[2주차 1강]
포인터(3)

학습 내용

- 9.3 배열과 포인터

학습 목표

- 9.3 배열과 포인터와의 관계를 이해한다.

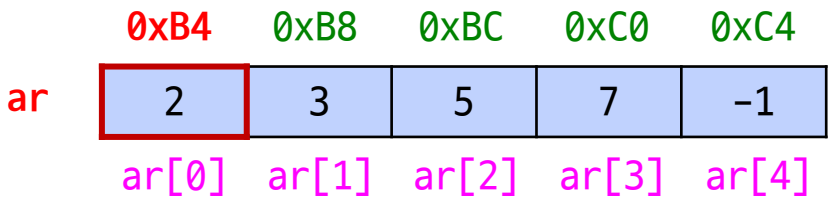
C 프로그래밍 및 실습

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

3. 배열과 포인터

배열 이름의 비밀

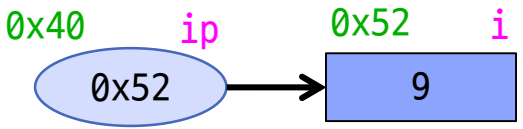
- 배열 이름은 배열의 0번 원소의 시작 주소를 의미 (특별하다)
  - ✓ ar : &ar[0]
  - ✓ ar의 자료형 : int *
  - ✓ 비교) &ar는 **전체 배열**의 시작 주소 (값은 같지만 다른 자료형)

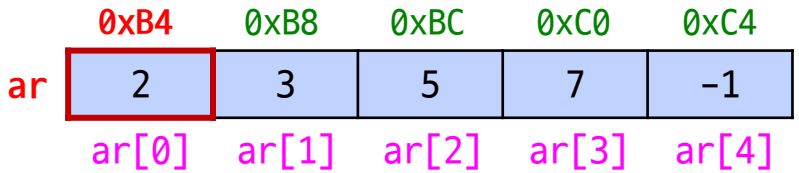


배열 이름의 비밀 (예: 대입문 오른쪽)

- 일반 변수와 배열 비교

일반 변수	배열
`int i=9, *ip = &i;` `i` : 변수 i에 저장된 값 `&i` : 변수 i의 주소	`int ar[5]={2, 3, 5, 7, -1};` `ar[2]` : 원소 ar[2]에 저장된 값 `&ar[2]` : 원소 ar[2]의 주소
`ip` : 변수 ip에 저장된 값 `&ip` : 변수 ip의 주소	`ar` : 0번 원소의 **주소** `&ar` : (전체) 배열의 주소





※ 배열 원소는 일반 변수와 동일하게 취급됨

주소를 이용한 배열 참조

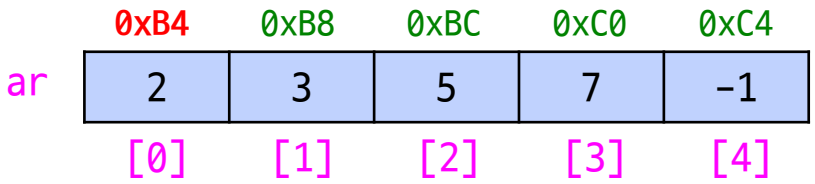
- 배열 이름은 주소를 의미하므로, **참조 연산자**와 함께 사용 가능
  - ✓ ar : 0번 원소의 주소
  - ✓ *ar : 0번 원소의 주소에 저장된 값, 즉, 0번 원소의 값을 의미

```
int ar[5]={2, 3, 5, 7, -1};

printf("%p %d %d\n", ar, ar[0], *ar);
```

실행 결과

```
001E40B4 2 2
```



배열 주소에 대한 증감 연산

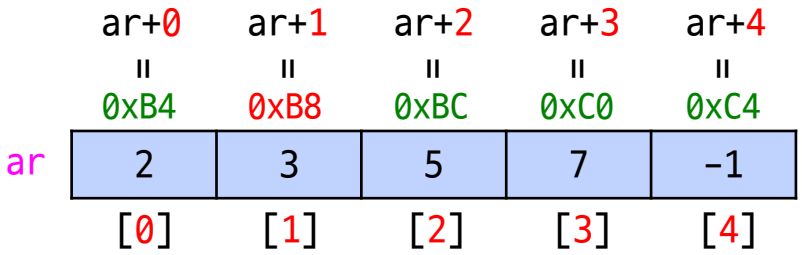
- **배열 원소 하나의 크기 만큼** 증가 or 감소 (int 배열의 경우: 4)
- ar+**i** : 배열 ar의 **i** 번째 원소의 주소
- *(ar+**i**) : 배열 ar의 **i** 번째 원소의 값, 즉, ar[i]

```
int ar[5]={2, 3, 5, 7, -1};
printf("%p %d %d\n", ar+1, ar[1], *(ar+1));
```

실행 결과

```
001E40B8 3 3
```

ar+1의 값은
001E40**B5**가 아님!!



※ 실습하기

[예제 9.4]
char형 배열과 double형 배열을 선언하고, 다음을 출력하라.

```
char car[5]={'H','e','l','l','o'};
double dar[5]={1.1, 2.2, 3.3, 4.4, 5.5};
```

- ✓ car, car[0], *car
- ✓ car+1, car[1], *(car+1)
- ✓ car+2, car[2], *(car+2)

- ✓ dar, dar[0], *dar
- ✓ dar+1, dar[1], *(dar+1)
- ✓ dar+2, dar[2], *(dar+2)

주소의 변화량을
주의해서 살펴보자

배열을 포인터 변수에 연결하여 사용하기

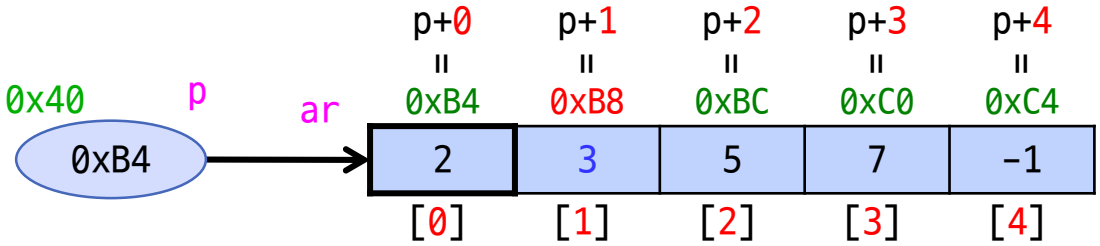
- 배열 이름은 주소를 의미하므로, 포인터 변수에 대입 가능
- 포인터 변수에 대한 증감 연산
  - ✓ 포인터 변수가 나타내는 자료형의 크기 단위로 증가 or 감소

```
int ar[5]={2, 3, 5, 7, -1};
int *p = ar;      ⇨ 포인터 변수에 ar 대입

printf("%p %d\n", p, *p));          ⇨ 0번 원소
printf("%p %d\n", p+1, *(p+1)); ⇨ 1번 원소
```

실행 결과

```
001E40B4 2
001E40B8 3
```



배열을 포인터 변수에 연결하여 사용하기

- 포인터 변수도 배열의 첨자 형태로 값을 참조 할 수 있다.

```
int ar[5]={2, 3, 5, 7, -1};
int *p = ar;

printf("%p %d %d\n", p, p[0], *p);          ⇨ 0번 원소
printf("%p %d %d\n", p+1, p[1], *(p+1));    ⇨ 1번 원소
```

실행 결과

```
001E40B4 2 2
001E40B8 3 3
```

※ 실습하기

[예제 9.5]
다음과 같이 포인터 변수를 선언하고 값을 출력하라.

```
char car[5]={'H','e','l','l','o'}, *cp=car;
double dar[5]={1.1, 2.2, 3.3, 4.4, 5.5}, *dp=dar;
```

- ✓ cp, cp[0], *cp
- ✓ cp+1, cp[1], *(cp+1)
- ✓ cp+2, cp[2], *(cp+2)

- ✓ dp, dp[0], *dp
- ✓ dp+1, dp[1], *(dp+1)
- ✓ dp+2, dp[2], *(dp+2)

주소의 변화량을
주의해서 살펴보자

3. 배열과 포인터

배열과 포인터의 관계 정리

- 배열과 포인터는 동일한 형태로 사용 가능 (둘 다 주소이니까)

```
int ar[5], *p = ar;
```

✓ 주소

ar+0 = p+0 = 0xB4 | ar+1 = p+1 = 0xB8 | ar+2 = p+2 = 0xBC | ar+3 = p+3 = 0xC0 | ar+4 = p+4 = 0xC4

0x40 p → 0xB4 ar

2	3	5	7	-1

✓ 정수

ar[0] = p[0] = *(ar+0) = *(p+0)

ar[1] = p[1] = *(ar+1) = *(p+1)

ar[2] = p[2] = *(ar+2) = *(p+2)

ar[3] = p[3] = *(ar+3) = *(p+3)

ar[4] = p[4] = *(ar+4) = *(p+4)

ar[4] = p[4] : 배열형태 참조

*(ar+4) = *(p+4) : 포인터형태 참조

배열과 포인터 정리 (복잡해 보이지만 다음 두 가지만 기억하자)

```
int ar[5], *p = ar;
```

- 주소에 1을 더하면, 원소의 크기만큼 주소가 증가한다.
  - ✓ ar + 3 , p + 3 : ar과 p 모두 주소
- 원소(변수) 값은 다음 두 가지 형태로 참조할 수 있다.
  - ✓ ar[3] 과 p[3] : 배열의 첨자 연산자 [ ] 사용
  - ✓ *(ar+3) 과 *(p+3) : 포인터의 참조 연산자 * 사용

배열 이름이든 포인터 변수이든 주소를 의미하고,
따라서 **참조 방식도 동일**하다.

배열과 포인터 주의사항 1

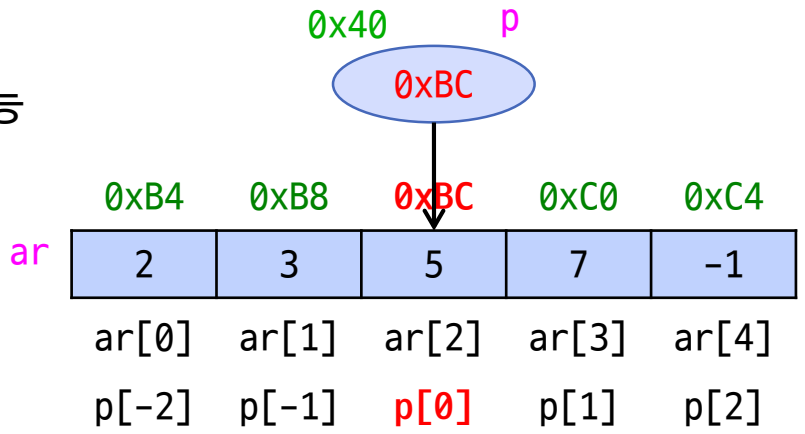
- 포인터를 배열의 중간 원소에 연결시키는 것도 가능

```
int ar[5]={2, 3, 5, 7, -1};
int *p = &ar[2];     // 2번 원소에 연결

printf("%p %d\n", ar, ar[0]);
printf("%p %d\n", p, p[0]);
```

실행 결과

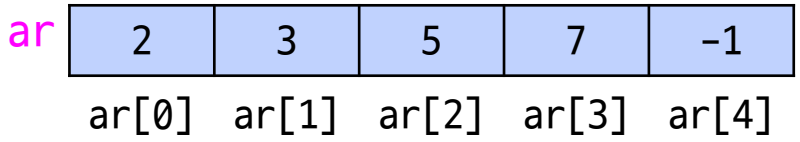
```
001E40B4 2
001E40BC 5
```



- 포인터는 단지 자신이 가리키는 주소를 기준으로 배열처럼 쓰는 것일 뿐

배열과 포인터 주의사항 2

- 포인터의 참조 연산자 사용시 괄호에 유의
  - ✓ *(ar+2) ➔ ar[2] ➔ 5
  - ✓ *ar + 2 ➔ *(ar) + 2 ➔ ar[0] +2 ➔ 4 (연산자 우선순위 때문)



배열과 포인터 주의사항 3

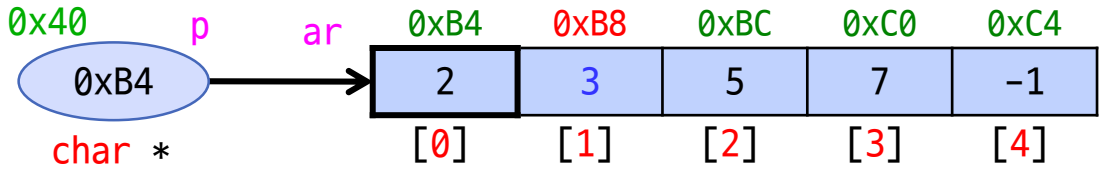
- 포인터 변수의 증감량은 가리키는 배열의 원소 크기가 아니라, **포인터 자신의 자료형에 의해** 결정
  - ✓ 예) char * 형 포인터에 int 배열을 연결하면

```
int ar[5]={2, 3, 5, 7, -1}, i;
char *p = (char *) ar;

for( i=0; i < 5 ; ++i )
  printf("%p, %d\n", p+i, *(p+i));
```

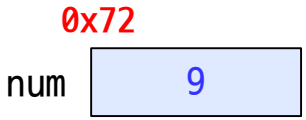
1씩 증가

```
001E40B4, 2
001E40B5, 0
001E40B6, 0
001E40B7, 0
001E40B8, 3
```

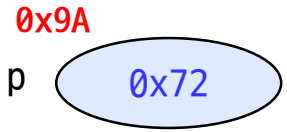


배열 이름과 포인터 변수의 차이점

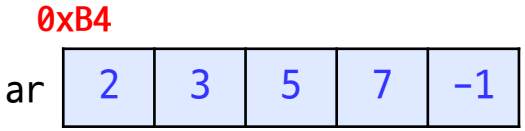
- int num;
  - ✓ 변수 num에 저장된 값(정수)은 변경 가능
  - ✓ 변수 num에 할당된 주소는 변경 불가



- int *p;
  - ✓ 변수 p에 저장된 값(주소)은 변경 가능
  - ✓ 변수 p에 할당된 주소는 변경 불가



- int ar[5];
  - ✓ 배열 ar에 저장된 값은 변경 가능
  - ✓ 배열 ar에 할당된 주소는 변경 불가



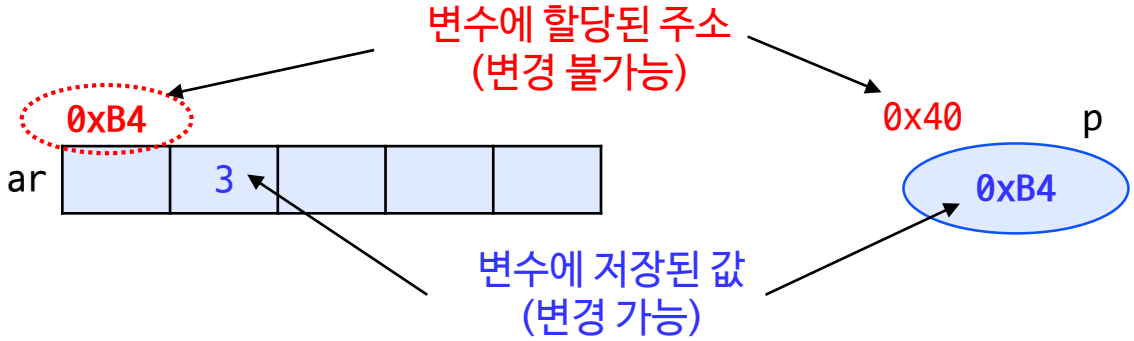
    - ✓ 배열 이름은 **포인터 상수**로 변경 하지 못한다.
    - ✓ 대입문의 왼쪽에서 사용될 때 (l-value) 차이 발생

배열 이름과 포인터 변수의 차이점

```
int num, *p, ar[5];

p = &num;  // 가능
++p;               // 가능
&p = ar;   // 불가능 (컴파일 에러)

ar = &num;       // 불가능 (컴파일 에러)
++ar;            // 불가능
&ar = &num;      // 불가능 (컴파일 에러)
```



학습 정리

- **배열이름**은 0번 원소의 주소를 의미하고, 배열 이름의 값은 변경 불가능함
- **배열**을 **포인터** 형태로 사용할 수 있고, 반대로 포인터를 배열 형태로 사용할 수 있음
- 포인터 변수의 **증감량**은 가리키는 배열의 원소 크기가 아니라,
  포인터 자신의 자료형에 의해 결정됨
- 변수에 할당된 주소는 변경이 불가능하나, 변수에 저장된 주소 값은 변경 가능함