

2010-1학기 프로그래밍입문(1)

chapter 06-6 참고자료

2차원 배열과 포인터배열

박종혁

Tel: 970-6702

Email: jhpark1@snut.ac.kr

출처: 뇌를 자극하는 C프로그래밍, 한빛미디어

□ 2차원 배열의 선언과 초기화

- 2차원 배열은 1차원 배열을 배열요소로 갖는 새로운 배열이다.
 - 3명의 학생에 대한 4과목의 학생 점수를 처리하는 예

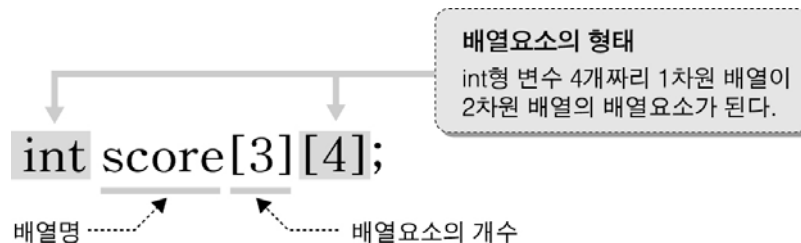
```
int score1[4];  
int score2[4];  
int score3[4];
```

} 같은 형태의 1차원 배열이 3개 필요하다.

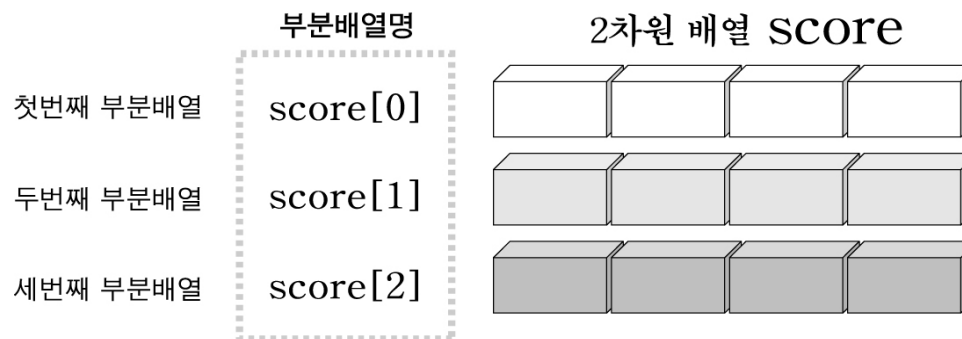


▶ 2차원 배열의 선언과 구조

- 2차원 배열은 1차원 배열처럼 배열명과 첨자를 사용하여 선언한다.
 - 각 배열요소의 형태는 int형 변수 4개짜리 1차원 배열이다.
 - 전체 배열은 12개의 int형 기억공간이 1차원 배열의 형태로 할당된다.

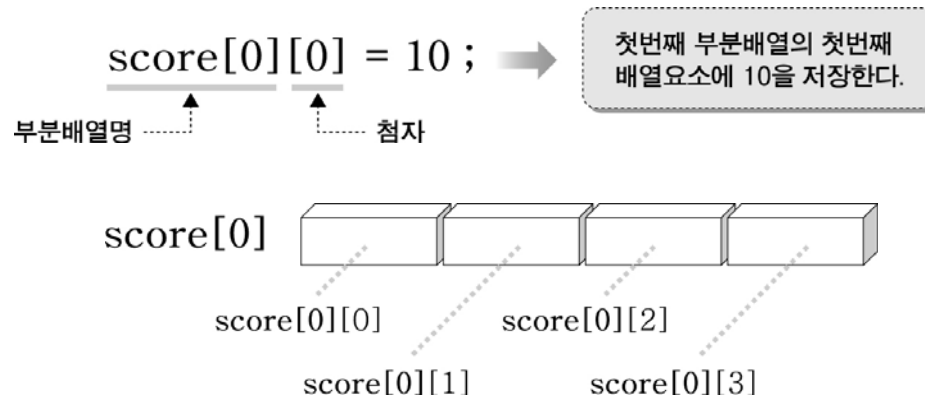


- 2차원 배열은 논리적으로 행렬로 표현된다.
 - 배열요소는 1차원 배열이므로 **부분배열**이라고 한다.



▶ 배열요소의 참조

- 2차원 배열은 부분배열의 배열요소를 다시 참조해야 한다.



- 첫 번째 부분배열에 4과목의 점수를 입력하는 예

```

int score[3][4]; // 2차원 배열 선언
int j;           // 반복 제어변수
printf("네 과목의 점수를 입력하세요 : ");
for(j=0; j<4; j++){ // j는 0부터 3까지 4번 반복
    scanf("%d", &score[0][j]);
} // 첫 번째 부분배열의 배열요소에 각각 점수 입력
  // 결국 한 학생의 모든 점수를 입력 받는다.
    
```

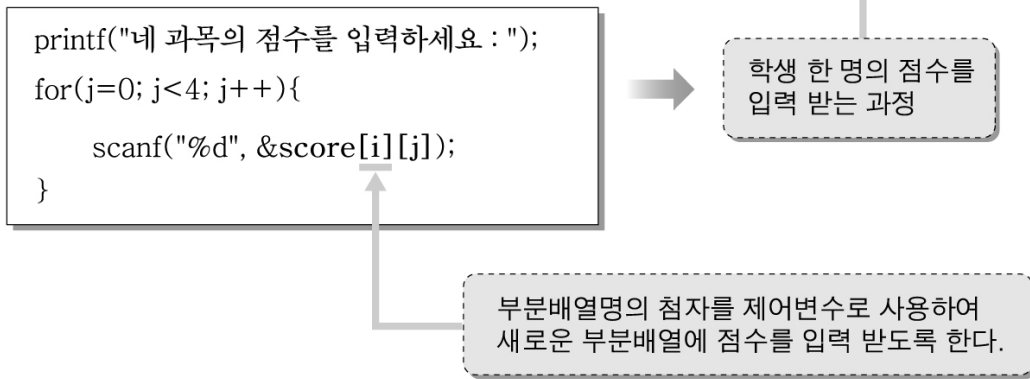


▶ 2중 for문으로 모든 배열요소를 처리한다.

- 나머지 두 명의 점수를 입력 받기 위해 점수를 입력 받는 과정을 반복한다.

```
int score[3][4];
int i, j;
```

```
for(i=0; i<3; i++){ ← i는 0부터 2까지 세 번 반복된다.
```



j 반복문 →

i 반복문 ↓

	j가 0일 때	j가 1일 때	j가 2일 때	j가 3일 때
i가 0일 때	score[0][0]	score[0][1]	score[0][2]	score[0][3]
i가 1일 때	score[1][0]	score[1][1]	score[1][2]	score[1][3]
i가 2일 때	score[2][0]	score[2][1]	score[2][2]	score[2][3]

▶ 2차원 배열을 사용하여 학생들의 점수를 처리하는 프로그램

```
#include <stdio.h>

int main()
{
    int score[3][4];           // 3명의 4과목 점수를 저장할 2차원 배열 선언
    int i, j;                 // 2중 for문을 위한 반복 제어변수
    int tot;                  // 총점을 저장할 변수
    double avg;              // 평균을 저장할 변수

    for(i=0; i<3; i++){      // 3명이므로 3번 반복
        printf("네 과목의 점수를 입력하세요 : ");
        for(j=0; j<4; j++){  // 4과목이므로 4번 반복
            scanf("%d", &score[i][j]); // 점수 입력
        }
    }

    for(i=0; i<3; i++){
        tot=0;                // 각 학생의 점수를 새롭게 누적할 때마다 0으로 초기화
        for(j=0; j<4; j++){
            tot+=score[i][j]; // 한 학생의 점수를 총점에 누적한다.
        }
        avg=tot/4.0;          // 한 명의 총점을 모두 누적한 후에 평균 계산
        printf("총점 : %d, 평균 : %.2lf\n", tot, avg); // 총점, 평균 출력
    }

    return 0;
}
```

▶ 2차원 배열의 초기화

- 기본적으로 1차원 배열을 초기화하는 방식과 같다.
 - 초기화 값은 행 단위로 차례로 저장된다.

```
int nums[3][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

1	2	3	4
5	6	7	8
9	10	11	12



첫번째 행이 먼저 초기화된
후에 두 번째, 세 번째 행이
차례로 초기화 된다.

- 기억공간의 수보다 초기화 값이 적으면 남은 기억공간은 0으로 초기화 된다.

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

1	2	3	4
5	6	0	0
0	0	0	0

▶ 2차원 배열의 초기화

- 초기화 값을 논리적 구조에 맞게 행으로 구분하고자 하면 중괄호를 한번 더 사용한다.

```
int nums[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
```

첫번째 행
두 번째 행
세 번째 행

- 행 초기화 괄호를 사용하면 행 단위로 초기값을 생략할 수 있다.

```
int nums[3][4] = { {1}, {5, 6}, {9, 10, 11} };
```

1	0	0	0
5	6	0	0
9	10	11	0

- 1차원 배열과 마찬가지로 배열의 첨자를 생략할 수 있다.

행 초기화 괄호를 사용한 경우

행 첨자 생략 열 첨자는 생략 불가능

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

→ 행의 수가 세 개

1	0	0	0
2	3	0	0
4	5	6	0

} 3행 4열의 기억공간 할당

행 초기화 괄호를 사용하지 않은 경우

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

첫번째 행 초기값 두 번째 행 초기값

1	2	3	4
5	6	0	0

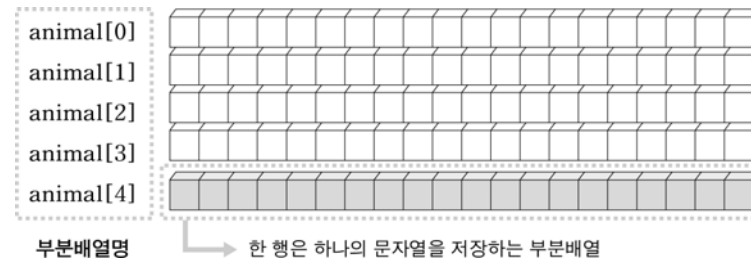
} 2행 4열의 기억공간 할당

□ 2차원 문자배열

- 여러 개의 문자열을 저장하기 위해서는 2차원 문자배열이 필요하다.
 - 5개의 문자열을 저장할 2차원 문자배열

```
char animal[5][20];
```

100개의 char형
기억공간 할당



- 마지막 부분배열에 문자열을 입력할 때(부분배열명을 전달인자로 준다).

```
scanf("%s", animal[4]);    // gets함수를 사용하면 gets(animal[4]);
```

 ↖ 전체가 배열명이다!

- 반복문을 사용하여 모든 부분배열에 문자열을 입력하자.

```
for(i=0; i<5; i++){  
    printf("문자열을 입력하세요 : ");  
    scanf("%s", animal[i]);    // i의 값이 변하면서 각 부분배열명이 된다.  
}
```

▶ 2차원 문자배열의 초기화

- 기본적으로 각 기억공간들을 문자상수로 초기화할 수 있다.

```
char animal[5][10]={ {'c', 'a', 't', '\0'}, {'h', 'o', 'r', 's', 'e', '\0'},  
                    {'d', 'o', 'g', '\0'}, {'t', 'i', 'g', 'e', 'r', '\0'},  
                    {'e', 'l', 'e', 'p', 'h', 'a', 'n', 't', '\0'} };
```

- 각 행은 문자열상수로 직접 초기화할 수 있다.

`char str[10] = "cat";` → 하나의 문자열 상수는 1차원 문자배열을 초기화할 수 있다.

```
char animal[5][10] = { "cat", "horse", "dog", "tiger", "elephant" };
```



c	a	t	\0	\0	\0	\0	\0	\0	\0
h	o	r	s	e	\0	\0	\0	\0	\0
d	o	g	\0	\0	\0	\0	\0	\0	\0
t	i	g	e	r	\0	\0	\0	\0	\0
e	i	e	p	h	a	n	t	\0	\0

→ 남은 기억공간들은
널문자로 채워진다.

▶ 2차원 문자배열을 초기화하고 출력하는 프로그램


```
#include <stdio.h>

int main()
{
    char animal[][10]={ "monkey", "elephant", "dog", "sheep", "pig",
                        "lion", "tiger", "puma", "turtle", "fox" };
    // 2차원 문자배열의 선언과 초기화

    int i;                // 반복 제어변수
    int count;            // 문자열의 개수를 저장할 변수

    count=sizeof(animal)/sizeof(animal[0]); // 초기화된 문자열의 수를 계산한다.
    for(i=0; i<count; i++){ // 문자열의 개수만큼 반복
        printf("%s\n", animal[i]); // 저장된 문자열의 출력
    }

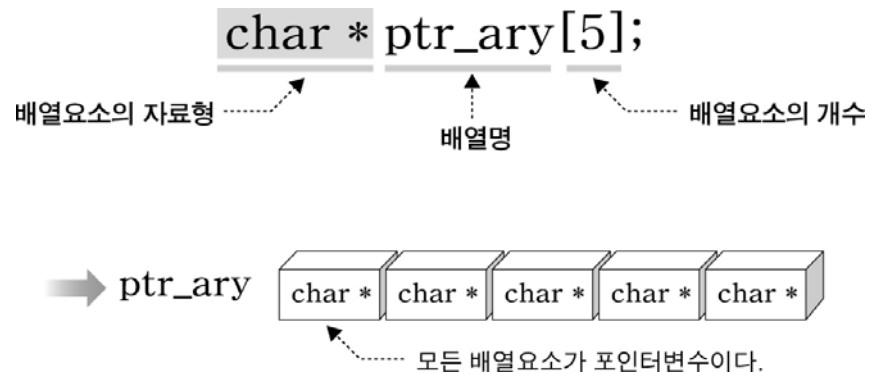
    return 0;
}
```

문자열의 개수 계산  $\frac{\text{sizeof(animal)}}{\text{sizeof(animal[0])}}$

전체 배열의 크기 / 부분배열 하나의 크기

□ 포인터배열

- 포인터배열은 포인터변수들을 배열요소로 갖는 배열이다.



- 세 번째 배열요소에 문자열상수를 대입하고 출력할 때

```
ptr_ary[2]="tiger"; // 문자열상수는 포인터이므로 포인터만을 저장하는 것이다.  
printf("%s", ptr_ary[2]);
```

▶ 포인터배열을 사용하여 여러 개의 문자열을 출력하는 프로그램

```
#include <stdio.h>
```

```
int main()  
{
```

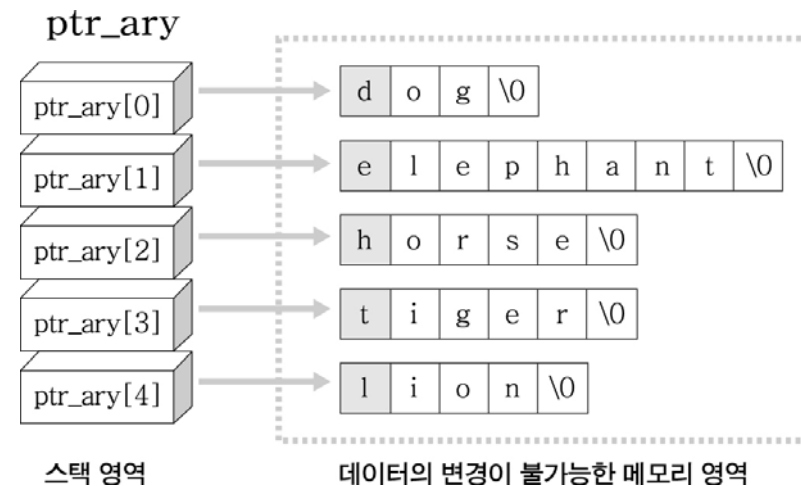
```
    char *ptr_ary[5];  
    int i;
```

```
    ptr_ary[0]="dog";  
    ptr_ary[1]="elephant";  
    ptr_ary[2]="horse";  
    ptr_ary[3]="tiger";  
    ptr_ary[4]="lion";
```

```
    for(i=0; i<5; i++){  
        printf("%s\n", ptr_ary[i]);  
    } // 배열요소를 하나씩 참조하여 모든 문자열을 출력한다.
```

```
    return 0;
```

```
}
```



▶ 포인터배열의 초기화

- 문자열상수는 포인터이므로 포인터배열에 초기값으로 사용한다.

```
char *ptr_ary[5]={“dog”, “elephant”, “horse”, “tiger”, “lion”};
```

- 2차원 문자배열에 초기화하는 것은 모든 문자열이 배열에 복사되는 것이고 포인터배열에 초기화하는 것은 포인터만 초기화 되는 것이다.

```
char animal[5][10] = { "dog", "elephant", "horse", "tiger", "lion" };  
char *ptr_ary[5]   = { "dog", "elephant", "horse", "tiger", "lion" };
```

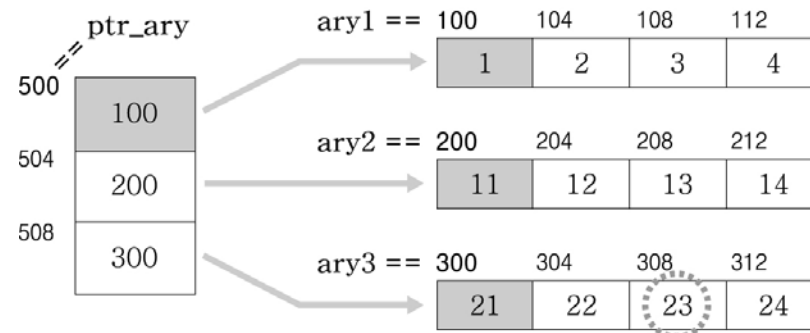
배열의 형태에 따라 문자열을 복사하거나 포인터를 저장한다.

초기화 방법은 같다.

▶ 포인터배열은 2차원 배열처럼 활용된다.

- 1차원 배열의 배열명을 포인터배열에 저장하면 포인터배열을 2차원 배열처럼 사용할 수 있다.

```
int ary1[4]={1, 2, 3, 4};  
int ary2[4]={11, 12, 13, 14};  
int ary3[4]={21, 22, 23, 24};  
int *ptr_ary[3]={ary1, ary2, ary3};  
// 각 배열명을 포인터배열에 초기화한다.
```



(각 기억공간의 주소값은 설명의 편의를 위해 임의로 붙인 것입니다.)

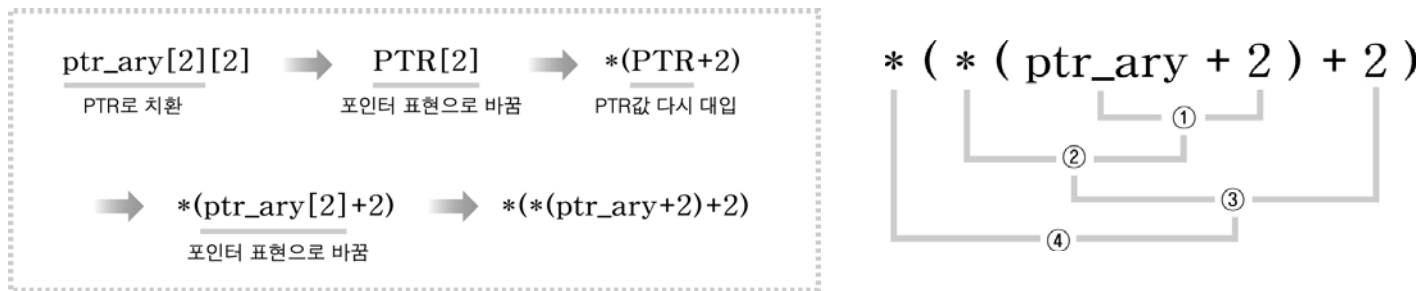
- ary3배열의 세 번째 배열요소(23값)를 참조하는 과정

1. 먼저 `ptr_ary` 배열의 세 번째 배열요소를 참조한다. ⇒ `ptr_ary[2]`
2. 참조된 배열요소 `ptr_ary[2]`는 배열명 `ary3`를 저장한 포인터변수이므로 배열명처럼 사용하여 `ary3`의 세 번째 배열요소를 참조한다.

⇒ `printf("%d", ptr_ary[2][2]);`

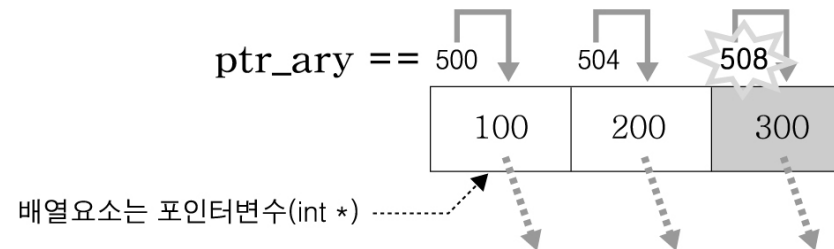
▶ 포인터배열은 2차원 배열처럼 활용된다.

- ptr_ary[2][2]가 참조되는 과정의 주소값을 계산해보자.
 - 일단 포인터표현으로 바꾸고 연산순서를 따라간다.



- ①번 연산 : 포인터배열의 세 번째 배열요소를 가리키는 포인터가 구해진다.

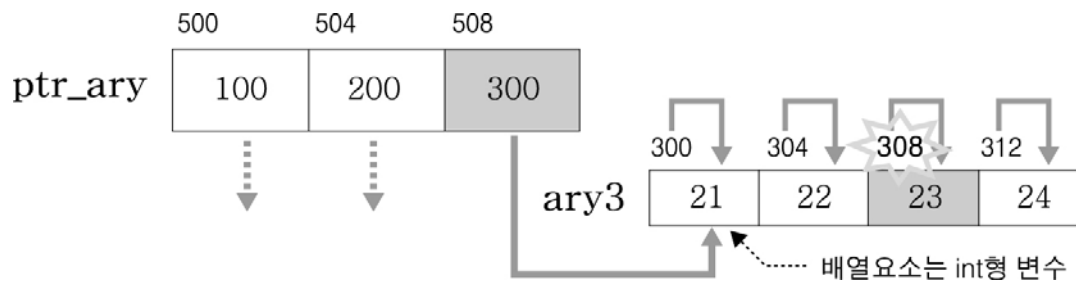
$$ptr_ary+2 = ptr_ary+(2*\text{sizeof}(ptr_ary[0])) = 500+(2*4) = 508$$



▶ 포인터배열은 2차원 배열처럼 활용된다.

- ②번 연산 : 포인터배열의 세 번째 배열요소의 값 300번지가 구해진다.
- ③번 연산 : ary3배열의 세 번째 기억공간을 가리키는 포인터가 구해진다.

$$300+2 \Rightarrow 300+(2*\text{sizeof}(\text{ary3}[0])) \Rightarrow 308$$



- ④번 연산 : 308번지는 포인터이므로 참조연산을 수행하면 값23이 참조된다.

▶ 반복문으로 모든 배열요소의 값을 출력하자.

```
#include <stdio.h>

int main()
{
    int ary1[4]={1, 2, 3, 4};
    int ary2[4]={11, 12, 13, 14};
    int ary3[4]={21, 22, 23, 24};
    int *ptr_ary[3]={ary1, ary2, ary3}; // 포인터배열에 각 배열명을 초기화한다.
    int i, j; // 반복 제어변수

    for(i=0; i<3; i++){
        for(j=0; j<4; j++){
            printf("%5d", ptr_ary[i][j]); // 3행 4열의 2차원 배열처럼 출력할 수 있다.
        }
        printf("\n"); // 한 행을 출력한 후에 줄을 바꾼다.
    }

    return 0;
}
```

출력 결과

```
1  2  3  4
11 12 13 14
21 22 23 24
```