

void main(){
    int x = 10, y = 10;

    swap(x, y);

    printf("%d %d\n", x, y);
}
```

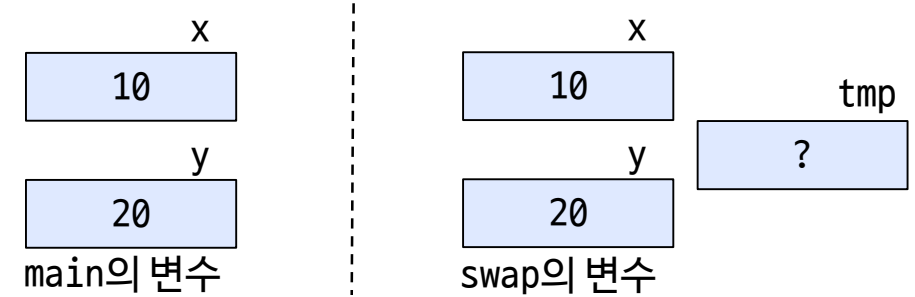


실행 결과

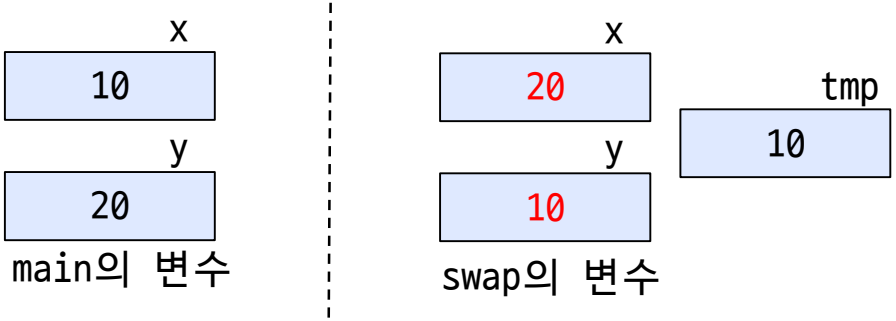
```
20 10 (swap)
10 20 (main)
```

5. 포인터 인자와 주소 반환(2)

왜 안 바뀌는 지 메모리 그림을 그려 확인해보자.



swap 함수 시작 시 메모리 그림



swap 함수 종료 직전 메모리 그림

5. 포인터 인자와 주소 반환(2)

swap함수에서 포인터를 인자로 사용하면?

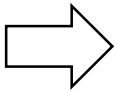
- ✓ 함수의 인자는 정수 포인터
- ✓ 호출 시 주소 전달

```
void swap(int *px, int *py){
    int tmp = *px;
    *px = *py;
    *py = tmp;
    printf("%d %d\n", *px, *py);
}

void main(){
    int x = 10, y = 20;

    swap(&x, &y);

    printf("%d %d\n", x, y);
}
```

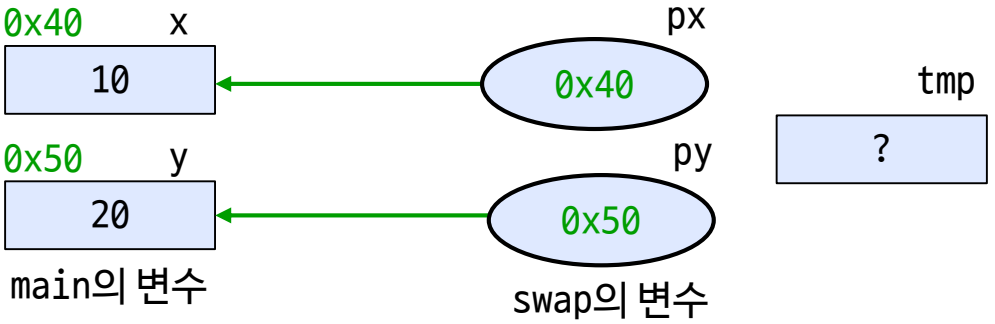


실행 결과

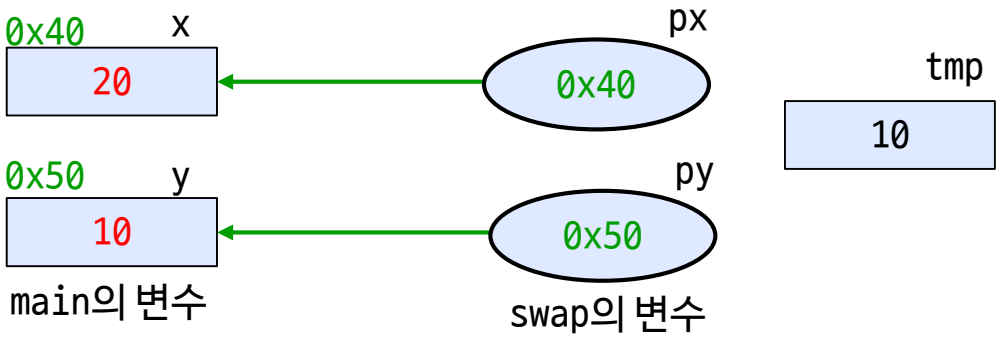
```
20 10 (swap)
20 10 (main)
```

5. 포인터 인자와 주소 반환(2)

main 함수의 값이 왜 바뀌는 지 메모리 그림을 그려 확인해보자.



swap 함수 시작 시 메모리 그림



swap 함수 종료 직전 메모리 그림

5. 포인터 인자와 주소 반환(2)

배열 인자 다시 보기

- 아래 코드에서 init 함수에서 배열 값을 변경하면, main의 배열 값도 변경된다. (함수 단원에서 학습했음)
- 왜 그럴까? 배열의 시작 주소가 인자로 전달되기 때문

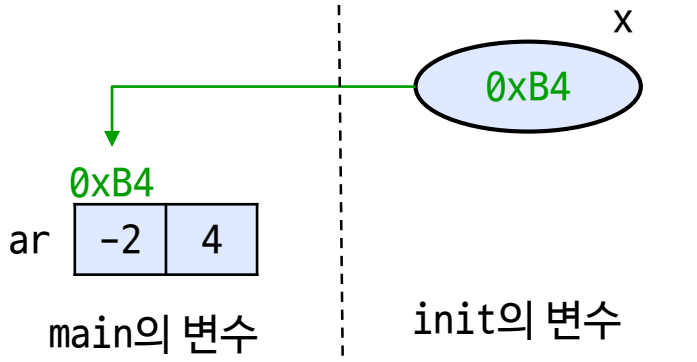
```
void init(int x[]){
    x[0] = x[1] = 0;
}
void main(){
    int ar[2]={-2,4};

    init(ar);

    printf("%d %d",ar[0],ar[1]);
}
```

실행 결과

```
0 0
```



init 함수 시작 시 메모리 그림

5. 포인터 인자와 주소 반환(2)

배열 인자 다시 보기

- init 함수의 형식 인자 int x[]은 **포인터 (배열 아님에 주의)**

```
void init(int x[]){
    ...
}
```

=

```
void init(int *x){
    ...
}
```

- 위 두 함수는 문법적으로 완전히 동일
- 배열과 포인터 중 의미를 두고자 하는 형태로 사용
  - ✓ 보통 함수 본체에서 배열 형태로 사용하는 경우, 배열 형태의 인자 사용(왼쪽 형태)

5. 포인터 인자와 주소 반환(2)

scanf와 printf의 인자

- scanf( )에서 변수 앞에 **&** (주소 연산자)를 붙이는 이유

```
int x;
scanf("%d", &x);
```

- 사용자로 부터 입력 받은 값을 변수 x에 저장해야 한다.
- scanf 함수에서 호출한 함수의 변수 x의 값을 바꾸려면, **주소**를 전달
- printf( )는?

```
int x = 0;
printf("%d", x);
```

✓ 출력 시는 x의 **값**을 넘겨주면 충분

5. 포인터 인자와 주소 반환(2)

scanf( )에서 항상 & (주소 연산자)를 붙여야 하는가?

- ✓ 주소 값이 인자로 전달되면 됨

```
int x[5], *p = &x[1];

scanf("%d", &x[0]);
scanf("%d", p);
scanf("%d", p+1);

printf("%d %d %d\n",x[0],x[1],x[2]);
printf("%p %p %p\n",&x[0],&x[1],&x[2]);
```

입력 예시

```
1 -4 9
```

출력 예시

```
1 -4 9
00B6FB60 00B6FB64 00B6FB68
```

5. 포인터 인자와 주소 반환(2)

주소를 반환하는 함수

- 주소도 값이므로 함수의 반환값으로 사용될 수 있다.
- 변수와 마찬가지로, 함수 이름 앞에 * **(참조연산자)** 을 붙여 주소를 반환함을 표시

```
void main(){
    int ar[5]={2,1,3,0,4};
    int *p1;

    p1 = next_addr(&ar[1]);

    printf("%d",*p1);
}
```

```
int *next_addr(int *p)
{
    return p+1;
}
```

실행 결과

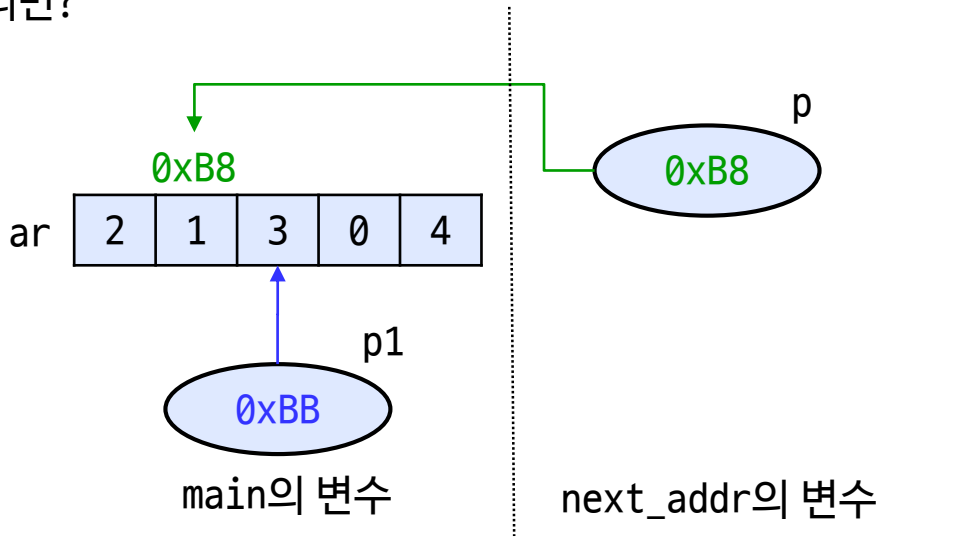
```
3
```

5. 포인터 인자와 주소 반환(2)

주소를 반환하는 함수

- 앞의 코드에 대해 메모리 그림을 그리면?

```
int *next_addr(int *p)
{
    return p+1;
}
void main(){
    int ar[5]={2,1,3,0,4};
    int *p1;
    p1 = next_addr(&ar[1]);
    printf("%d",*p1);
}
```



next_addr 함수 종료 직전 메모리 그림

※ 실습하기

[예제 9.8] 두 정수 변수의 **주소**를 인자로 받아, 값이 작은 변수의 **주소**를 반환하는 함수를 작성해보자.

- ✓ 변수에 저장된 값은 다르다고 가정
- ✓ main 함수는 아래의 코드 사용

```
void main(){
    int ar[5]={2,1,3,0,4};
    int *p1;

    p1 = smaller(&ar[1], &ar[3]);

    printf("%d",*p1);
}
```

```
?   smaller (   ?   )
{
          ?
}
```

실행 결과

```
0
```

학습 정리

- 주소를 함수에 전달하면 호출된 함수에서 호출한 함수의 지역 변수에 접근할 수 있어 **값을 변경할 수 있음**
- 배열을 인자로 사용하는 경우에도 배열의 주소가 전달되기 때문에, 호출된 함수에서 호출한 함수의 배열 값을 **변경할 수 있음**
- 주소도 함수의 **반환 값**으로 사용될 수 있음