포인터

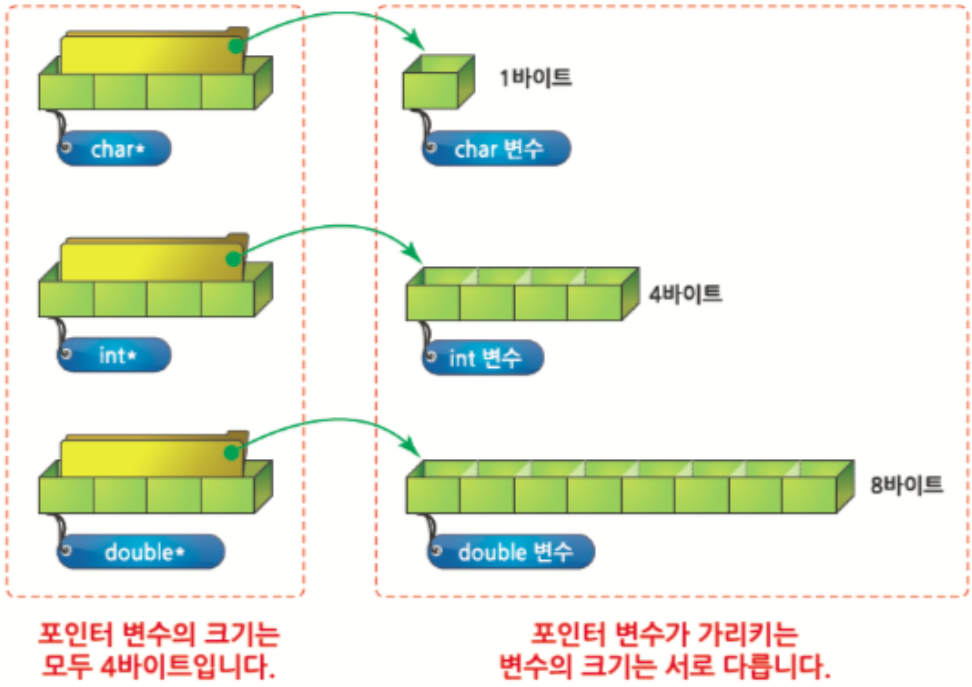
1. 강의

▼ 포인터

- 포인터는 주소를 저장하는 변수이다.
- 포인터 변수는 다른 변수를 가리키는 변수이다. → 주소를 이용해 변수에 접근한다.
- 포인터 선언 형식

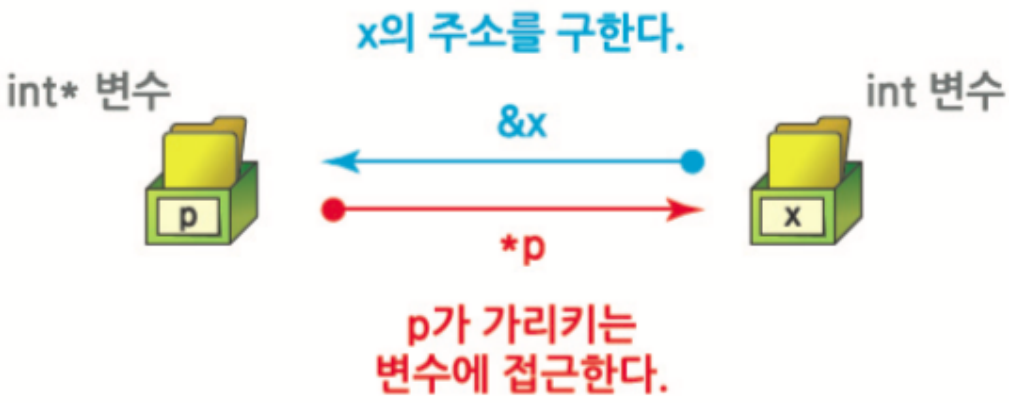
```
데이터형 * 변수명;
char   * pc ;
int    * pi ;
double * pd ;
```

- char*는 char형 변수의 주소, int*는 int형 변수의 주소, double*는 double형 변수
  의 주소를 저장한다.
- char*형 변수는 char형 변수를 가리키고, int*형 변수는 int형 변수를 가리키고,
  double*형 변수는 double형 변수를 가리킨다.
- 포인터의 변수의 크기는 주소를 저장 하므로, char, int, double 형이든 상관없이 모두 동일하다. (32비트 운영체제에선 4 byte, 64비트 운영체제에선 8 byte)



- ex81.c 예제
- int* p; == int *p; == int * p;
- int *p1, p2; (p2는 포인트 변수가 아니다.)
- int *p1, *p2;

▼ 포인터 사용



```
int x;
int *p;
p = &x;  ──────────────  p에 x의 주소를 보관한다.
*p = 10; ──────────────  p가 가리키는 변수, 즉 x에 10을 대입한다.
                         p가 int*형이므로 *p는 int형이 된다.
printf("%d", *p); ─────  p가 가리키는 변수의 값을 출력한다.
```

- ex82.c 예제
- * 의 용도
  - 곱하기 : 곱셈 연산 수행
  - 포인터 수식어 : 포인터를 만들어 준다.
  - 간접 참조 연산자 : 포인터 변수가 가리키는 변수의 값

▼ 이중 포인터

- double_pointer.c 참조
- p에는 x의 주소가 대입되어 있고(가리키고), *P 는 x의 주소에 있는 값을 읽어 온다.
- a에는 p의 주소가 대입되어 있고(가리키고), *a 는 x의 주소를 읽어 온다.
- *(x의 주소) = x 값이 된다.

▼ 포인터 변수의 데이터 형 == 포인터 변수가 가리키는 변수의 데이터 형

- char 변수의 주소는 char* 형 변수에 저장
- int 변수의 주소는 int* 형 변수에 저장
- double 변수의 주소는 doub* 형 변수에 저장
- 그렇지 않을 경우, 컴파일 경고나, 실행 에러가 발생 할 수 있다.

▼ 포인터 변수 초기화

- 포인터 변수를 선언만 해놓고 초기화 하지 않으면, 포인터 변수가 메모리의 아무곳이나 가리킨다.
- 이 상황에서 값을 저장하면, 메모리 어딘가에 함부로 값을 저장하게 되므로, 포인터 변수는 선언하며 초기화를 하는 것이 안전하다.

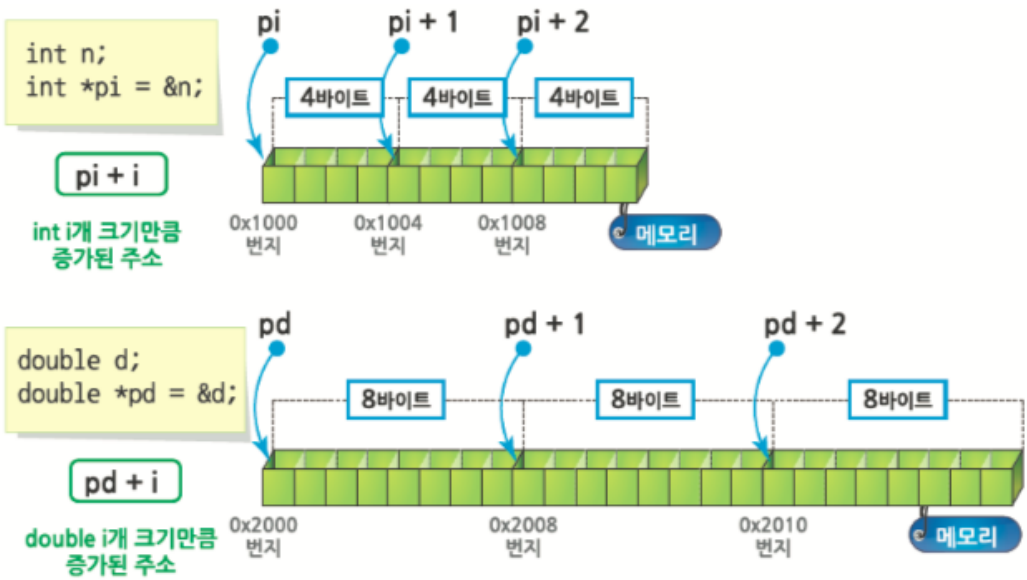
- <stdio.h> 에서 제공하는 null 을 사용해 초기화 한다. → 이것을 null 포인터라 한다.
- 프로그램 안에서 포인터를 안전하게 사용하려면 null 포인터인지 검사하고 값을 대입시킨다.

```
int *p = NULL;

~ ~ ~

if(p !=NULL) {

  *p = 10;
}
```

- 포인터 변수에 주소 연산자 & 를 사용하지 않고 직접 주소를 넣어주는 경우, (char*) 형으로 형변환을 한 후에 넣어 주어야한다.
- 단지 주소를 0×123456 이라고 타이핑해 넣는 것 만으로 주소가 되지 않는다.

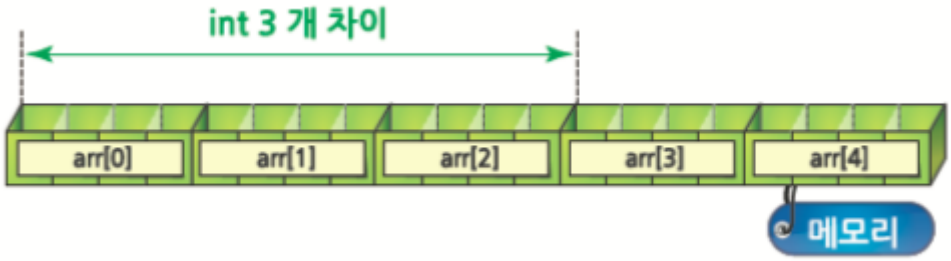
```
p = (char*) 0x123456;
```

▼ 포인터 연산 : +, -



- ex83.c 예제 참조
- 포인터 + 1을 하면, 포인터의 각 자료형(char, int, double) 만큼 크기를 증가시켰을 때의 주소가 연산의 결과이다.

- 즉, 포인터 + 1 했을때, char 형은 1 바이트, int 형은 4 바이트, double 형은 8 바이트 증가 후의 주소가 연산 결과가 된다.
- 포인터 - 포인터 = 포인터가 가리키는 데이터형 몇 개 크기만큼 차이가 나는지를 알수 있음 (예제 ex83-1.c)
- 즉, arr[3]의 주소로부터 arr[o]의 주소가 몇개째(포인터가 가리키는 데이터형) 떨어져 있는지를 계산한다.

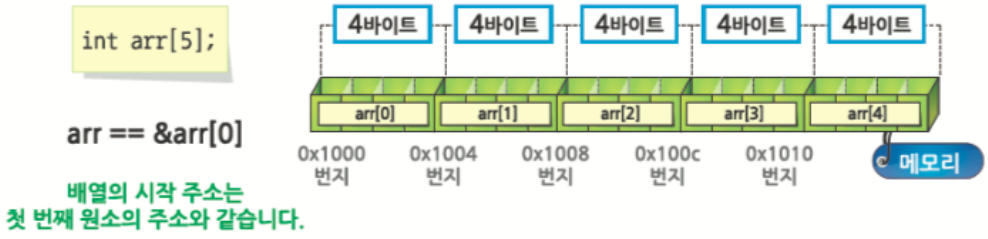


▼ 포인터 연산 : ++, - -

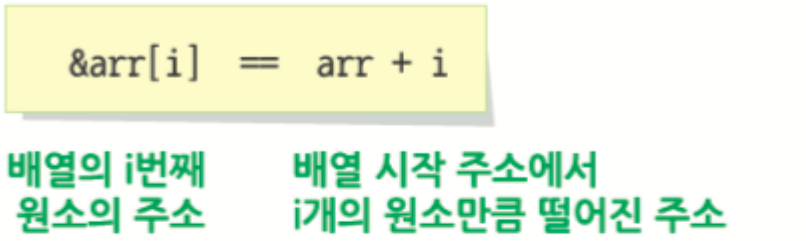
- 포인터 형에 따라 ++, - - 의 연산 결과도 달라 진다.
- 포인터 연산에는 이외에도 ==, ! = 등의 관계 연산도 수행 할 수 있다.

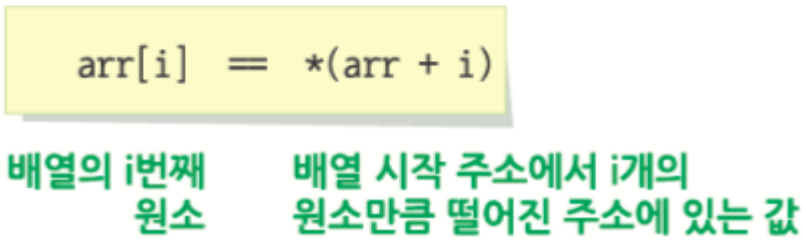
▼ 포인터와 배열

- 배열을 인덱스 없이 배열명만 사용하면 배열의 시작 주소를 의미한다.

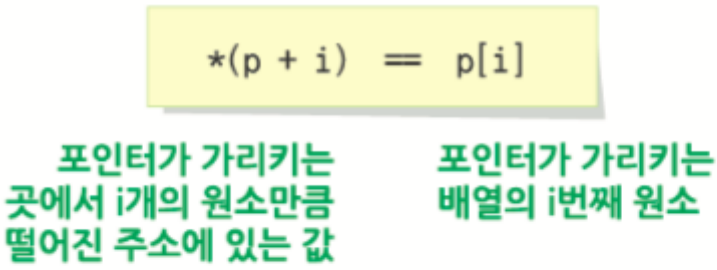


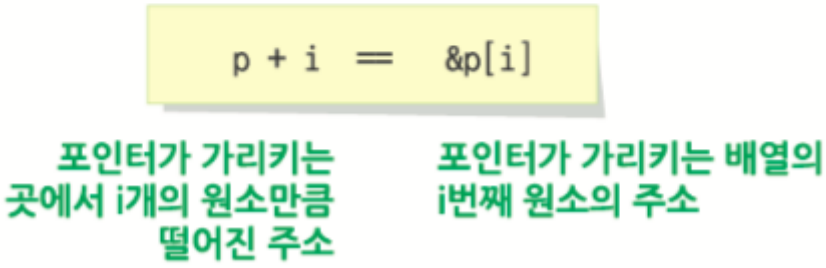
- ex85.c 참조
- 배열접근에 사용하는 [ ] 는 컴퓨터 내부적으로 포인터 연산을 수행해 구현한다.
- arr[i] = *(arr+i)
- &arr[i] = arr + i





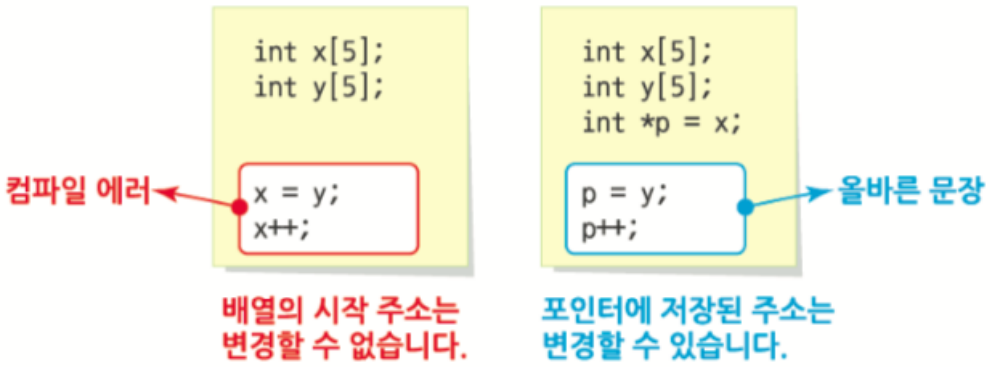
- 위와 같은 성질로 , 포인터 변수를 배열 이름인 것처럼 사용 할 수 있다.
- 배열의 시작 주소로 초기화된 포인터는
  - *(p+i) = p[i] ⇒ p로부터 i 만큼 떨어진 주소에 있는 값 = 포인터가 가리키는 배열의 i번째 원소
  - p + i = &p[i] ⇒ p로부터 i 만큼 떨어진 주소 = 포인터가 가리키는 배열 i번째 원소의 주소





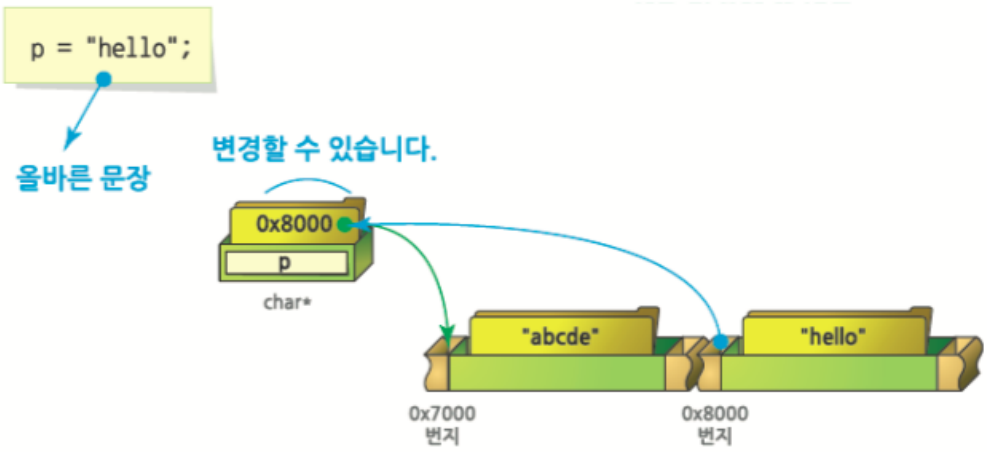
- ex86.c 예제 실행 : p가 처음에 arr[2]를 가리키는 경우에 대한 예제

- 배열과 포인터는 같은 방법으로 사용할 수 있지만, 배열은 주소를 변경 할 수 없는 포인터이다. (ex86-1.c)



▼ 포인터와 문자열

- 문자열을 포인터에 할당 하면, 문자열은 문자열 리터럴로 간주 된다.
- 문자열 리터럴은 변경 불가능 하며 포인터에는 문자열 리터럴의 주소값이 할당 된다. (ex87)

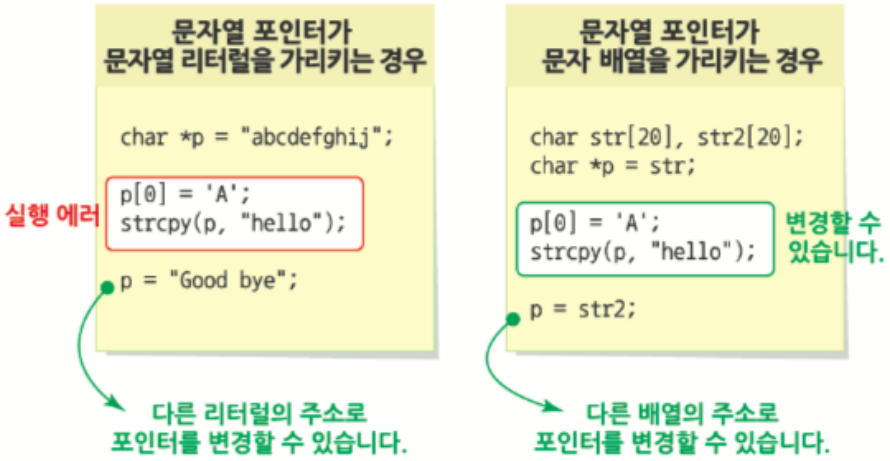


- 문자열 리터럴의 주소를 저장하는 포인터 변수는 문자열을 변경하는 strcpy, strcat 의 첫번째 인수로 사용 될 수 없다. (문자열 리터럴이 아닌 문자 배열의 경우에는 가능하다.)
- 포인터 변수에 할당된 문자열의 값을 변경하려면 에러가 발생하지만, 문자열을 가리키는 주소는 변경이 가능하므로, 포인터 변수에 새로운 문자열 리터럴의 주소값을 할당하는 것은 가능하다.

- 문자열을 비교할 때도 == 를 사용하지 않고, strcmp 를 사용해야 한다.

```
char str[20] = "Hello World";
if (str == "Hello World")  // 문자 배열과 문자열 상수의 주소를 비교
{
}
if (strcmp(str, "Hello World") ==0 )
{
}
```

- char* 형에는 문자열 리터럴의 주소나 문자열 배열의 주소를 저장할 수 있다.
- 포인터에 문자열 배열의 시작 주소냐, 문자열 리터럴의 주소냐에 따라 연산 과정에 에러가 생길수도 있거나 혹은 제대로 실행 될 수도 있다. (ex88.c)



-

▼ const 포인터 (ex800.c)

- const 가 데이터형 앞에 있을 경우 → 주소 변경 가능, 변수의 값은 변경 불가능
- const 가 포인터 변수명 앞에 있을 경우 → 주소 변경 불가능, 변수의 값은 변경 가능
- const 가 데이터형, 변수명 모두 앞에 있을 경우 → 주소 변경 불가능, 변수의 값 변경 불가능

2. 실습

▼ ex82_after.c

- 포인터 pc 와 정수형 변수 c 선언
- c =22 대입, pc = &c 대입
- c 의 주소와 값 출력
- pc의 주소와 값 출력
- c= 11 로 변경한후
- c 와 pc 의 주소, 값 출력
- *pc =2 로 변경 후
- c 와 pc 의 주소 및 값 출력

▼ ex85_after.c

1. 배열을 이용한 처리

- 배열 x[6] 을 선언
- for문을 사용하여, scanf로 &x[i] 값을 6개 받아 들여 배열을 완성한다.
- 완성된 배열의 각 원소를 sum += x[i] 를 수행하여 합을 구한다.

2. 포인터를 이용한 처리

- 배열 x[6] 선언
- for 문, scanf 사용 x + i 를 사용하여 자료 6개를 받아 들임
- 완성된 배열의 각 원소를 *(x+i)로 접근하여 sum을 구하시오.

▼ 10_3_pointer_exer_3.c (ex86-2.c 참조)