배열의 특징

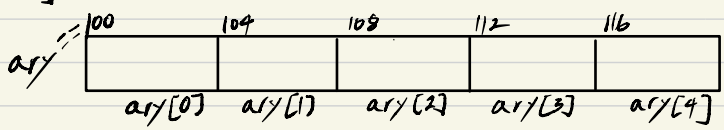
배열은 자료형이 같은 변수를 메모리에 **연속으로** 할당한다.

⇓

첫 번째 요소의 주소값만 알면 나머지의 주소도 알 수 있다.

ex) `int ary[5];`

컴파일



★배열명은 첫번째 배열 요소의 주소 값이다.★

∴ *ary는 첫번째 배열요소이다.

주소의 연산

주소 + 정수 ⟶ 주소 + (정수 * 주소를 구한 변수의 크기)

ex) 정수형 변수 a의 주소 : 100

&a == 100

&a + 1 == 100 + 1*4 == 104

배열과 포인터

간접참조 연산자 *

*주소값

↳ 해당 주소 값을 갖는 메모리에 접근

배열명 == 첫 배열 요소의 주소 값

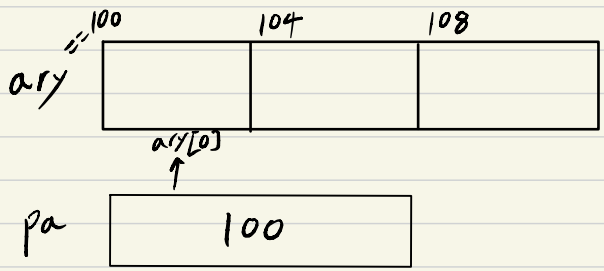
∴ *배열명 == 첫 배열 요소

포인터에 배열명 대입

배열명은 첫 배열 요소의 주소값이다. ⇒ 배열명은 상수이다.
포인터 변수는 주소 값을 저장하는 변수다.

∴ 포인터 변수에 배열명을 대입할 수 있다.

ex) `int ary[3];`
`int* pa = ary;` // pa는 ary[0]을 가리킨다.



`*(pa+1) = 20` // pa+1 == 100 + 1 * 4 == 104, 즉 *(pa+1) == ary[1]

※ 포인터 변수에 배열명 대입 시 다음과 같이 사용도 가능

```
pa[0] == *pa
printf("%d", pa[1]);
```

Difference

배열명과 포인터의 차이는?

1. sizeof 연산결과가 다르다.
   - sizeof (배열명) : 배열 전체 크기
   - sizeof (포인터변수) : 포인터 하나의 크기
2. 배열명은 상수고 포인터는 변수다.
   - ex) pa++ ⇒ 가능
   - ary++ ⇒ Error

포인터의 뺄셈

포인터 - 포인터 ⟶ 값의 차 / 자료형의 크기

↓ 자료형이 같아야 한다.

ex) `int ary[5] = {10, 20, 30, 40, 50};`
`int* pa = ary;` // ary를 100이라 가정하자
`int* pb = pa + 3;`

⇓

`*pa == 10, *pb == 40`
`pb - pa == (112 - 100) / 4 == 3`

↓

배열요소의 위치가 3개 떨어져 있다.

X. 관계연산 (>, <, >=, <=) 등도 가능하다

배열을 처리하는 함수

배열명을 인수로 전달 ⟶ 함수의 매개변수인 포인터에 대입 ⟶ 배열 전체에 접근 가능

```
ex) int main(void) {
        int ary[5] = {10,20,30,40,50};
        print_ary(ary)      //주소값을 인수로 전달
        :
    }

    void print_ary(int* pa) {      // pa = ary
        for(int i=0; i<5; i++) {
            printf("%5d", pa[i]);   //또는 *(pa+i)
        }
    }
```

포인터로 배열명을 받음으로써 두 함수간 메모리 공유 가능

매개변수에 배열을 선언하는 경우

↳ 이 경우, 배열명은 컴파일 과정에서 첫번째 배열요소를 가리키는 포인터로 나뀐다.

ex) `void func(int pa[5])` —compile→ `void func(int* pa)`

배열 요소의 개수는 아무 의미가 없다.

Also, 포인터이므로 sizeof 연산자의 결과 값이 포인터 하나의 크기로 나온다.