7. 배열

♲ 배열이란?

배열이란 같은 자료형을 가지는 변수를 메모리에 빈틈없이 연달아 만들 수 있게 해주는 C언어의 도구이다. 단 이때 만들어지는 변수들을 각각의 이름을 가지고 있는 것이 아닌 메모리만 할당받게 된다. 따라서 배열은 하나의 대표이름(배열의 이름은 포인터 상수이다.)으로 여러 개의 자료를 관리하기 위한 데이터 저장방법이라 할 수 있다.

♲ 배열의 장 · 단점

배열의 장점

1. 배열은 메모리상에 연달아 빈틈없이 만들어 짐으로 변수를 따로 만드는 것보다 메모리를 정략할 수 있다.
2. 배열은 하나의 대표이름으로 변수 여러 개에 해당하는 기억당소를 한 번에 할당 받을 수 있으며 일괄적인 처리가 가능하게 해준다.
3. 만들기가 쉽고 원소 호출 시 포인터 연산을 통해 빠른 속도로 원소를 꺼내 올 수 있다.

배열의 단점

1. 한 번 만들어진 배열은 크기를 변경 할 수 없다. (동적 메모리 할당 제외)
2. 원소들이 빈틈없이 연달아 만들어져 있기 때문에 원소들 사이에 다른 원소를 삽입하거나 삭제하기가 번거롭다.
3. 배열은 포인터의 한 종류임으로 모든 연산이 포인터 연산으로 변경되어 동작한다. 따라서 배열을 call by value 형태로 함수의 매개변수로 건네 줄 수 없다. (배열을 매개변수로 주면 call by pointer가 된다.)

♲ 배열을 만드는 방법과 사용하는 방법

배열을 만들기 위해서는 일반 변수와 동일한 방법으로 만들되 만드는 변수명 뒤쪽에 [배열의크기]를 붙인다. 즉, “①자료형 ②배열명[③배열의크기]” 와 같은 형태로 만들면 배열이 만들어 진다.
배열을 위와 같은 방식대로 선언하면 “①자료형”과 같은 자료형을 가지는 변수가 “③배열의크기”만큼 보관 될 수 있도록 메모리가 할당되어 그 시작 주소가 “②배열명”에 보관된다.

예) int arr[4]; => 자료형이 int인 변수가 4개 보관될 수 있도록 16byte의 메모리가 할당되어 할당된 공간이 시작하는 곳의 주소가 arr이라는 포인터 상수에 보관 된다.

배열을 만들 때 “[배열의 크기]”를 한 개 사용해 배열을 만들면 1차원 배열이라 하고, 두 개 사용하면 2차원 배열, 세 개 사용하면 3차원 배열이라 한다. 배열은 만들 때 반드시 크기가 정해져 있어야 한다. (프로그램의 실행 중에 크기를 결정 할 수는 없다.)

만들어진 배열을 사용할 때는 배열의 이름과 인덱스 번호를 같이 사용해 원소를 호출한다. 인덱스 번호란 배열안에 있는 원소의 위치를 말하며 배열의 크기가 n이라 할 경우 인덱스 번호는 0 ~ n-1번까지 존재하게 된다.

예) arr[2]; => 배열안에 있는 원소 중 인덱스 번호가 2번인 자료가 나오게 된다.

배열을 전역에 선언하면 전역배열, 배열은 만들면서 앞쪽에 static을 붙이면 정적 배열이 된다.
즉, 배열도 일반 변수와 같이 여러 형태로 만들 수 있다. (전역, 정적, 외부, 오토 등 등)

♲ 1차원 배열

1차원 배열은 하나의 인덱스로 만든 배열을 말한다.

1차원 배열을 만드는 방법

자료형 배열명[배열의크기];

예) int arr[4];

그림 1) 자료형이 int인 4칸 짜리 배열이 메모리에 만들어지는 물리적 형태

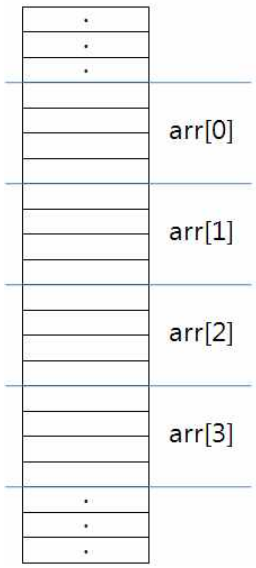


그림 2) 배열의 표현하는 논리적 형태

1차원 배열안에 있는 원소에 값을 집어 넣거나 배열에 있는 원소의 값을 불러오는 방법

배열명[인덱스번호];

예) arr[0], arr[1], arr[2], arr[3] (배열은 인덱스는 0 ~ 배열의크기-1 번까지 부여된다.)

1차원 배열을 만들면서 초기값을 부여하는 방법과 그 예

1. int arr[3] = { 10,20,30 }; char arr[3] = { 'A' , 'B', 'C' };
   => 배열에 넣은 값 들이 순서대로 들어간다.
2. int arr[3] = { 0 }; char arr[3] = {NULL};
   => 배열은 일부의 원소만 초기화 시키면 초기화가 안 된 나머지 원소는 자동으로 0 또는 NULL 값으로 지동 초기화 된다. 따라서 위의 방법을 사용하면 모든 원소가 0(문자나 포인터는 NULL)으로 초기화 된다.
3. int arr[] = { 10,20,30 }; char arr[] = { 'A' , 'B', 'C' };
   => 초기 값을 넣어주면 배열의 크기를 생략 할 수 있다. 이때 배열은 넣어준 초기 값을 보관할 수 있는 크기로 만들어 진다.

1차원 배열 전체의 크기는 “원소 하나의 크기 * 원소의 개수”로 계산 할 수 있다. 또한 배열 전체의 크기를 구하려면 sizeof 연산자에 배열의 이름을 넣어주면 된다.

```
예 ) int arr[3];
     printf("sizeof(arr) = %dWn", sizeof(arr) );   // 배열 arr의 총 크기인 12가 출력된다.
```

배열은 자신의 인덱스를 변수나 수식으로 대체해 호출 할 수 있다. 단, 이 경우 인덱스에 들어가는 값은 결과가 반드시 정수이어야 한다.

```
예 ) int arr[3];
     int a = 1;
     arr[a] = 10;
     printf("arr[1] = %dWn", arr[a] );   // a가 1임으로 배열의 [1]번째 원소가 출력된다.
```

♲ 1차원 배열의 사용 예제

예제 7-1 : 일차원 배열을 만들고 사용하는 방법

```
#include <stdio.h>
void main ( ) {
    short arr[3];
    arr[0] = 10;
    arr[1] = 20;
    arr[2] = 30;
    printf("arr[0] = %dWn", arr[0]);
    printf("arr[1] = %dWn", arr[1]);
    printf("arr[2] = %dWn", arr[2]);
}
```

예제 7-2 : 일차원 배열을 만들고 사용하는 방법 - 2

```
#include <stdio.h>
void main ( ) {
    double arr[3];
    arr[0] = 12.3;
    arr[1] = 23.4;
    arr[2] = 34.5;
    printf("arr[0] = %lfWn", arr[0]);
    printf("arr[1] = %lfWn", arr[1]);
    printf("arr[2] = %lfWn", arr[2]);

}
```

예제 7-3 : 지역변수로 선언한 배열은 최초 선언 시 쓰레기 값이 들어 있는 것을 확인하는 예제

```
#include <stdio.h>
void main ( ) {
    int arr[3];
    printf("arr[0]=%d, arr[1]=%d, arr[2]=%d₩n", arr[0], arr[1], arr[2]);
}
```

예제 7-4 : 배열을 선언함과 동시에 초기값을 부여하는 방법 - 1

```
#include <stdio.h>
void main ( ) {
    int arr[3] = { 10,20,30 };
    printf("arr[0]=%d, arr[1]=%d, arr[2]=%d₩n", arr[0], arr[1], arr[2]);
            // 순서대로 10,20,30이 출력된다.
}
```

예제 7-5 : 배열을 선언함과 동시에 초기값을 부여하는 방법 - 2

```
#include <stdio.h>
void main ( ) {
    int arr[3] = { 10 };
    printf("arr[0]=%d, arr[1]=%d, arr[2]=%d₩n", arr[0], arr[1], arr[2]);
            // 배열의 일부 원소만 초기화를 시키면 초기화가 안된 나머지 값은 자동으로
}           // 0값을 가지게 된다.
```

예제 7-6 : 배열을 선언함과 동시에 초기값을 부여하는 방법 - 3

```
#include <stdio.h>
void main ( ) {
    int arr[3] = { 0 };
    printf("arr[0]=%d, arr[1]=%d, arr[2]=%d₩n", arr[0], arr[1], arr[2]);
            // 배열의 일부 원소만 초기화를 시키면 초기화가 안된 나머지 값은 자동으로
}           // 0값을 가지게 된다. 따라서 위의 방법을 사용하면 모든 원소가 0이 된다.
```

예제 7-7 : 배열을 선언함과 동시에 초기값을 부여하는 방법 - 4

```
#include <stdio.h>
void main ( ) {
    int arr[] = { 10, 20 };
    printf("arr[0]=%d, arr[1]=%d₩n", arr[0], arr[1]);
    // 배열은 반드시 선언 시 크기가 명시되어 있어야 하지만 초기값을 부여한다면 크기를
}   // 생략 할 수 있다. 이 경우 초기값을 보관할 만큼의 크기로 배열이 선언된다.
```

예제 7-8 : 배열 선언 시 크기가 명시되어 있지 않아 오류가 발생하는 예제

```
#include <stdio.h>
void main ( ) {
    int arr[];          // 배열의 크기가 명시되어 있지 않음으로 에러
    int x;
    int brr[x];         // 쓰레기 값을 가지는 변수의 크기로 배열을 만들었음으로 에러
    scanf("%d", &x);
    int crr[x];         // 입력 받은 값을 바탕으로 배열을 만들려고 했지만 입력은
                        // 프로그램 실행 시 동작함으로 컴파일 중 배열을 크기를 알 수
}                       // 에러가 발생한다.
```

예제 7-9 : 배열의 크기를 출력하는 예제

```
#include <stdio.h>
void main ( ) {
    int arr[3] = { 10,20,30 };
    printf("sizeof(arr) = %d₩n", sizeof(arr) );       // 배열 전체의 크기가 출력된다.
    printf("sizeof(arr[0]) = %d₩n", sizeof(arr[0]) ); // 원소 하나의 크기가 출력된다.
    printf("sizeof(arr)/sizeof(arr[0]) = %d₩n", sizeof(arr)/sizeof(arr[0] );
}           // 배열전체 크기를 원소하나의 크기로 나누면 원소의 갯수를 구할 수 있다.
```

예제 7-10 : 배열에 있는 원소를 호출 할 경우 인덱스를 변수나 수식으로 대체해 사용한 예제

```
#include <stdio.h>
void main ( ) {
    int arr[4];
    int x = 1;
    arr[0] = 3;
    arr[x] = 5;         // x가 1임으로 배열의 [1]번 원소가 나온다.
    arr[x+1] = 7;       // x+1이 2임으로 배열의 [2]번 원소가 나온다.
    arr[arr[0]] = 9;    // arr[0]이 3임으로 배열의 [3]번 원소가 나온다.
    printf("arr[0]=%d, arr[1]=%d, arr[2]=%d, arr[3]=%d₩n",arr[0],arr[1],arr[2],arr[3]);
                        // 각 각 순서대로 3, 5, 7, 9 가 출력된다.
}
```

예제 7-11 : 호출할 원소의 번호를 입력받아 사용하는 예제

```
#include <stdio.h>
void main ( ) {
    int arr[4] = { 10,20,30,40 };
    int x;
    printf("몇 번째 원소를 호출 할지 입력해 주세요 (0~3) : ");
    scanf("%d", &x);
    printf("배열의 %d번째 원소는 %d입니다.₩n", x, arr[x]);
}
```

예제 7-12 : 반복문을 이용해 배열안에 있는 원소를 모두 출력하는 예제

```
#include <stdio.h>
void main ( ) {
    int arr[10] = { 10,20,30,40,50,60,70,80,90,100 };
    int i;
    for(i=0; i<=9; I++) {          // i가 0부터 배열의 마지막 인덱스까지 변하도록 반복
        printf("arr[%d] = %d₩n", i, arr[i] );
    }
}
```

예제 7-13 : 배열에 있는 원소들의 총합과 평균을 구하는 예제

```
#include <stdio.h>
void main ( ) {
    int arr[10] = { 51, 45, 64, 35, 74, 25, 81, 22, 89, 15 };
    int i, sum = 0;        // sum은 총점을 저장하기 위한 변수
    for(i=0; i<=9; i++ ) {
        sum = sum + i;
    }
    printf("배열안에 있는 원소의 총합 : %d₩n", sum);
    printf("배열안에 있는 원소의 평균 : %.2lf₩n", sum/10.0);  // 자동 형 변환
}
```

예제 7-14 : 배열에 있는 원소들 중 최대값과 최소값을 구하는 예제

```
#include <stdio.h>
void main ( ) {
    int arr[10] = { 51, 45, 64, 35, 74, 25, 81, 22, 89, 15 };
    int i, max, min;          // max는 최대값, min은 최소값을 저장하기 위한 변수
    max = min = arr[0];       // max와 min을 배열의 0번째 원소 값으로 초기화
    for(i=1; i<=9; i++ ) {
        if(arr[i] > max) {    // 꺼내온 원소가 max보다 더 큰 값이라면
            max = arr[i];     // 그 값을 max에 넣는다.
        }
        if(arr[i] < min) {    // 꺼내온 원소가 min보다 더 작은 값이라면
            min = arr[i];     // 그 값을 min에 넣는다.
        }
    }
    printf("배열안에 있는 원소 중 최대값 : %d₩n", max);
    printf("배열안에 있는 원소 중 최소값 : %d₩n", min);
}
```

예제 7-15 : 배열에 있는 원소들 중 50~80점 사이에 있는 점수의 갯수를 구하는 예제

```
#include <stdio.h>
void main ( ) {
    int arr[10] = { 51, 45, 64, 35, 74, 25, 81, 22, 89, 15 };
    int i, count = 0;        // count는 개수를 저장하기 위한 변수
    for(i=0; i<=9; i++ ) {
        if(arr[i] >=50 && arr[i] <=80 ) {
            count++;
        }
    }
    printf("배열안에 있는 원소 중 50~80사이에 있는 원소의 갯수 : %d₩n", count);
}
```

♲ 2차원 배열

2차원 배열은 두 개의 인덱스로 만든 배열을 말한다. 첫 번째 인덱스를 행 번호라 하고 두 번째 인덱스를 열 번호라 한다. 행은 몇 번째 줄인지를 뜻하고 열은 몇 번째 칸인지를 뜻한다. 이차원 배열은 표시할 때 평면으로 표시하지만 실제 배열이 저장되는 메모리는 일차원이기 때문에 이차원 배열도 메모리에는 한줄로 저장된다. (한 번에 한 행씩 저장 된다. 이것을 행우선사상방식 이라 한다.)

2차원 배열을 만드는 방법

자료형 배열명[배열의 행크기][배열의 열크기];

예) short arr[2][3] = { 10,20,30,40,50,60 };

그림 1) 자료형이 short인 2행3열 짜리 배열이 메모리에 만들어지는 물리적 형태

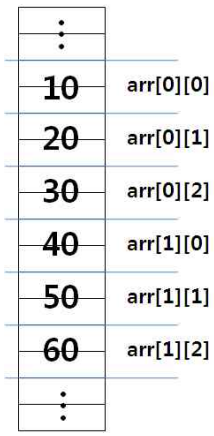
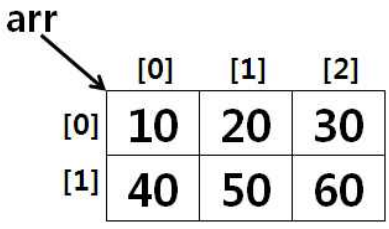


그림 2) 2차원 배열을 표현하는 논리적 형태



2차원 배열안에 있는 원소에 값을 집어 넣거나 배열에 있는 원소의 값을 불러오는 방법

배열명[행번호][열번호];

예) arr[0][0], arr[0][1], arr[1][0], arr[1][1]

배열의 길이를 n행 m열이라 할 때 호출 할 수 있는 원소의 행 번호는 0~n-1번 이고 열 번호는 0~m-1번까지 이다.

예) double arr[3][4]; 과 같은 형태로 배열을 만들었다면 호출할 수 있는 원소들은 아래와 같다.

```
arr[0][0], arr[0][1], arr[0][2], arr[0][3]      // 열 번호는 0~3까지 사용 할 수 있다.
arr[1][0], arr[1][1], arr[1][2], arr[1][3]
arr[2][0], arr[2][1], arr[2][2], arr[2][3]      // 행 번호는 0~2까지 사용 할 수 있다.
```

2차원 배열을 만들면서 초기값을 부여하는 방법과 그 예

1. int arr[2][3] = { {10,20,30}, {40,50,60} };
   => 배열에 넣은 값 들이 순서대로 들어간다. { }로 하나가 한 개의 행을 의미하게 된다.
2. int arr[2][3] = { 10,20,30,40,50,60 };
   => 배열에 넣은 값 들이 순서대로 들어간다. 어차피 이차원 배열도 메모리에는 한 줄로 보관됨으로 메모리에 만들어지는 원소들 순서대로 초기화 시킨 값이 하나씩 들어간다.
3. int arr[2][3] = { 0 }; char arr[2][3] = {NULL};
   => 일차원 배열과 동일하게 일부의 원소만 초기화 시키면 초기화가 안 된 나머지 원소는 자동으로 0 또는 NULL값으로 지동 초기화 된다. 따라서 위의 방법을 사용하면 모든 원소가 0(문자나 포인터는 NULL)으로 초기화 된다.
4. int arr[][3] = { {10,20,30}, {40,50,60} };
   => 이차원 배열 선언 시 초기 값을 넣어주면 배열의 행 크기를 생략 할 수 있다. (열 크기는 어떠한 경우에도 생략이 불가능하다. 이차원 배열 은 원소 하나가 일차원 배열인 배열이기 때문이다. 자세한 설명은 다음 챕터에서 한다.) 이때 배열은 넣어준 초기 값을 보관할 수 있을 만큼의 행을 가지는 배열로 만들어 진다. (초기값이 6개임으로 2행 3열 짜리 이차원 배열로 만들어진다.)

```
int arr[][3] = { 10,20,30,40 };  // 4개의 자료를 보관하기 위해 2행이 필요함으로 2행3열
                                    짜리 배열로 만들어 진다.
int arr[][3] = { {10},{20},30,40 };  // 0행에는 10만 1행에는 20만 초기 값으로 들어가고
                                    30과 40은 2행의 초기 값으로 들어가게 된다. 따
                                    라서 배열을 3행 3열로 만들어 진다.
```

2차원 배열 전체의 크기는 “원소 하나의 크기 * 한 행을 구성하는 원소의 갯수 * 행수”로 계산할 수 있다. (한 행을 구성하는 원소의 개수는 열의 크기와 은 의미이다.)
즉, "한행의 크기 * 행수" == " (원소하나의크기 * 열의크기) * 행수" 이다.
배열의 크기는 sizeof 연산자를 이용해 배열의 이름으로 측정할 수 있고
한 행의 크기는 sizeof 연산자를 이용해 한 행의 이름(배열의 이름과 인덱스 하나)으로 측정할 수 있다.

예) int arr[2][3];

```
printf("sizeof(arr) = %dWn", sizeof(arr) );   // 배열 arr의 총 크기인 24가 출력된다.
printf("sizeof(arr[0]) = %dWn", sizeof(arr[0]) );   // [0]행의 크기인 12가 출력된다.
printf("sizeof(arr[1]) = %dWn", sizeof(arr[1]) );   // [1]행의 크기인 12가 출력된다.
printf("sizeof(arr[0][0]) = %dWn", sizeof(arr[0][0]) );   // 원소 하나의 크기인 4가
                                                          //  출력된다.
```

이차원 배열 역시 자신의 인덱스를 변수나 수식으로 대체해 호출 할 수 있다.

예) int arr[2][3] = { 10,20,30,40,50,60 };

```
int a = 1, b = 2;
printf("arr[%d][%d]=%d₩n", a, b, arr[a][b] ); // a가 1이고 b가 2임으로 arr[1][2]번째
                                              // 원소에 들어 있는 60이 출력된다.
```

♲ 2차원 배열의 사용 예제

예제 7-16 : 이차원 배열을 만들고 사용하는 방법

```
#include <stdio.h>
void main ( ) {
    short arr[2][3];
    arr[0][0] = 10;
    arr[0][1] = 20;
    arr[0][2] = 30;
    arr[1][0] = 40;
    arr[1][1] = 50;
    arr[1][2] = 60;
    printf("arr[0][0]=%d, arr[0][1]=%d, arr[0][2]=%d₩n", arr[0][0], arr[0][1], arr[0][2]);
    printf("arr[1][0]=%d, arr[1][1]=%d, arr[1][2]=%d₩n", arr[1][0], arr[1][1], arr[1][2]);
}
```

예제 7-17 : 이차원 배열을 만들면서 초기값을 부여 하는 방법

```
#include <stdio.h>
void main ( ) {
    short arr[2][3] = {10,20,30,40,50,60};
    short brr[2][3] = { {10,20,30}, {40,50,60} };
    short crr[2][3] = { 10,20,30, {40,50,60} };
    short drr[2][3] = { {10,20,30}, 40,50,60 };
    short err[2][3] = { 0 };
    short frr[][3] = { {10}, {40}, 50, 60 };
    short grr[][3] = { {10}, 20,30, {40} };
    printf("arr[0][0]=%d, arr[0][1]=%d, arr[0][2]=%d₩n", arr[0][0], arr[0][1], arr[0][2]);
    printf("arr[1][0]=%d, arr[1][1]=%d, arr[1][2]=%d₩n", arr[1][0], arr[1][1], arr[1][2]);
    printf("brr[0][0]=%d, brr[0][1]=%d, brr[0][2]=%d₩n", brr[0][0], brr[0][1], brr[0][2]);
    printf("brr[1][0]=%d, brr[1][1]=%d, brr[1][2]=%d₩n", brr[1][0], brr[1][1], brr[1][2]);
    printf("crr[0][0]=%d, crr[0][1]=%d, crr[0][2]=%d₩n", crr[0][0], crr[0][1], crr[0][2]);
    printf("crr[1][0]=%d, crr[1][1]=%d, crr[1][2]=%d₩n", crr[1][0], crr[1][1], crr[1][2]);
    printf("drr[0][0]=%d, drr[0][1]=%d, drr[0][2]=%d₩n", drr[0][0], drr[0][1], drr[0][2]);
    printf("drr[1][0]=%d, drr[1][1]=%d, drr[1][2]=%d₩n", drr[1][0], drr[1][1], drr[1][2]);
    printf("err[0][0]=%d, err[0][1]=%d, err[0][2]=%d₩n", err[0][0], err[0][1], err[0][2]);
    printf("err[1][0]=%d, err[1][1]=%d, err[1][2]=%d₩n", err[1][0], err[1][1], err[1][2]);
    printf("frr[0][0]=%d, frr[0][1]=%d, frr[0][2]=%d₩n", frr[0][0], frr[0][1], frr[0][2]);
    printf("frr[1][0]=%d, frr[1][1]=%d, frr[1][2]=%d₩n", frr[1][0], frr[1][1], frr[1][2]);
    printf("grr[0][0]=%d, grr[0][1]=%d, grr[0][2]=%d₩n", grr[0][0], grr[0][1], grr[0][2]);
    printf("grr[1][0]=%d, grr[1][1]=%d, grr[1][2]=%d₩n", grr[1][0], grr[1][1], grr[1][2]);
}
```

예제 7-18 : 이차원 배열의 크기를 알아보는 예제

```
#include <stdio.h>
void main ( ) {
    int arr[2][4];
    char brr[2][4];
    double crr[2][4];
    printf("sizeof(arr) = %dWn", sizeof(arr) );
    printf("sizeof(arr[0]) = %dWn", sizeof(arr[0]) );
    printf("sizeof(arr[0][0]) = %dWn", sizeof(arr[0][0]) );
    printf("Wn");
    printf("sizeof(brr) = %dWn", sizeof(brr) );
    printf("sizeof(brr[0]) = %dWn", sizeof(brr[0]) );
    printf("sizeof(brr[0][0]) = %dWn", sizeof(brr[0][0]) );
    printf("Wn");
    printf("sizeof(crr) = %dWn", sizeof(crr) );
    printf("sizeof(crr[0]) = %dWn", sizeof(crr[0]) );
    printf("sizeof(crr[0][0]) = %dWn", sizeof(crr[0][0]) );
    printf("Wn");
}
```

예제 7-19 : 반복문을 이용해 이차원 배열에 있는 값을 출력하는 예제 (행순으로 출력)- 1

```
#include <stdio.h>
void main ( ) {
    int arr[2][4] = { 10,20,30,40,50,60,70,80 };
    int i, j;
    for( i=0; i<=1; i++) {
        for( j=0; j<=3; j++) {
            printf("arr[%d][%d] = %dWn", i, j, arr[i][j]);
        }
        printf("Wn");
    }
}
```

예제 7-20 : 반복문을 이용해 이차원 배열에 있는 값을 출력하는 예제 (열순으로 출력) - 2

```
#include <stdio.h>
void main ( ) {
    int arr[2][4] = { 10,20,30,40,50,60,70,80 };
    int i, j;
    for( i=0; i<=3; i++) {
        for( j=0; j<=1; j++) {
            printf("arr[%d][%d] = %dWn", j, i, arr[j][i]);
        }
        printf("Wn");
    }
}
```

예제 7-21 : 이차원 배열 안에 있는 원소들을 총합, 최대값, 최소값을 구하는 예제

```
#include <stdio.h>
void main ( ) {
    int arr[3][4] = { 54,45,59,65,37,75,82,14,81,94,51,34 };
    int i, j, sum=0, max, min;
    max = min = arr[0][0];
    for( i=0; i<=2; i++) {
        for( j=0; j<=3; j++) {
            sum = sum + arr[i][j];
            if(arr[i][j] > max) {
                max = arr[i][j];
            }
            if(arr[i][j] < min) {
                min = arr[i][j];
            }
        }
    }
    printf("총  합 : %d₩n", sum);
    printf("최대값 : %d₩n", max);
    printf("최소값 : %d₩n", min);
}
```

예제 7-22 : 이차원 배열 안에 입력한 값이 있는지 찾아주는 예제

```
#include <stdio.h>
void main ( ) {
    int arr[3][4] = { 54,45,59,65,37,75,82,14,81,94,51,34 };
    int i, j, in;
    printf("검색하고 싶은 값을 넣어주세요 : " );
    scanf("%d", &in);
    for( i=0; i<=2; i++) {
        for( j=0; j<=3; j++) {
            if( arr[i][j] == in) {
                printf("%d는 배열안에 존재합니다.₩n", in)
                return;
            }
        }
    }
    printf("%d는 배열안에 없습니다. ₩n", in);
}
```

예제 7-23 : 이차원 배열을 이용한 출력방법

```
#include <stdio.h>
int arr[5][5];
void print( ) {
    int i, j;
    for( i=0; i<=4; i++) {
        for( j=0; j<=4; j++) {
            printf("%2d", arr[i][j]);
        }
        printf("₩n");
    }
    printf("₩n");
}
void main ( ) {
    int i, j, count=0;
    for( i=0; i<=4; i++) {
        for( j=0; j<=4; j++) {
            arr[i][j] = ++count;
        }
    }
    print( );
    count = 0;
    for( i=4; i>=0; i--) {
        for( j=0; j<=4; j++) {
            arr[i][j] = ++count;
        }
    }
    print( );
    count = 0;
    for( i=0; i<=4; i++) {
        for( j=4; j>=0; j--) {
            arr[i][j] = ++count;
        }
    }
    print( );
    count = 0;
    for( i=4; i>=0; i--) {
        for( j=4; j>=0; j--) {
            arr[i][j] = ++count;
        }
    }
    print( );
    count = 0;
    for( i=0; i<=4; i++) {
        for( j=0; j<=i; j++) {
            arr[i][j] = 0;
        }
    }
    print( );
}
```

♲ 3차원 배열

3차원 배열은 세 개의 인덱스로 만든 배열을 말한다. 첫 번째 인덱스를 면 번호라 하고 두 번째 인덱스를 행 번호, 세 번째 인덱스를 열 번호라 한다. 면은 어떠한 이차원 배열인지를 의미하고 행은 몇 번째 줄인지, 열은 몇 번째 칸인지를 뜻한다. 삼차원 배열은 표시할 때 입체적으로 표시하지만 실제 배열이 저장되는 메모리는 일차원이기 때문에 이차원 배열도 메모리에는 한 줄로 저장된다. (면우선, 다음 행우선으로 저장 된다.)

3차원 배열을 만드는 방법

자료형 배열명[배열의 면크기][배열의 행크기][배열의 열크기];

예) short arr[2][2][3] = { {{10,20,30},{40,50,60}}, {{70,80,90},{100,110,120}} };

그림 1) 자료형이 short인 2면2행3열 짜리 배열이 메모리에 만들어지는 물리적 형태

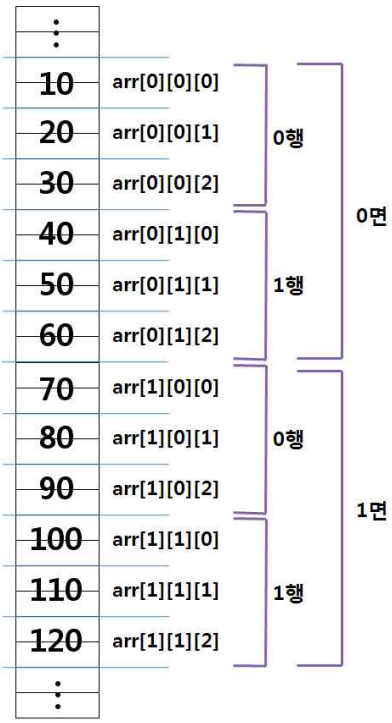
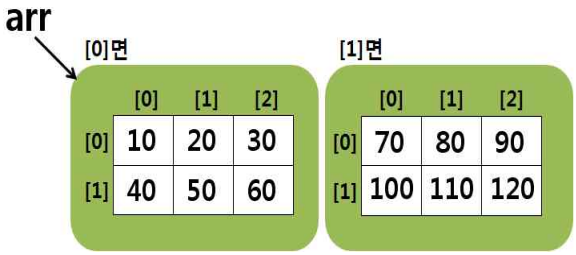


그림 2) 2차원 배열을 표현하는 논리적 형태



3차원 배열안에 있는 원소에 값을 집어 넣거나 배열에 있는 원소의 값을 불러오는 방법

배열명[면번호][행번호][열번호];

예) arr[0][0][0], arr[0][1][2], arr[1][0][2], arr[1][1][3]

배열의 길이를 k면 n행 m열이라 할 때 호출 할 수 있는 원소의 면 번호는 0~k-1면, 행 번호는 0~n-1번 이고 열 번호는 0~m-1번까지 이다.

예) double arr[2][2][3]; 과 같은 형태로 배열을 만들었다면 호출할 수 있는 원소들은 아래와 같다.

arr[0][0][0], arr[0][0][1], arr[0][0][2] => 0면 0행의 원소들
arr[0][1][0], arr[0][1][1], arr[0][1][2] => 0면 1행의 원소들
arr[1][0][0], arr[1][0][1], arr[1][0][2] => 1면 0행의 원소들
arr[1][1][0], arr[1][1][1], arr[1][1][2] => 1면 1행의 원소들

3차원 배열을 만들면서 초기값을 부여하는 방법과 그 예

1. int arr[2][2][3] = { {{10,20,30},{40,50,60}}, {{70,80,90},{100,110,120}} };
   => { } 하나가 한 면을 그 안의 { } 가 한 행을 의미하게 된다.
2. int arr[2][2][3] = { 10,20,30,40,50,60,70,80,90,100,110,120 };
   => 배열에 넣은 값 들이 순서대로 들어간다. 어차피 삼차원 배열도 메모리에는 한 줄로 보관됨으로 메모리에 만들어지는 원소들 순서대로 초기화 시킨 값이 하나씩 들어간다.
3. int arr[2][2][3] = { 0 }; char arr[2][2][3] = {NULL};
   => 일차원 배열과 동일하게 일부의 원소만 초기화 시키면 초기화가 안 된 나머지 원소는 자동으로 0 또는 NULL값으로 지동 초기화 된다. 따라서 위의 방법을 사용하면 모든 원소가 0(문자나 포인터는 NULL)으로 초기화 된다.
4. int arr[][2][3] = { {{10,20,30},{40,50,60}}, {{70,80,90},{100,110,120}} };
   => 삼차원 배열 선언 시 초기 값을 넣어주면 배열의 면 크기를 생략 할 수 있다. (행이나 열 크기는 어떠한 경우에도 생략이 불가능하다. 삼차원 배열 은 원소 하나가 이차원 배열인 배열이기 때문이다. 자세한 설명은 다음 챕터에서 한다.) 이때 배열은 넣어준 초기 값을 보관할 수 있을 만큼의 면을 가지는 배열로 만들어 진다. (초기값이 12개임으로 2면 2행 3열 짜리 삼차원 배열로 만들어진다.)

3차원 배열 전체의 크기는 “원소 하나의 크기 * 열의갯수 * 행수 * 면수”로 계산 할 수 있다.
즉, "한 면의 크기 * 면수" == " ((원소하나의크기 * 열의크기) * 행수) * 면수" 이다.
배열의 크기는 sizeof 연산자를 이용해 배열의 이름으로 측정할 수 있고
한 면의 크기는 sizeof 연산자를 이용해 한 면의 이름(배열의 이름과 인덱스 하나)으로 측정할 수 있다.

예) int arr[2][2][3];

```
printf("sizeof(arr) = %d₩n", sizeof(arr) );   // 배열 arr의 총 크기인 48이 출력된다.
printf("sizeof(arr[0]) = %d₩n", sizeof(arr[0]) );   // [0]면의 크기인 24가 출력된다.
printf("sizeof(arr[0][0]) = %d₩n", sizeof(arr[0][0]) );   // [0]면 [0]행의 크기인 12가 출력된다.
printf("sizeof(arr[0][0][0]) = %d₩n", sizeof(arr[0][0][0]) );   // 원소하나의 크기인 4가 출력된다.
```

삼차원 배열 역시 자신의 인덱스를 변수나 수식으로 대체해 호출 할 수 있다.

예) int arr[2][2][3] = { {{10,20,30},{40,50,60}}, {{70,80,90},{100,110,120}} };

```
int a = 1, b = 1, c=1;
printf("arr[%d][%d][%d]=%d₩n", a, b, c, arr[a][b][c] );
   // a가 1이고 b가 1, c가 1임으로 arr[1][1][1]번째 원소에 들어 있는 110이 출력된다.
```

♲ 3차원 배열의 사용 예제

예제 7-24 : 삼차원 배열을 만들고 사용하는 방법

```
#include <stdio.h>
void main ( ) {
    short arr[2][2][3];
    int i, j, k;
    arr[0][0][0] = 10;
    arr[0][0][1] = 20;
    arr[0][0][2] = 30;
    arr[0][1][0] = 40;
    arr[0][1][1] = 50;
    arr[0][1][2] = 60;
    arr[1][0][0] = 70;
    arr[1][0][1] = 80;
    arr[1][0][2] = 90;
    arr[1][1][0] = 100;
    arr[1][1][1] = 110;
    arr[1][1][2] = 120;
    for(i=0; i<=1; i++) {
        for(j=0; j<=1; j++) {
            for(k=0; k<=2; k++) {
                printf("arr[%d][%d][%d] = %d₩n", i, j, k, arr[i][j][k] );
            }
            printf("₩n");
        }
        printf("₩n");
    }
}
```