C 프로그래밍 및 실습

[15주차 1강]
포인터(3)

학습 내용

9.2 포인터 선언과 사용 (2부)

9.3 배열과 포인터 (1부)

학습 목표

9.2 포인터의 사용시 주의사항을 이해한다.

9.3 배열과 포인터와의 관계를 이해한다.

C 프로그래밍 및 실습

9.2 포인터 선언과 사용 (2부)

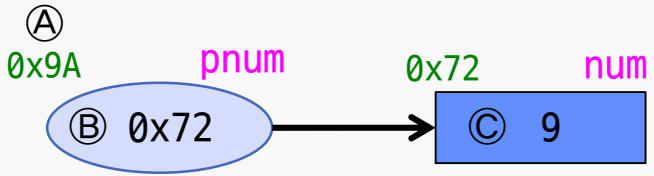
9.3 배열과 포인터 (1부)

포인터와 관련한 두 연산자 정리 (pnum 기준)

- **주소연산자(&):** 해당 변수의 주소 값 (그림의 Ⓐ)
- **변수 이름:** 변수 영역 또는 변수에 저장된 값 (그림의 Ⓑ)
- **참조연산자(*):** 포인터가 가리키는 변수(그림의 Ⓒ)

```
int num = 9, *pnum = &num;

printf("%p\n", &pnum);        ⇨ Ⓐ 0x9A
printf("%p\n", pnum);         ⇨ Ⓑ 0x72
printf("%d\n", *pnum);        ⇨ Ⓒ 9
```

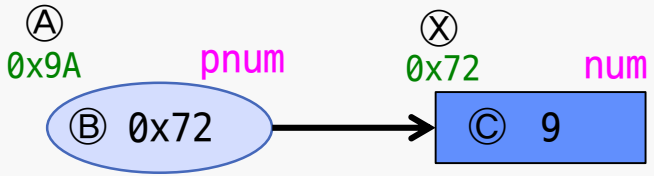


포인터와 관련한 두 연산자 정리

- (주의) pnum과 &num의 값은 동일하지만, 지칭하는 부분은 전혀 다름
- (질문) &*pnum 과 *&num이 각각 의미하는 부분은?

```
int num = 9, *pnum = &num;

printf("%p\n", &pnum);        ⇨ Ⓐ 0x9A
printf("%p\n", pnum);         ⇨ Ⓑ 0x72
printf("%d\n", *pnum);        ⇨ Ⓒ 9
printf("%p\n", &num);         ⇨ Ⓧ 0x72
```

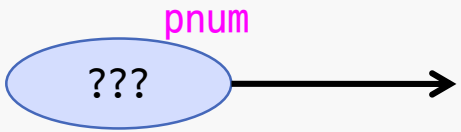


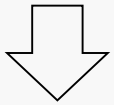
포인터를 이해하고 학습하기 위한 가장 좋은 방법은
메모리 그림을 그리는 것이다!!

포인터 주의사항 1 (초기화)

- 선언 후 연결 없이 바로 사용하면?

```
int *pnum ;    // pnum에는 쓰레기 값
*pnum = 9 ;    // 런타임(실행) 오류 발생
```



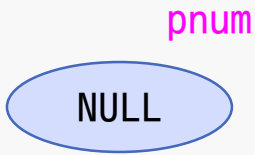


```
int *pnum, num;
pnum = &num; // 반드시 어떤 변수에 연결 후 사용
*pnum = 9;
```

널(NULL) 포인터

- **주소 값 0**을 나타내는 특별한 기호
- 아무것도 가리키지 않음을 의미
- NULL의 값은 0이므로, 조건문에서 사용하면 거짓에 해당
- 예기치 못한 오류 방지를 위해 포인터 변수를 NULL로 초기화

```
int *pnum = NULL;
```



포인터 주의사항 2

- **& (주소연산자)**는 포인터를 포함한 모든 변수에 사용가능

- *** (참조연산자)**는 포인터 변수에서만 가능
  - ✓ *num (num이 가리키는 변수)은 정의 되지 않음

```
int num=9, *pnum = &num;

printf("%p %p %d\n", &pnum, pnum, *pnum);

printf("%p %d %d\n", &num, num, *num); // 컴파일 오류
```

포인터 주의사항 3 (대입)

- **포인터의 자료형**과 **연결된 변수의 자료형**은 **일치**해야 한다.
- 서로 다른 자료형의 포인터 간 대입
  - ✓ 문법적으로는 허용이 되기도 하지만 (컴파일 경고만 발생)
  - ✓ **프로그램 오류의 원인**이 됨

```
int num;
char *pch = &num;          // 자료형 불일치

*pch = 4;

printf("%d %d\n", num, *pch);
```

실행 예시
(결과는 다를 수 있음)

```
-858993660 4
```

포인터의 크기

- **포인터의 종류(자료형)에 관계 없이** 주소를 저장하기 위해 필요한 공간은 동일
  - ✓단, 포인터의 크기는 시스템에 따라 다를 수는 있음
- sizeof 연산자를 이용하여 확인해보자.

```
char *pch;
int *pnum;
double *pdnum;

printf("%d\n", sizeof(pch));
printf("%d\n", sizeof(pnum));
printf("%d\n", sizeof(pdnum));
```

실행 결과

```
4
4
4
```

C 프로그래밍 및 실습

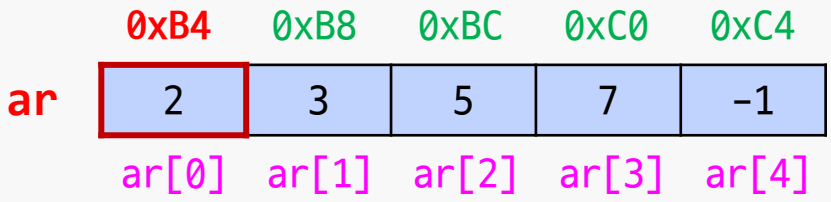
9.2 포인터 선언과 사용 (2부)

9.3 배열과 포인터 (1부)

배열 이름의 비밀 (예: 대입문 오른쪽)

- 배열 이름은 배열의 0번 원소의 시작 주소를 의미한다. (특별하다)
  - ✓비교) &ar는 전체 배열의 시작 주소 (값은 같지만 다른 자료형)

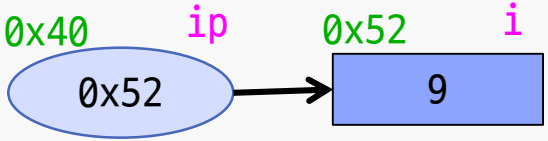
```
ar : 0번 원소의 주소
&ar : (전체)배열의 주소
```

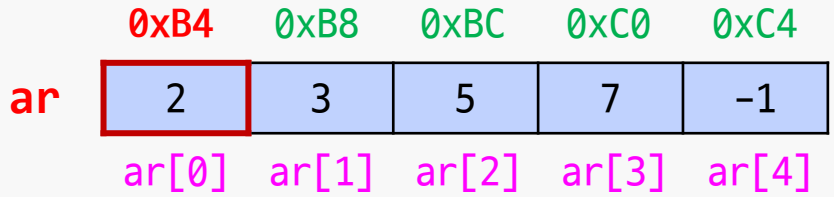


배열 이름의 비밀 (예: 대입문 오른쪽)

- 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

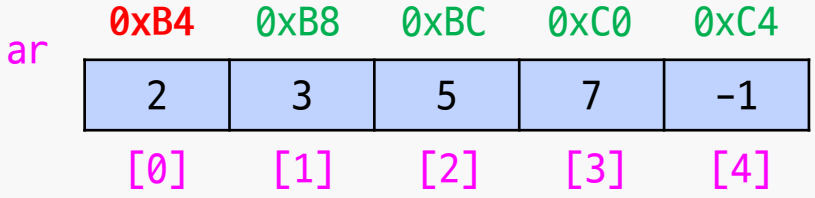
- 배열 이름은 주소를 의미하므로, **참조 연산자**와 함께 사용 가능
  - ✓ ar : 0번 원소의 주소
  - ✓ *ar : 0번 원소의 주소에 저장된 값, 즉, 0번 원소의 값을 의미

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

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

실행 결과

```
001E40B4 2 2
```



세종대학교

배열 주소에 대한 증감 연산

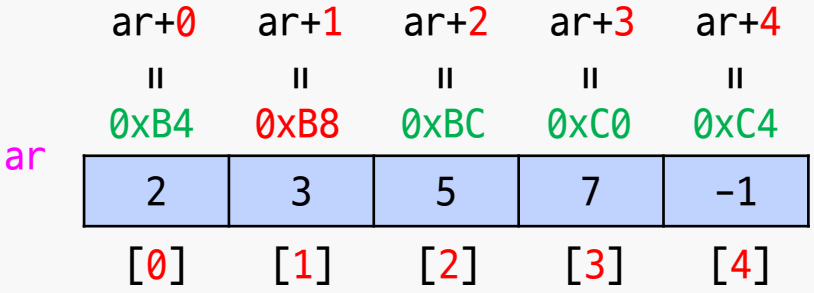
- 배열 원소 하나의 크기 만큼 증가 or 감소 (int 배열의 경우: 4)
- ar+i : 배열 ar의 i 번째 원소의 주소
- *(ar+i) : 배열 ar의 i 번째 원소의 값, 즉, ar[i]

```
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
001E40B5가 아님!!

실습 하기

[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

학습 정리

- 포인터 변수는 다른 변수에 **연결한 후에 사용**해야 함
- **NULL**은 주소 값 0을 나타내는 특별 기호로,
  아무것도 가리키지 않는다는 것을 의미함
- 포인터의 자료형과 연결된 변수의 자료형은 일치해야 함
- **배열 이름**은 0번 원소의 주소를 의미하고, 배열 이름의 값은 변경 불가능 함