Ch 07 배열

목차

배열의 필요성

이름이 다른 변수를 여러 개 사용하는 경우

```
int main(void)
{
  int a, b, c, d, e;
  int sum = 0;

  scanf("%d", &a);
  sum += a;
  scanf("%d", &b);
  sum += b;
  scanf("%d", &c);
  sum += c;
  scanf("%d", &d);
  sum += d;
  scanf("%d", &e);
  sum += e;

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

이름이 다른 변수 5개를 선언한다.

변수의 이름이 다르기 때문에 반복문을 사용할 수 없다.

코드가 길고 복잡하다.

배열을 사용하는 경우

```
int main(void)
{
  int arr[5];
  int sum = 0;
  int i;

  for (i = 0; i < 5; i++)
  {
    scanf("%d", &arr[i]);
    sum += arr[i];
  }

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

크기가 5인 배열을 선언한다.

반복문을 사용할 수 있다.

배열의 원소에 대하여 같은 코드를 수행한다.

배열의 원소들은 같은 이름을 사용하며 인덱스로 구분해서 사용한다.

배열의 개념 (1)

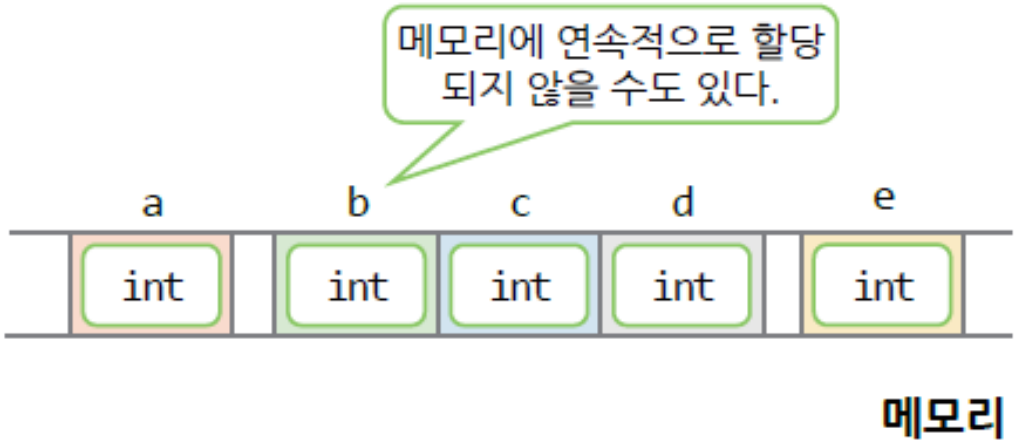
- 배열(array)은 같은 데이터형의 변수를 메모리에 연속적으로 할당하고 같은 이름으로 사용하는 기능이다.
- 배열의 원소(element)
  - 배열 안에 있는 변수 하나하나
- 배열의 인덱스(index) 또는 첨자
  - 배열의 원소를 구분하기 위한 번호
  - 배열의 인덱스는 0부터 시작한다.
  - arr[0], arr[1], ...과 같은 방식으로 배열의 원소를 구분한다.
- 배열의 각 원소도 변수이다.
  - 배열의 원소는 항상 연속된 메모리에 할당된다.

배열의 개념 (2)

- 배열의 이름은 배열 전체에 대한 이름이 된다.

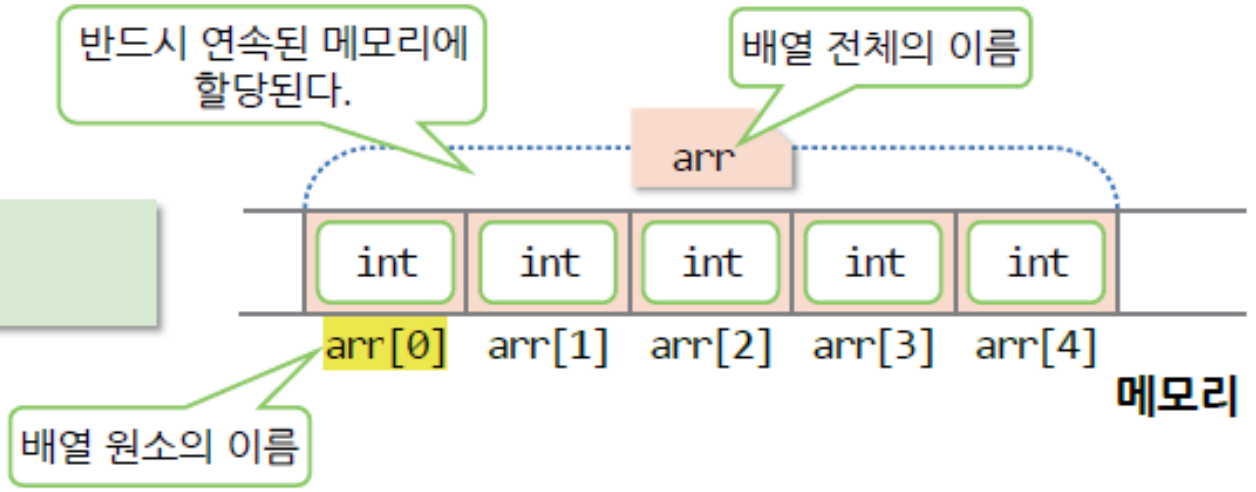
서로 다른 변수로 선언하는 경우

```
int a, b, c, d, e;
```



배열로 선언하는 경우

```
int arr[5];
```



배열의 선언 (1)

- 배열 원소의 데이터형과 배열의 이름을 쓰고 [ ] 안에 배열의 크기를 지정한다.

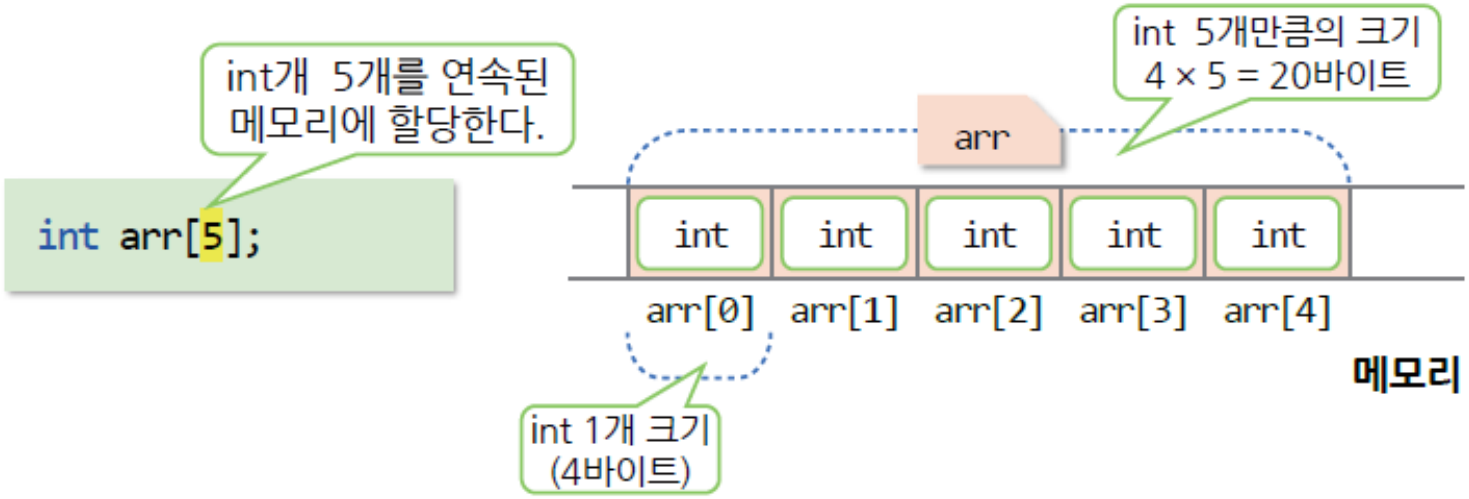
형식 데이터형 배열명[크기];

사용예

```
int arr[5];
double data[100];
char name[32];
```

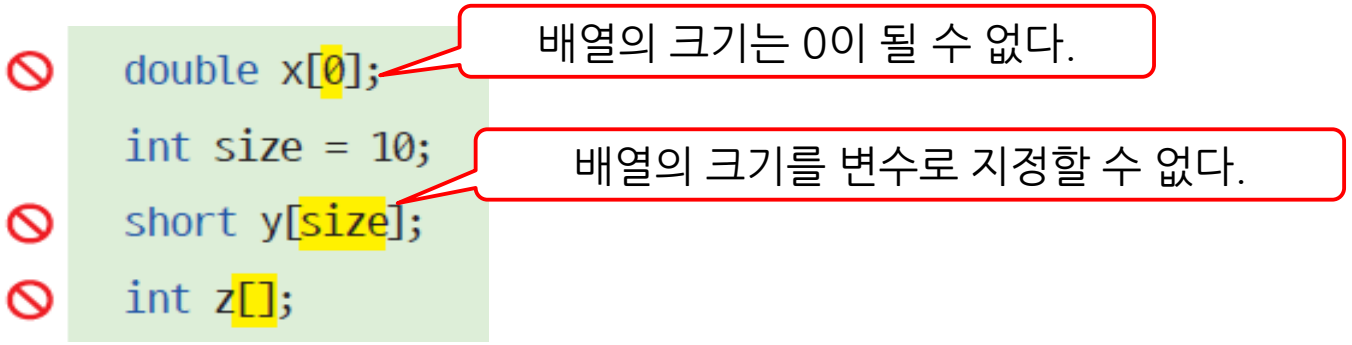
배열에 들어있는 원소의 개수

- 원소가 N개인 *type*형 배열을 선언하면 *type*형의 변수 N개를 메모리에 연속적으로 할당한다.

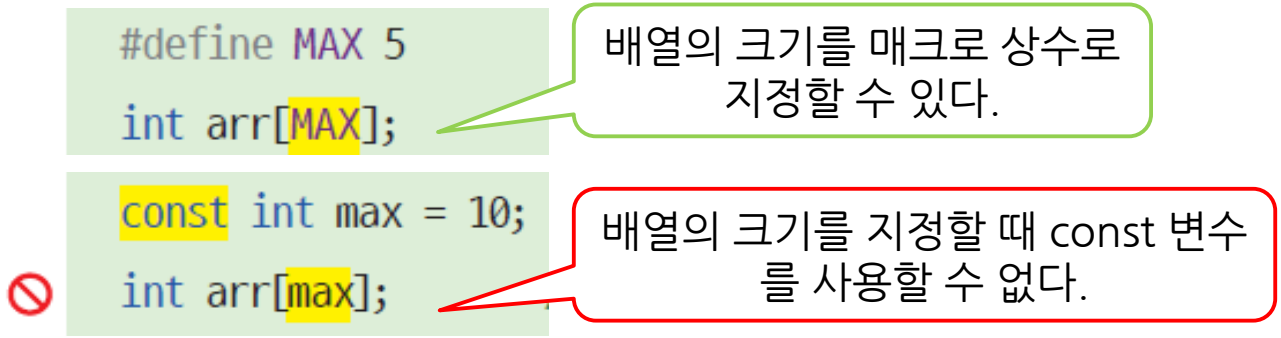


배열의 선언 (2)

- 배열의 크기는 반드시 0보다 큰 정수형 상수로 지정해야 한다.



- 매크로 상수는 배열의 크기를 지정하는 데 사용할 수 있다. const 변수는 변수이므로 배열의 크기를 지정하는 데 사용할 수 없다.



배열의 선언 (3)

- sizeof(*arr*)
  - arr 배열 전체의 바이트 크기
- sizeof(*arr[i]*)
  - arr 배열 원소의 바이트 크기

```
int arr[5];
printf("%d\n", sizeof(arr));
printf("%d\n", sizeof(arr[0]));
```

20바이트

4바이트

- sizeof(*arr*)/sizeof(*arr[0]*)
  - arr 배열의 크기, 즉 원소의 개수

```
int arr[5];
int size = sizeof(arr) / sizeof(arr[0]);
```

예제 7-1: 배열의 크기 구하기 (1)

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      int arr[5];
06      int size = 0;
07      int i;
08
09      printf("배열의 바이트 크기: %d\n", sizeof(arr));
10
11      size = sizeof(arr) / sizeof(arr[0]);
12      printf("배열의 크기: %d\n", size);
13
```

크기가 5인 int 배열을 선언한다.

배열 전체의 바이트 크기

배열의 크기
(원소의 개수)

예제 7-1: 배열의 크기 구하기 (2)

```
14      for (i = 0; i < size; i++)
15          arr[i] = 0;
16
17      for (i = 0; i < size; i++)
18          printf("%d ", arr[i]);
19      printf("\n");
20  }
```

배열은 주로 for문과 함께 사용된다.

실행 결과

```
배열의 바이트 크기: 20
배열의 크기: 5
0 0 0 0 0
```

배열의 크기를 변수에 구해서 사용하는 이유

- 배열의 크기를 변수에 구해두고 사용하는 대신 매크로 상수를 이용할 수도 있다.

배열의 크기로 리터럴 상수를 직접 사용하는 경우

```
int arr[5];
int i;

for (i = 0; i < 5; i++)
    arr[i] = 0;

for (i = 0; i < 5; i++)
    printf("%d ", arr[i]);
printf("\n");
```

배열의 크기를 변경하려면 코드 전체를 모두 수정해야 한다.

10

10

10

배열의 크기를 변수에 구해서 사용하는 경우

```
int arr[5];
int size = sizeof(arr)/sizeof(arr[0]);
int i;

for (i = 0; i < size; i++)
    arr[i] = 0;

for (i = 0; i < size; i++)
    printf("%d ", arr[i]);
printf("\n");
```

10

배열의 크기를 변경하려면 배열의 선언문만 수정하면 된다.

size는 그대로 사용할 수 있다.

size는 그대로 사용할 수 있다.

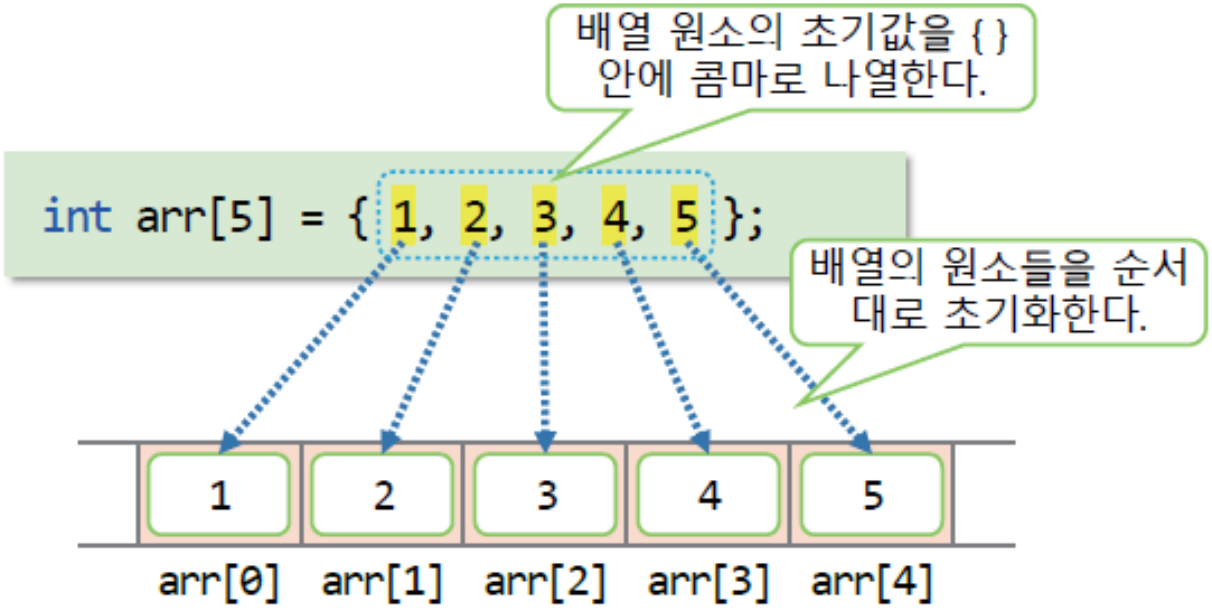
배열의 초기화 (1)

- 배열을 초기화하려면 배열 이름 다음에 =을 쓰고 { } 안에 배열 원소의 초기값을 콤마(,)로 나열한다.

형식 데이터형 배열명[크기] = {초기값목록};

사용예

```
int arr[5] = {1, 2, 3, 4, 5};
double data[100] = {0};
char name[32] = "Jin";
```



배열의 초기화 (2)

- { } 안에 나열된 초기값은, 배열의 0번째 원소부터 순서대로 초기화하는 데 사용된다.
- 배열의 크기보다 초기값을 부족하게 지정하면, 나머지 원소는 0으로 초기화한다.

```
int x[5] = { 1, 2, 3 };
```

{1, 2, 3, 0, 0}으로 초기화

- 초기값을 원소의 개수보다 많이 지정하면 컴파일 에러가 된다.

```
int x[5] = { 1, 2, 3, 4, 5, 6 };
```

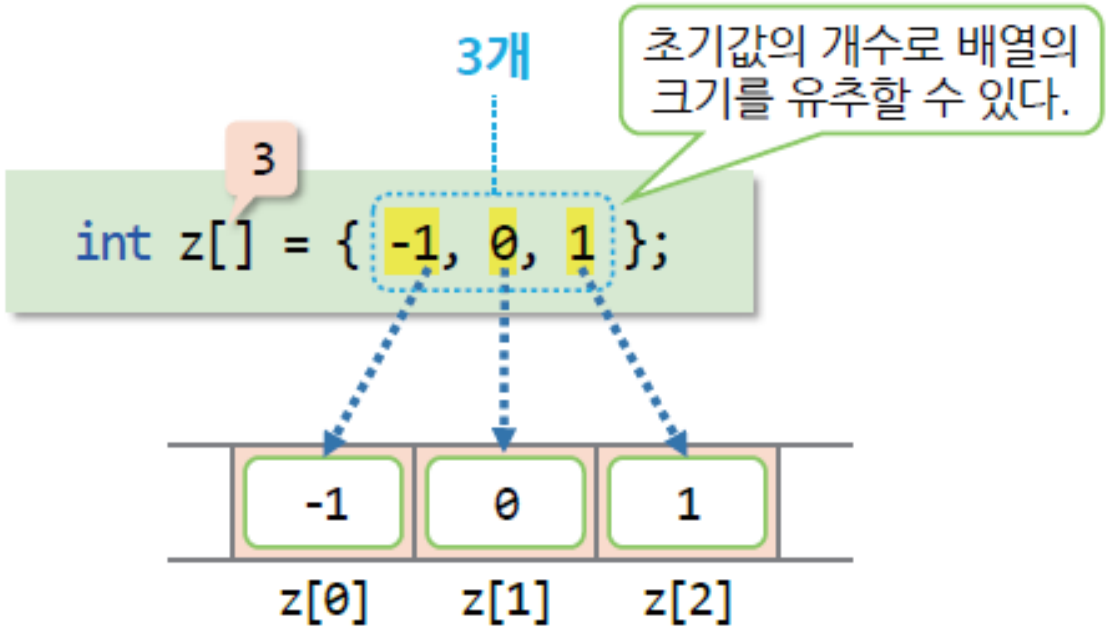
- { } 안을 비워 두면 컴파일 에러가 발생한다. 배열 전체를 0으로 초기화하려면 { } 안에 0을 써준다.

```
int y[5] = {};
```

```
int y[5] = { 0 };
```

배열의 초기화 (3)

- 배열의 초기값을 지정하는 경우에는 배열의 크기를 생략할 수 있다.
  - 배열의 크기를 생략하면, 초기값의 개수가 배열의 크기가 된다.



예제 7-2: 배열의 초기화 (1)

```
01  #include <stdio.h>
02  #define ARR_SIZE 5
03
04  int main(void)
05  {
06      int arr[ARR_SIZE] = { 1, 2, 3, 4, 5 };
07      int x[ARR_SIZE] = { 1, 2, 3 };
08      int y[ARR_SIZE] = { 0 };
09      int z[] = { -1, 0, 1 };
10      int i, size;
11
12      printf("arr = ");
13      for (i = 0; i < ARR_SIZE; i++)
14          printf("%3d ", arr[i]);
15      printf("\n");
```

배열의 크기만큼 초기값을 지정한다.

나머지 원소는 0으로 초기화된다.

배열 전체를 0으로 초기화한다.

초기화하는 경우에는 배열의 크기를 생략할 수 있다.

예제 7-2: 배열의 초기화 (2)

```
17      printf("x   = ");
18      for (i = 0; i < ARR_SIZE; i++)
19          printf("%3d ", x[i]);
20      printf("\n");
21
22      printf("y   = ");
23      for (i = 0; i < ARR_SIZE; i++)
24          printf("%3d ", y[i]);
25      printf("\n");
26
27      printf("z   = ");
28      size = sizeof(z) / sizeof(z[0]);
29      for (i = 0; i < size; i++)
30          printf("%3d ", z[i]);
31      printf("\n");
32  }
```

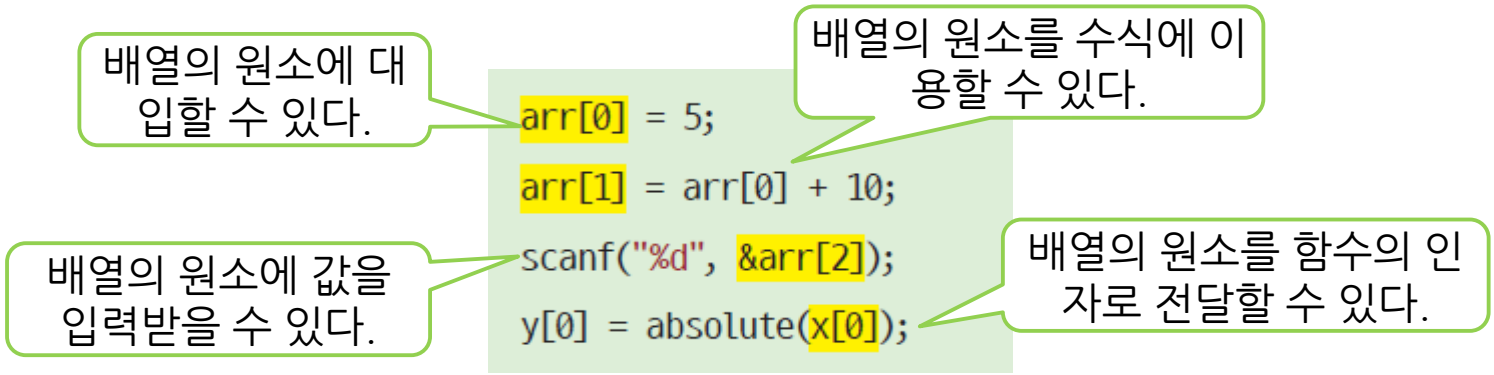
z 배열의 크기를 구한다.

실행 결과

```
arr =   1   2   3   4   5
x   =   1   2   3   0   0
y   =   0   0   0   0   0
z   =  -1   0   1
```

배열 원소의 사용 (1)

- 배열의 원소도 변수이다.
- *type*형 배열의 원소는 *type*변수가 사용되는 모든 곳에서 사용될 수 있다.



- 배열의 인덱스에는 변수나 변수를 포함한 수식을 사용할 수 있다.

```
for (i = 2; i < ARR_SIZE; i++)
    arr[i] = arr[i - 2] + arr[i - 1];
```

예제 7-3: 배열 원소의 사용 (1)

```
01  #include <stdio.h>
02  #define ARR_SIZE 5
03
04  unsigned int absolute(int x)
05  {
06      return x > 0 ? x : -x;
07  }
08
09  int main(void)
10  {
11      int x[ARR_SIZE] = { -4, 0, 28, 3, -12 };
12      unsigned int y[ARR_SIZE] = { 0 };
13      int i;
14
```

절대값을 구하는 함수

배열 전체를 0으로 초기화한다.

예제 7-3: 배열 원소의 사용 (2)

```
15      for (i = 0; i < ARR_SIZE; i++)
16          y[i] = absolute(x[i]);
17
18
19      for (i = 0; i < ARR_SIZE; i++)
20          printf("%d ", y[i]);
21      printf("\n");
22  }
```

배열의 원소를 함수의 인자로 전달한다.

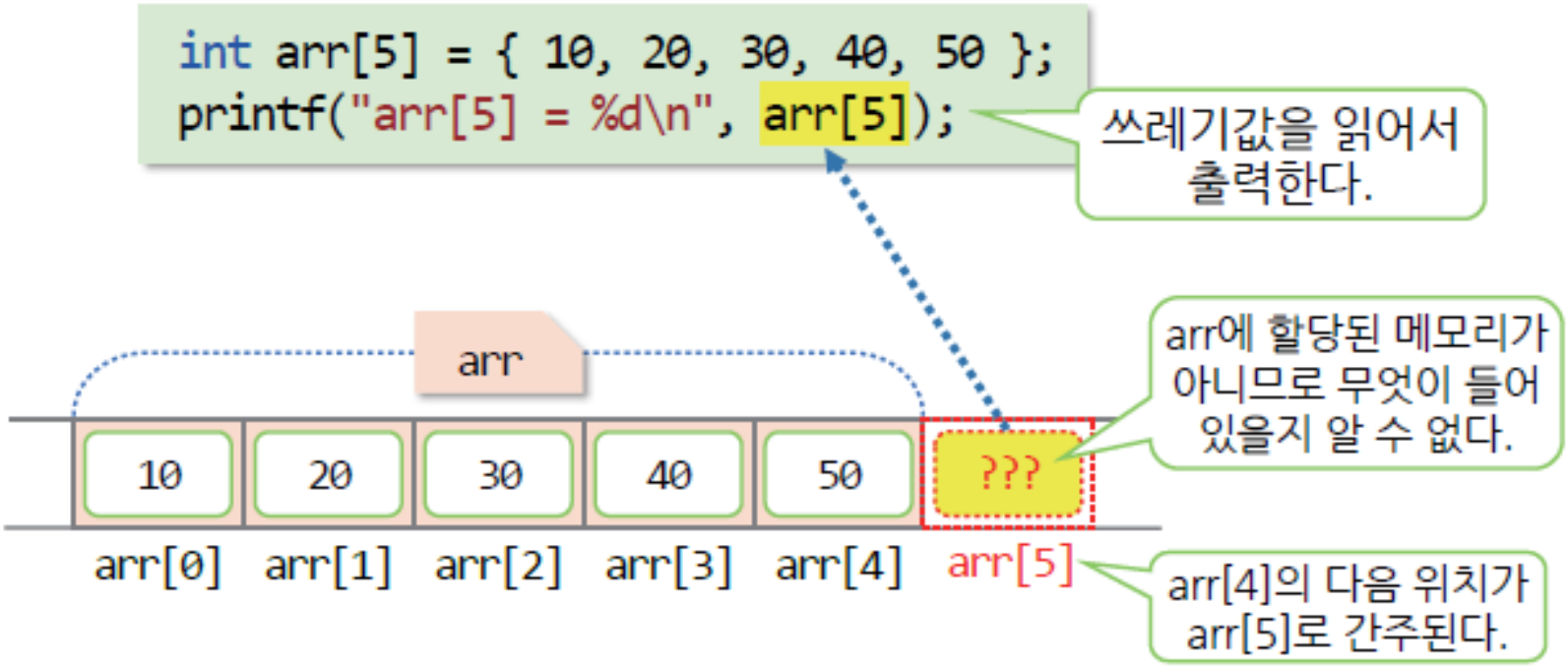
함수의 리턴값을 배열의 원소에 대입한다.

실행 결과

```
4 0 28 3 12
```

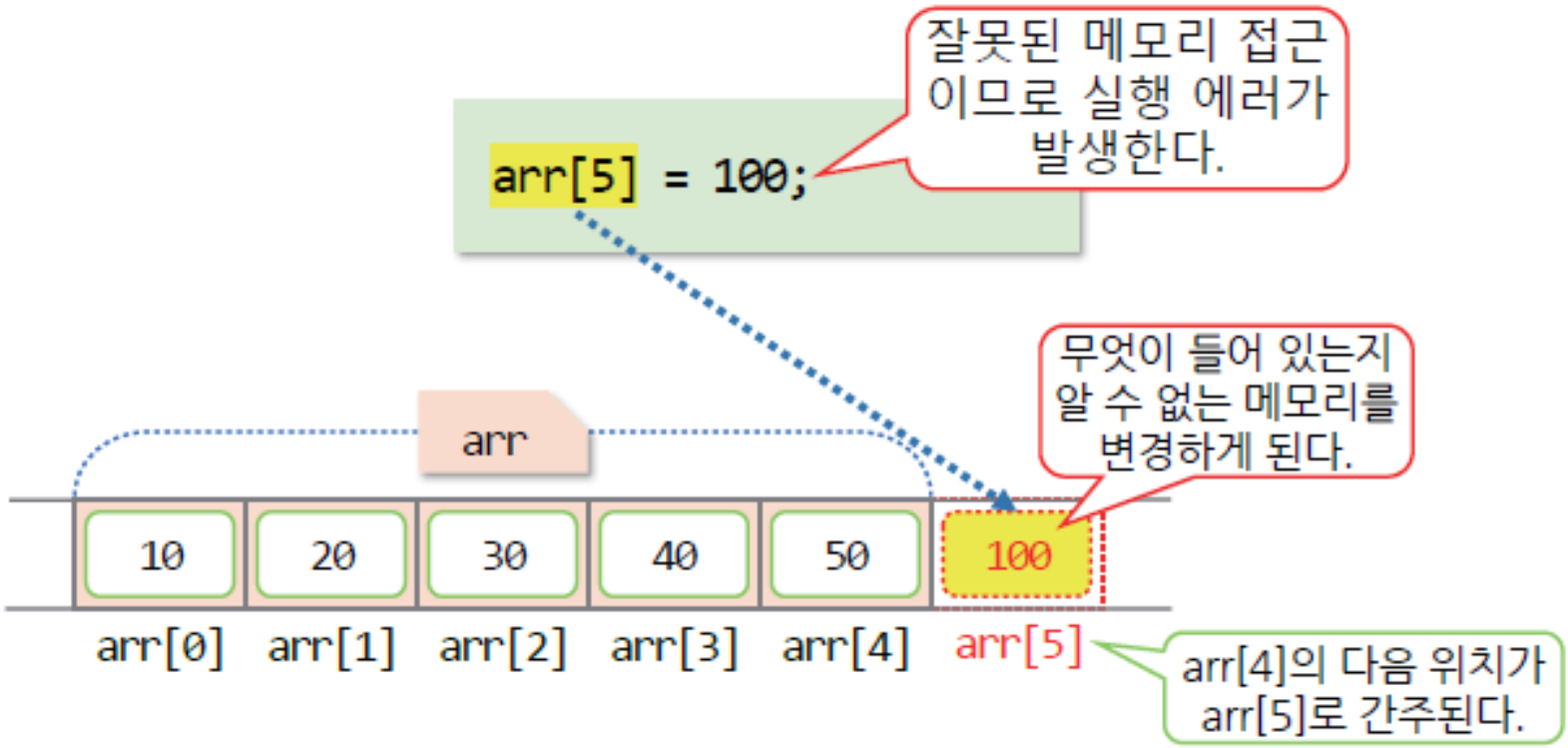
잘못된 인덱스의 사용 (1)

- 인덱스의 유효 범위는 0~(배열의 크기–1)이다.
- 잘못된 인덱스를 사용하지 않도록 주의해야 한다.
  - 잘못된 인덱스를 사용해도 컴파일 에러가 발생하지 않는다.
- 크기가 5인 arr 배열에 대해서 arr[5]를 읽어오면 쓰레기값이 된다.



잘못된 인덱스의 사용 (2)

- arr[5]에 값을 저장하려고 하면 엉뚱한 변수의 값이 변경되거나 프로그램이 죽는다. → 실행 에러



예제 7-4: 피보나치 수열 구하기 (1)

```c
#include <stdio.h>
#define ARR_SIZE 10

int main(void)
{
    int arr[ARR_SIZE] = {1, 1};
    int i;

    for (i = 2; i < ARR_SIZE; i++)
        arr[i] = arr[i - 2] + arr[i - 1];
```

{1, 1, 0, 0, ...}으로 초기화 한다.

배열의 인덱스로 정수식을 사용할 수 있다.

예제 7-4: 피보나치 수열 구하기 (2)

```
12      for (i = 0; i < ARR_SIZE; i++)
13          printf("%d ", arr[i]);
14      printf("\n");
15
16      printf("arr[10] = %d\n", arr[10]);
17      arr[10] = 100;
18  }
```

잘못된 인덱스로 배열의 원소를 읽어온다.

잘못된 인덱스로 배열의 원소를 변경하면 실행 에러가 발생한다.

실행 결과

```
1 1 2 3 5 8 13 21 34 55
arr[10] = -858993460
```

잘못된 인덱스로 배열의 원소를 출력하면 쓰레기 값을 출력한다.

배열의 복사

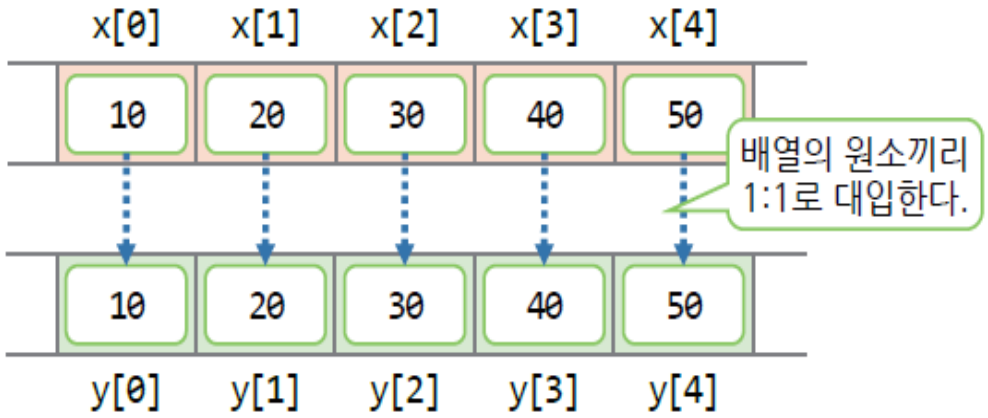
- 데이터형과 크기가 같은 경우에도 배열에 다른 배열을 대입할 수 없다.

```
int x[ARR_SIZE] = { 10, 20, 30, 40, 50 };
int y[ARR_SIZE] = { 0 };
y = x;
```

배열 이름으로 배열 전체를 대입할 수 없다.

- 배열을 복사하려면 배열의 모든 원소에 대하여 원소끼리 대입한다.

```
for (i = 0; i < ARR_SIZE; i++)
    y[i] = x[i];
```



예제 7-5: 배열의 복사

```c
#include <stdio.h>
#define ARR_SIZE 5

int main(void)
{
    int x[ARR_SIZE] = { 10, 20, 30, 40, 50 };
    int y[ARR_SIZE] = { 0 };
    int i;

    for (i = 0; i < ARR_SIZE; i++)
        y[i] = x[i];

    for (i = 0; i < ARR_SIZE; i++)
        printf("%d ", y[i]);
    printf("\n");
}
```

실행 결과

```
10 20 30 40 50
```

x와 y는 데이터형과 크기가 같은 배열이다.

배열의 원소끼리 1:1로 대입한다.

배열의 비교

- 두 배열이 같은지 비교하기 위해서 == 연산자로 직접 배열을 비교하면 배열의 시작 주소를 비교한다.
  - 배열의 이름은 배열의 시작 주소이기 때문이다.

x와 y의 주소가 같은지 비교한다.

```
int x[ARR_SIZE] = { 10, 20, 30, 40, 50 };
int y[ARR_SIZE] = { 10, 20, 30, 40, 50 };
if (x == y)
    printf("두 배열의 주소가 같습니다.\n");
```

- 배열의 내용이 같은지 비교하려면 for문을 이용해서 원소끼리 비교해야 한다.
  - 모든 원소의 값이 같으면, 배열 전체의 내용이 같다.

예제 7-6: 배열의 비교 (1)

```
01  #include <stdio.h>
02  #define ARR_SIZE 5
03
04  int main(void)
05  {
06      int x[ARR_SIZE] = { 10, 20, 30, 40, 50 };
07      int y[ARR_SIZE] = { 10, 20, 30, 40, 50 };
08      int i;
09      int is_equal;
10
11      if (x == y)
12          printf("두 배열의 주소가 같습니다.\n");
13      else
14          printf("두 배열의 주소가 다릅니다.\n");
```

배열이 같은지를 나타내는 변수

x와 y의 주소가 같은지 비교한다.

예제 7-6: 배열의 비교 (2)

```
16      is_equal = 1;
17      for (i = 0; i < ARR_SIZE; i++) {
18          if (x[i] != y[i]) {
19              is_equal = 0;
20              break;
21          }
22      }
23      if (is_equal == 1)
24          printf("두 배열의 내용이 같습니다.\n");
25      else
26          printf("두 배열의 내용이 다릅니다.\n");
27  }
```

배열의 모든 원소가 같은지 비교한다.

서로 다른 원소가 있으면 더 이상 비교할 필요가 없다.

실행 결과

```
두 배열의 주소가 다릅니다.
두 배열의 내용이 같습니다.
```

다차원 배열의 개념 (1)

- 다차원 배열은 원소에 접근할 때 2개 이상의 인덱스를 사용한다.
  - 행렬이나 표, 폭과 높이가 있는 이미지 데이터 등을 나타내는 데 이용

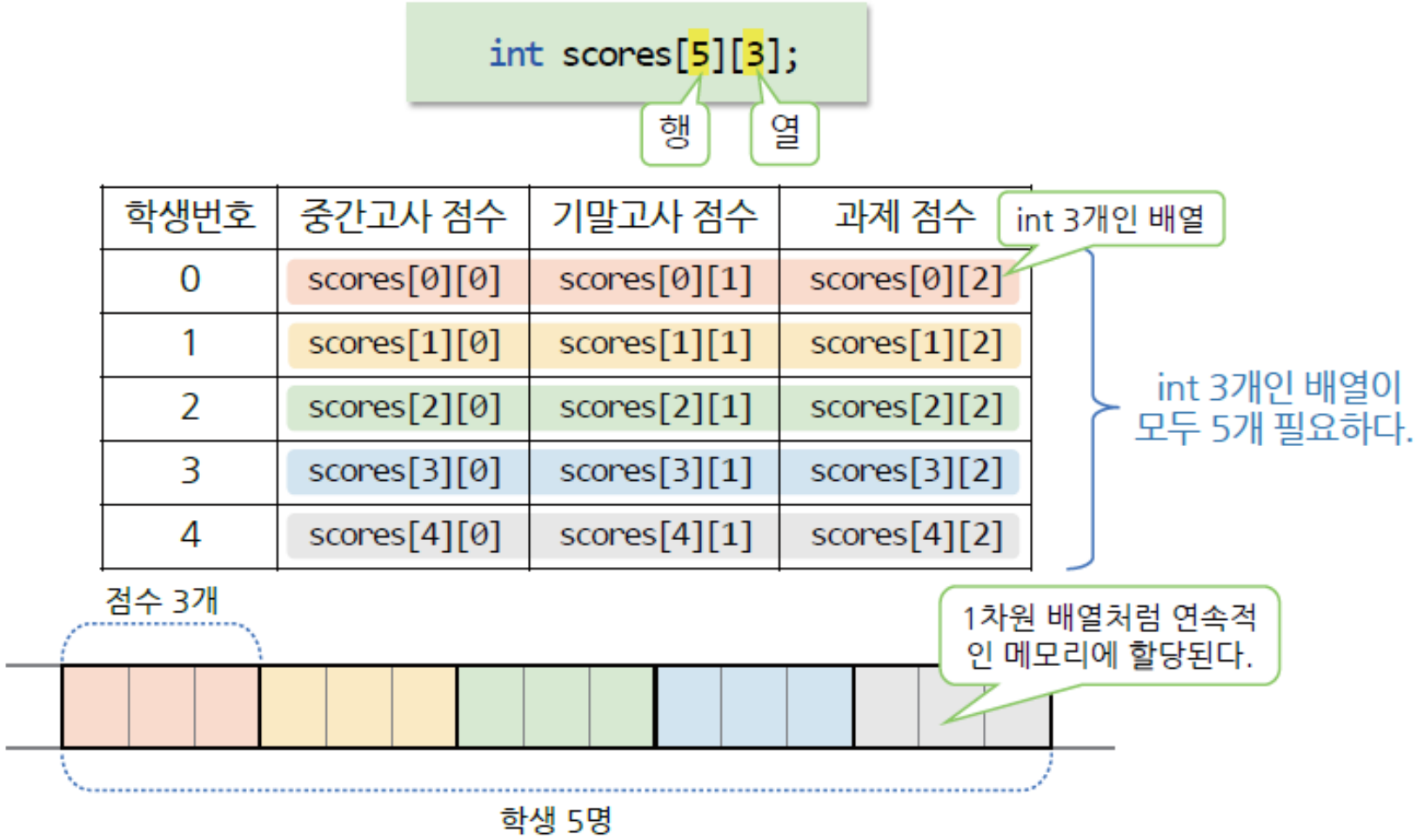
```
int scores[5][3];
```

점수 3개를 저장하는 배열이 5개 필요하다. (점수 3개 × 학생 5명)

- 2차원 배열은 행(row)과 열(column)의 개념으로 이해할 수 있다.
- 다차원 배열의 차수에는 제한이 없다.

다차원 배열의 개념 (2)

- 다차원 배열도 1차원 배열처럼 메모리에 연속적으로 할당된다.



2차원 배열의 선언 (1)

형식 데이터형 배열명[행크기][열크기];

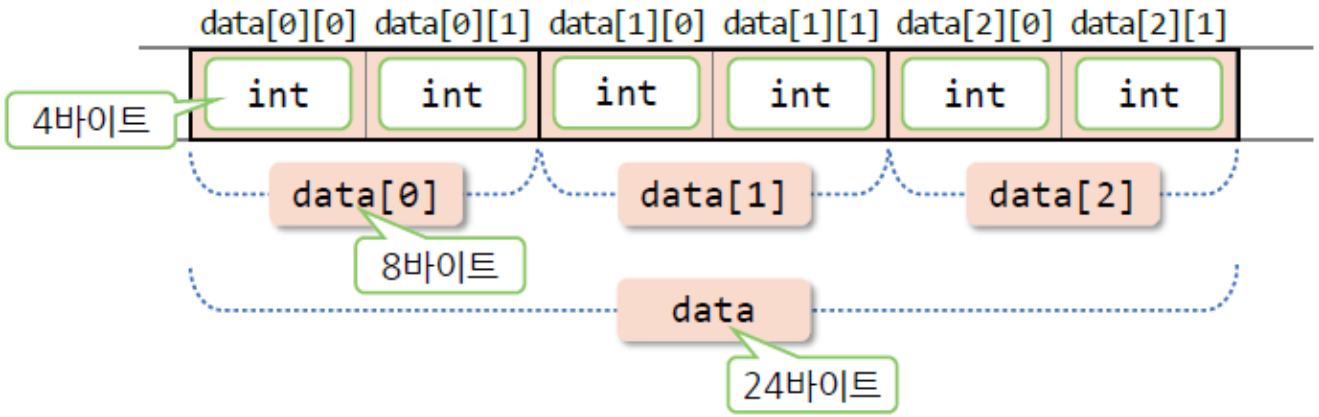
사용예

```
int scores[5][3];
double matrix[4][4];
char passwords[10][32];
```

원소의 개수는
'(열 크기)×(행 크기)'개

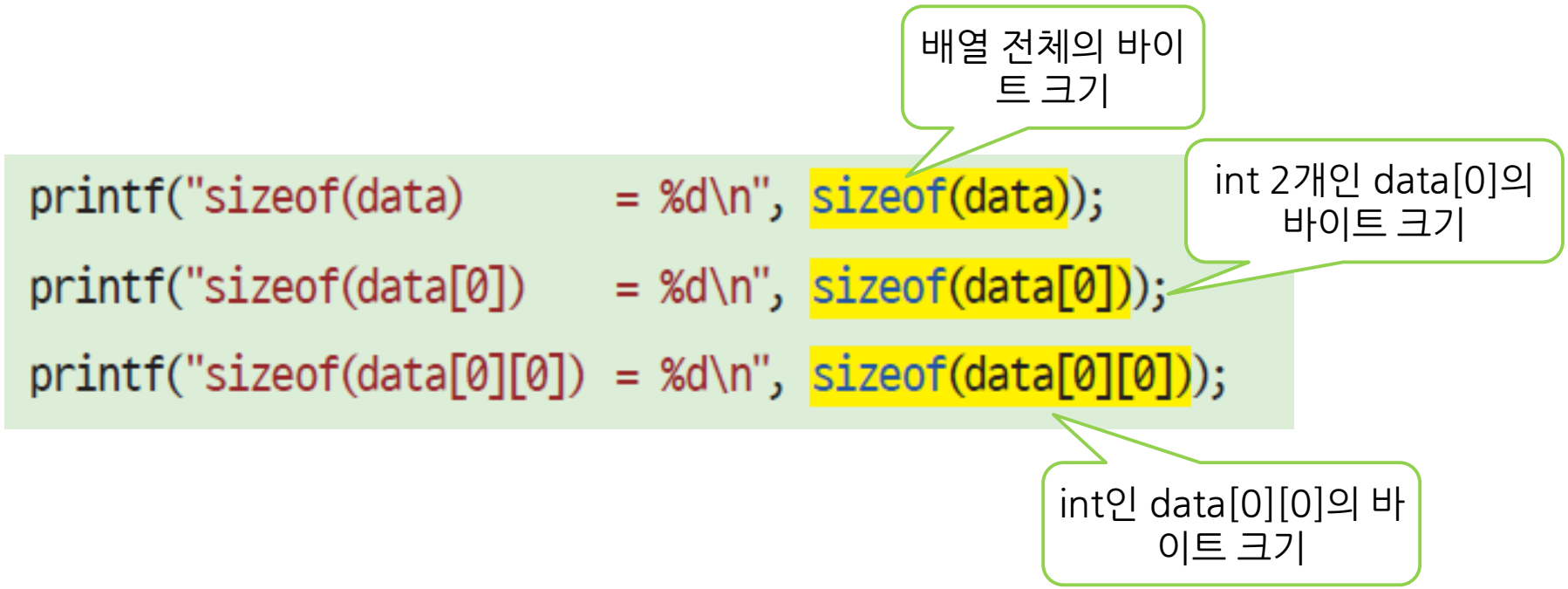
```
#define ROW 3
#define COL 2
int data[ROW][COL];
```

배열의 크기를 매크로 상수로 지정한다.



2차원 배열의 선언 (2)

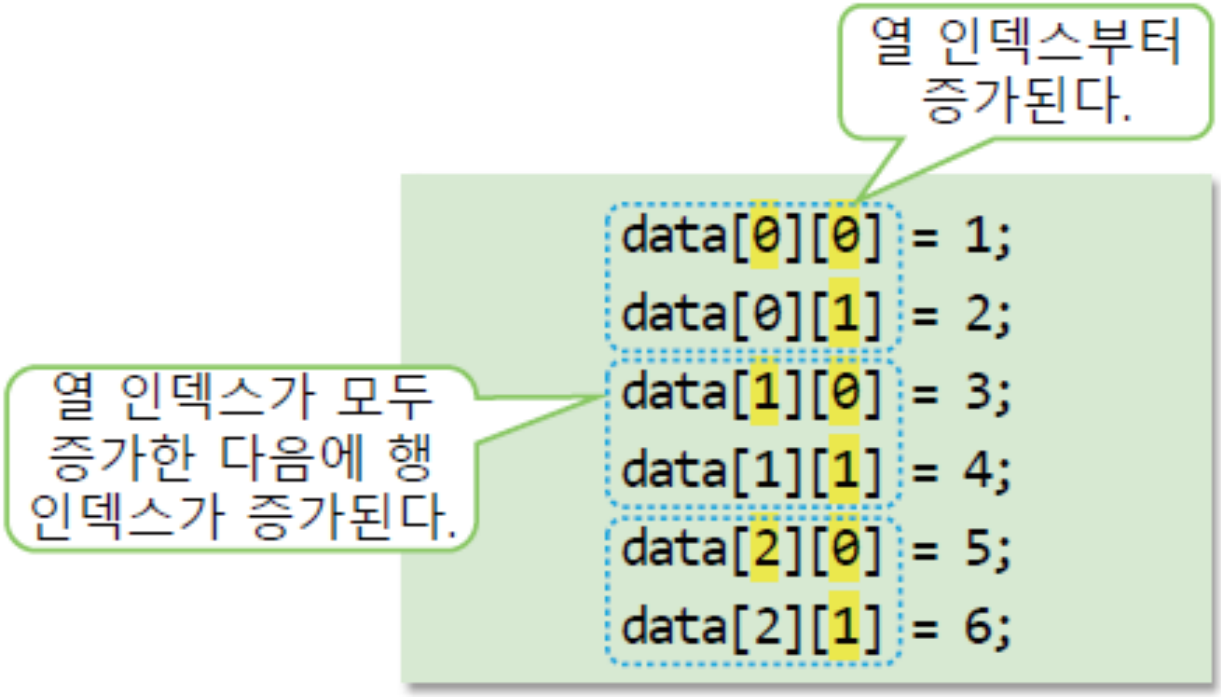
- 2차원 배열의 원소도 메모리에 연속적으로 할당된다.




```
printf("sizeof(data)      = %d\n", sizeof(data));
printf("sizeof(data[0])   = %d\n", sizeof(data[0]));
printf("sizeof(data[0][0]) = %d\n", sizeof(data[0][0]));
```

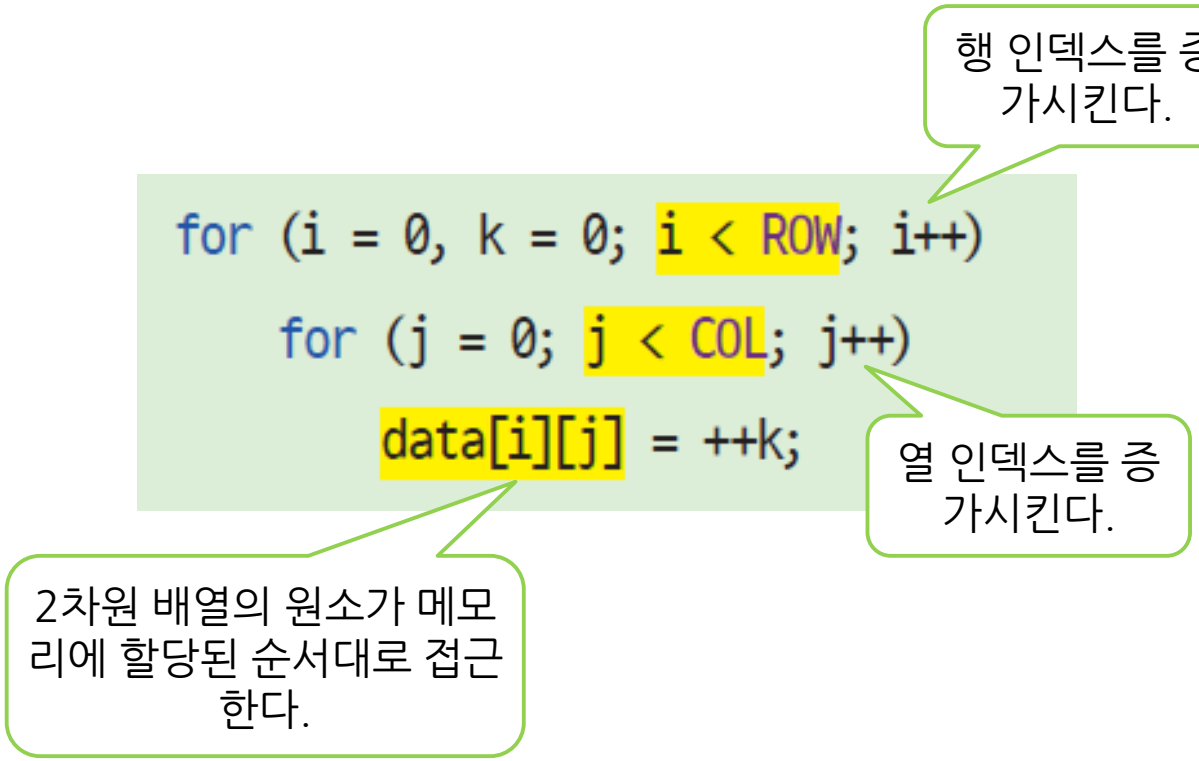
2차원 배열의 사용 (1)

- 2차원 배열의 원소에 접근할 때는 인덱스를 2개 사용한다.
  - 2차원 배열에서는 열 인덱스가 먼저 증가되고, 그 다음에 행 인덱스가 증가된다.



2차원 배열의 사용 (2)

- 2차원 배열은 중첩된 for와 함께 사용할 수 있다.
  - 안쪽 for문을 이용해서 열 인덱스를 증가시키고, 바깥쪽 for문을 이용해서 행 인덱스를 증가시키면 2차원 배열의 원소가 메모리에 할당된 순서대로 접근할 수 있다.



예제 7-7: 2차원 배열의 선언 및 사용 (1)

```
01  #include <stdio.h>
02  #define ROW 3
03  #define COL 2
04
05  int main(void)
06  {
07      int data[ROW][COL];
08      int i, j, k;
09
10      for (i = 0, k = 0; i < ROW; i++)
11          for (j = 0; j < COL; j++)
12              data[i][j] = ++k;
13
```

int[2]를 3개 메모리에 할당한다.

data 배열의 각 원소가 할당된 순서대로 접근한다.

예제 7-7: 2차원 배열의 선언 및 사용 (2)

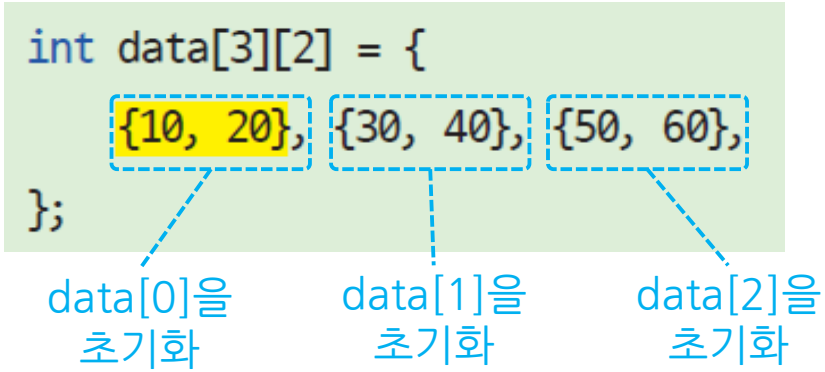
```
14      for (i = 0; i < ROW; i++) {
15          for (j = 0; j < COL; j++)
16              printf("%3d ", data[i][j]);
17          printf("\n");
18      }
19
20      printf("sizeof(data)        = %d\n", sizeof(data));
21      printf("sizeof(data[0])     = %d\n", sizeof(data[0]));
22      printf("sizeof(data[0][0]) = %d\n", sizeof(data[0][0]));
23  }
```

실행 결과

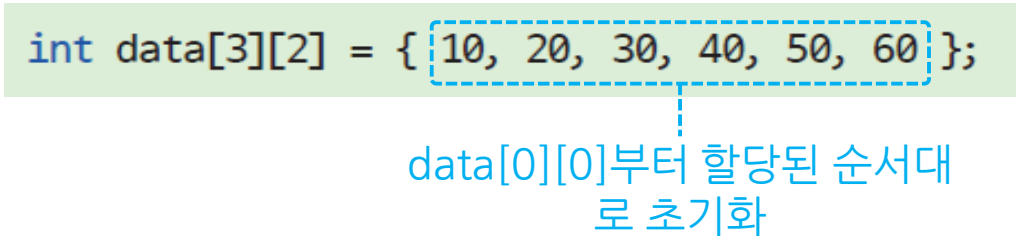
```
  1   2
  3   4
  5   6
sizeof(data)        = 24
sizeof(data[0])     = 8
sizeof(data[0][0]) = 4
```

2차원 배열의 초기화 (1)

- 초기값을 열 크기의 개수만큼씩 { }로 묶어서 다시 { } 안에 나열한다.



- 1차원 배열처럼 { } 안에 값만 나열할 수도 있다.
  - 배열의 원소가 메모리에 할당된 순서대로 초기화한다.



2차원 배열의 초기화 (2)

- 초기값을 생략하면 나머지 원소를 0으로 초기화한다.

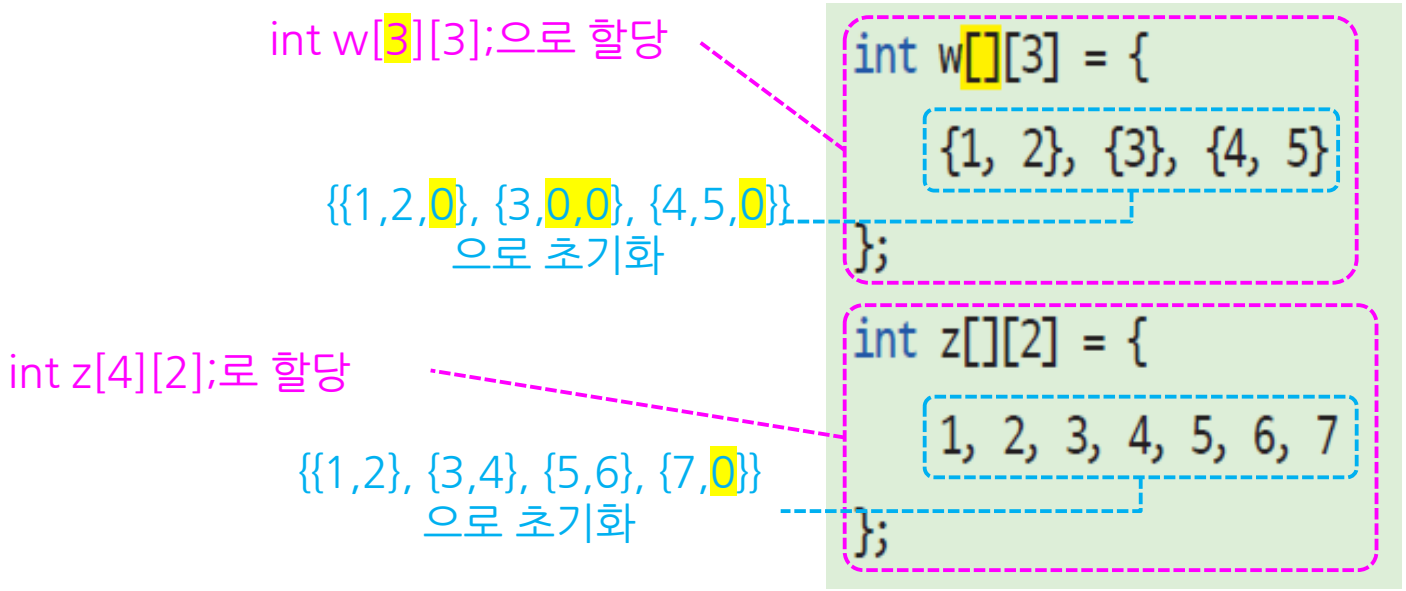
```
int x[4][3] = {
    {1, 2, 3},
    {4, 5},
    {6}
};
int y[3][2] = { 1, 2, 3 };
```

{{1,2,3}, {4,5,0}, {6,0,0}, {0,0,0}}
으로 초기화

{{1, 2}, {3, 0}, {0, 0}}
으로 초기화

2차원 배열의 초기화 (3)

- 2차원 배열을 초기화할 때는, 배열의 행 크기를 생략할 수 있다.



- 2차원 배열의 열 크기는 생략할 수 없다.

```
int v[3][] = {
    {1, 2}, {3}, {4, 5, 6}
};
```

열 크기를 유추할 수 없으므로 컴파일 에러

예제 7-8: 2차원 배열의 초기화

```
01  #include <stdio.h>
02  #define COL 2

04  int main(void)
05  {
06      int data[][COL] = {
07          {10, 20}, {30, 40}, {50, 60},
08      };
09      int row_size = sizeof(data) / sizeof(data[0]);
10      int i, j;

12      for (i = 0; i < row_size; i++) {
13          for (j = 0; j < COL; j++)
14              printf("%3d ", data[i][j]);
15          printf("\n");
16      }
17  }
```

int data[3][2]로

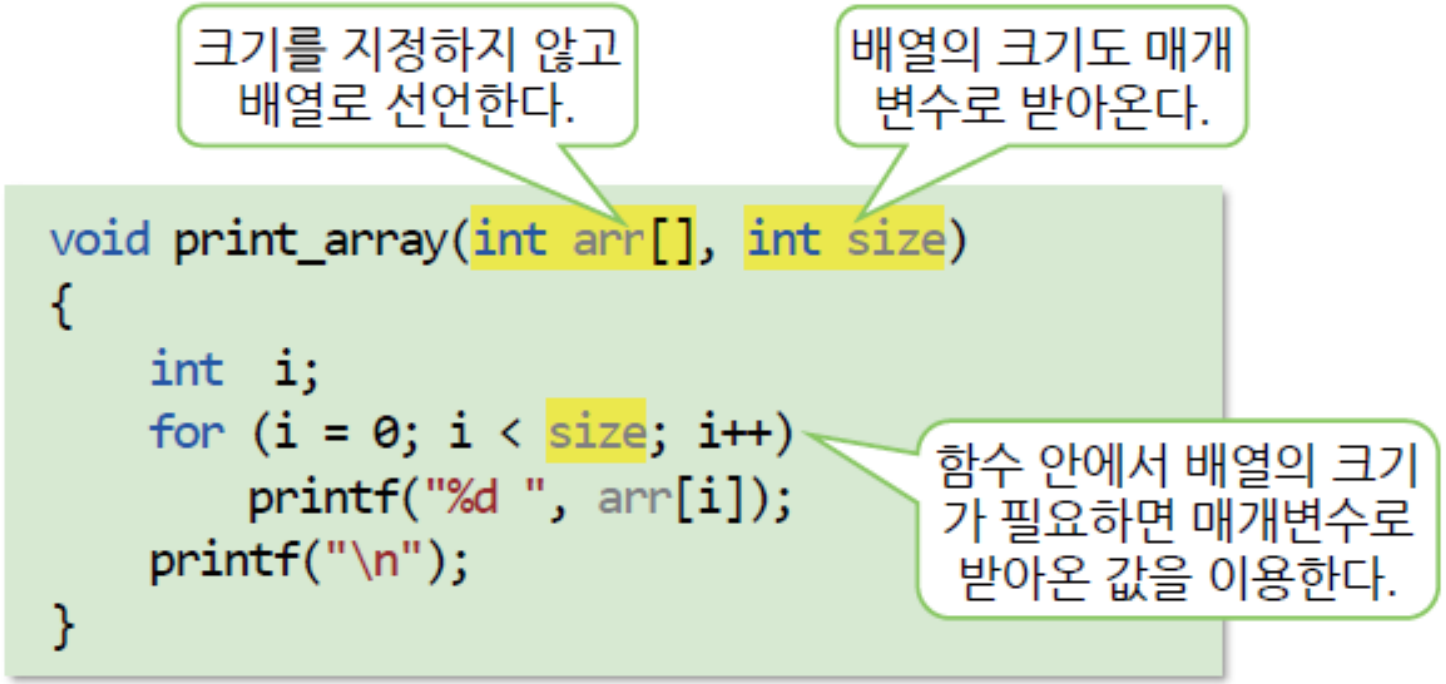
행 크기를 구한다.

실행 결과

```
 10  20
 30  40
 50  60
```

배열을 매개변수로 갖는 함수의 정의

- 함수의 매개변수로 배열을 선언할 때, 배열의 크기는 생략한다.
- 함수 안에서 배열의 크기가 필요하면 배열의 크기도 매개변수로 받아온다.



배열을 매개변수로 갖는 함수를 정의하는 방법

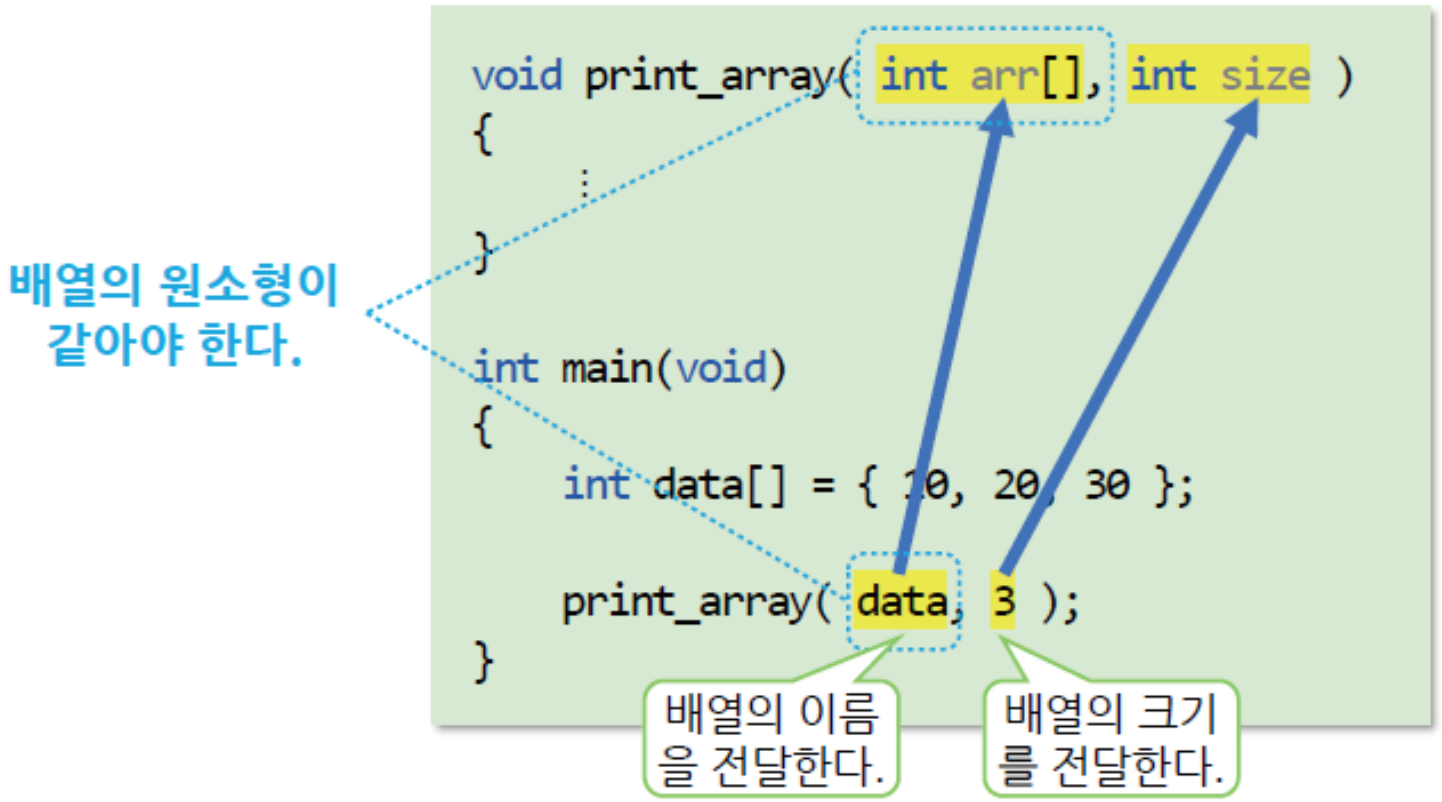
① 함수의 매개변수로 배열을 선언할 때는 배열의 원소형, 매개변수명(배열명)과 [ ]를 적어준다. [ ] 안에 배열의 크기를 쓰지 않고 비워둔다.

② 배열의 크기를 전달받기 위한 정수형의 매개변수가 필요하다.

③ 함수 안에서 배열의 크기가 필요할 때는 매개변수로 전달받은 배열의 크기를 이용한다.

배열을 매개변수로 갖는 함수의 호출 (1)

- 배열의 이름과 배열의 크기를 인자로 전달한다.
  - [ ] 없이 배열의 이름만 써준다.



배열을 매개변수로 갖는 함수의 호출 (2)

- 함수 안에서는 항상 매개변수로 전달받은 배열의 크기를 이용한다.

```
int x[5] = { 1, 2, 3, 4, 5 };
print_array(x, 3);
```

x가 크기가 3인 배열인 것처럼 {1, 2, 3}만 출력한다.

- 매개변수의 원소형과 인자로 전달하는 배열의 원소형이 같아야 한다.

```
float grades[3] = { 4.0, 4.3, 3.7 };
print_array(grades, 3);
```

인자와 매개변수의 원소형이 같아야 한다.

예제 7-9: 배열의 출력 (1)

```
01  #include <stdio.h>
02  void print_array(int arr[], int size);
03
04  int main(void)
05  {
06      int data[3] = { 10, 20, 30 };
07      int x[] = { 1, 2, 3, 4, 5 };
08      int size = sizeof(x) / sizeof(x[0]);
09
10      printf("data = ");
11      print_array(data, 3);
12
13      printf("x    = ");
14      print_array(x, size);
```

함수 선언

x 배열의 크기

배열 이름과 크기를 인자로 전달한다.

예제 7-9: 배열의 출력 (2)

```
16      printf("x    = ");
17      print_array(x, 3);
18  }
19
20  void print_array(int arr[], int size)
21  {
22      int  i;
23      for (i = 0; i < size; i++)
24          printf("%d ", arr[i]);
25      printf("\n");
26  }
```

x 배열을 크기가 3인 배열처럼 출력한다.

배열형의 매개변수에서 크기는 생략한다.

배열의 크기는 별도의 매개변수로 받아와야 한다.

실행 결과

```
data = 10 20 30
x    = 1 2 3 4 5
x    = 1 2 3
```

배열의 탐색과 정렬

- **탐색(search)** 또는 검색
  - 주어진 데이터 집합에서 조건이 만족하는 데이터를 찾는 것
  - 검색할 상품명을 입력하면 여러 쇼핑몰의 상품 정보 중에서 상품명이 일치하는 항목을 찾아서 화면에 표시하는 기능

- **정렬(sort)**
  - 주어진 데이터 항목을 지정된 순서로 나열하는 것
  - 가격 비교 사이트에서 검색 결과로 표시된 상품 목록을 낮은 가격 순으로 보거나 높은 가격 순으로 확인하는 기능

배열의 탐색

- 탐색 키(key)와 같은 값을 가진 원소를 찾는다.

- 순차 탐색(sequential search)
  - 배열의 0번째 원소부터 순서대로 탐색 키와 비교해서 값이 같은 원소를 찾는다.

예제 7-10: 배열의 탐색 (1)

```
01  #define  _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  void print_array(int arr[], int size)
05  {
06      int  i;
07      for (i = 0; i < size; i++)
08          printf("%d ", arr[i]);
09      printf("\n");
10  }
11
12  int main(void)
13  {
14      int data[] = { 12, 34, 51, 22, 91, 12, 15 };
```

크기를 생략한 배열을 선언한다.

예제 7-10: 배열의 탐색 (2)

```
15      int size, i;
16      int key;
17
18      size = sizeof(data) / sizeof(data[0]);
19      printf("data = ");
20      print_array(data, size);
21
22      printf("찾을 값(키)? ");
23      scanf("%d", &key);
24      for (i = 0; i < size; i++) {
25          if (data[i] == key)
26              printf("찾은 원소의 인덱스: %d\n", i);
27      }
28  }
```

탐색 키를 입력받는다.

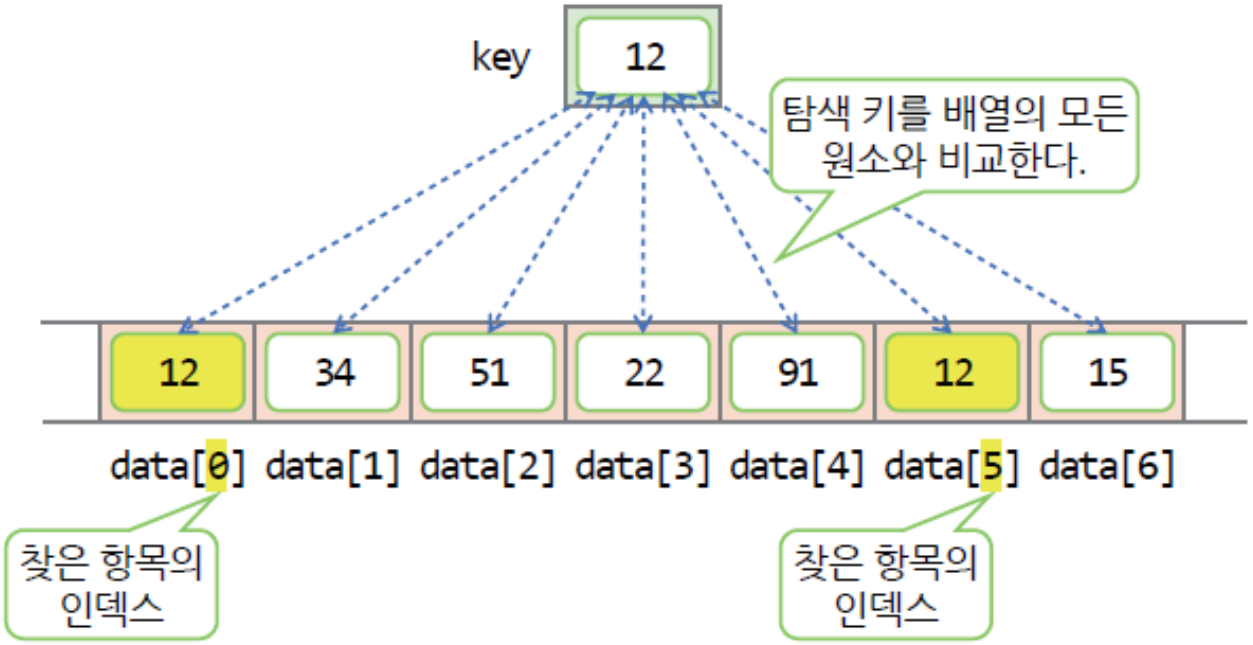
탐색 키와 값이 같은 원소를 찾는다.

실행 결과

```
data = 12 34 51 22 91 12 15
찾을 값(키)? 12
찾은 항목의 인덱스 : 0
찾은 항목의 인덱스 : 5
```

[예제 7-10]의 실행 과정

- 탐색 키와 값이 같은 원소를 모두 찾아서 인덱스를 출력한다.



- 탐색 키와 일치하는 첫 번째 원소만 찾고 탐색을 종료하려면 break로 for문을 탈출한다.

배열의 정렬

- 원소들을 비교해서 크기가 커지는 순서 또는 작아지는 순서로 나열한다.
  - 오름차순(ascending order) : 크기가 커지는 순서
  - 내림차순(descending order) : 크기가 작아지는 순서

- 선택 정렬(selection sort)
  - 전체 배열의 원소 중 가장 작은 값을 선택해서 배열의 0번 원소로 옮기고, 그 다음 작은 값을 선택해서 배열의 1번 원소로 옮기는 식으로 진행된다.

예제 7-11: 오름차순 선택 정렬 (1)

```c
#include <stdio.h>
#define SIZE 5

void print_array(int arr[], int size)
{
    int  i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main(void)
{
    int data[SIZE] = { 52, 31, 28, 17, 46 };
    int i, j, temp;
    int index_min;
```

실행 결과

```
i = 0 일때 정렬 결과 : 17 31 28 52 46
i = 1 일때 정렬 결과 : 17 28 31 52 46
i = 2 일때 정렬 결과 : 17 28 31 52 46
i = 3 일때 정렬 결과 : 17 28 31 46 52
```

아직 정렬되지 않은 원소 중
가장 작은 원소의 인덱스

예제 7-11: 오름차순 선택 정렬 (2)

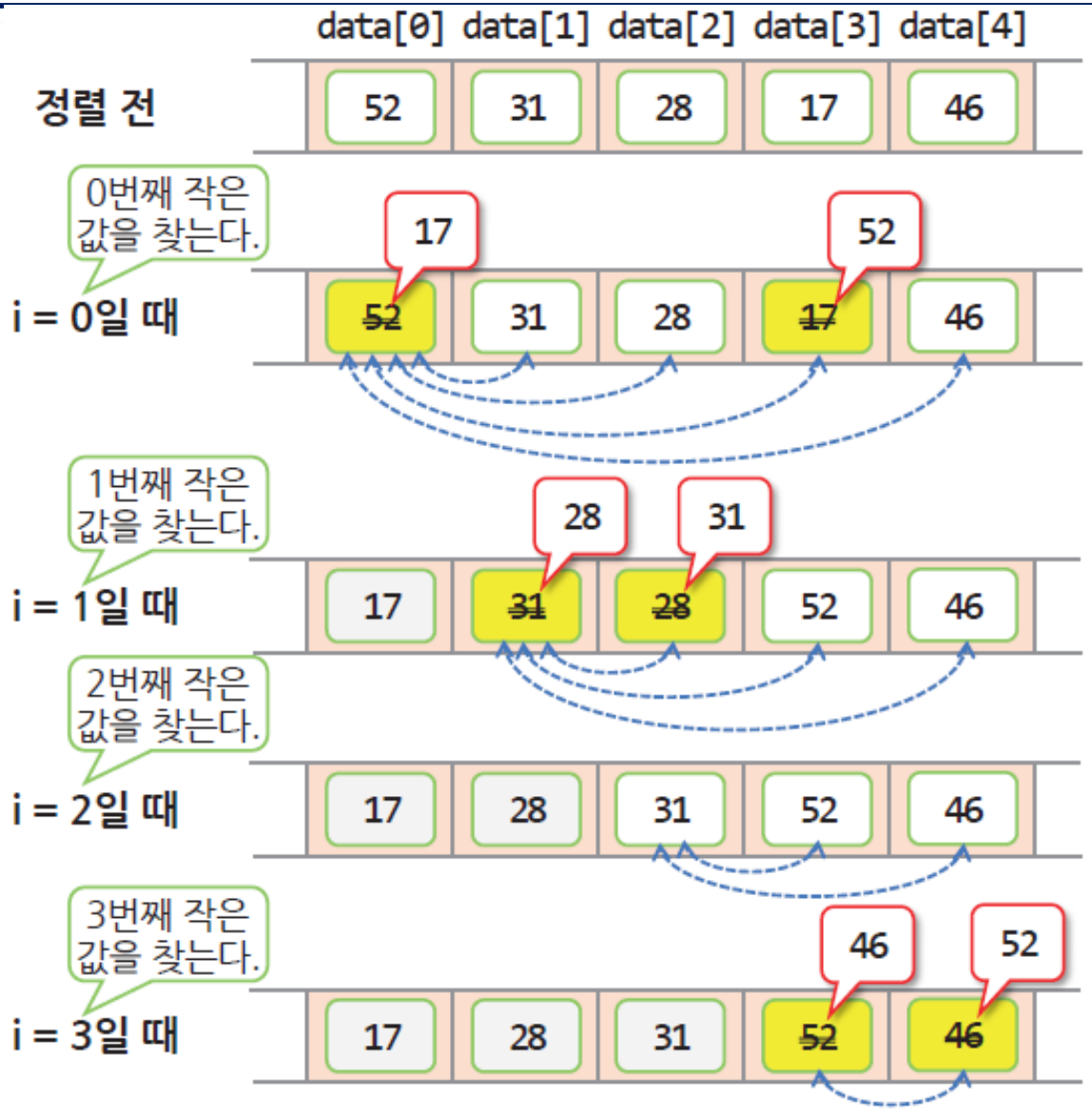
```
18      for (i = 0; i < SIZE - 1; i++) {
19          index_min = i;
20          for (j = i + 1; j < SIZE; j++) {
22              if (data[index_min] > data[j])
23                  index_min = j;
24          }
25          if (i != index_min) {
26              temp = data[i];
27              data[i] = data[index_min];
28              data[index_min] = temp;
29          }
31          printf("i = %d 일때 정렬 결과 : ", i);
32          print_array(data, SIZE);
33      }
34  }
```

data[0]~data[i-1]는 정렬된 상태

data[i]~data[SIZE-1] 중에서 가장 작은 원소의 인덱스를 찾는다.

data[i]를 data[index_min]와 맞바꾼다.

선택 정렬의 수행 과정



data[0]과 data[1]~data[4]를 비교해서 data[0]을 가장 작은 data[3]과 맞바꾼다.

data[1]과 data[2]~data[4]를 비교해서 data[1]을 가장 작은 data[2]와 맞바꾼다.

data[2]와 data[3]~data[4]를 비교해서 data[2]가 가장 작으므로 그대로 둔다.

data[3]과 data[4]를 비교해서 data[3]을 더 작은 data[4]와 바꾼다.


Q&A