

<7장> 배열

학습 목표

- 배열의 개념과 필요성을 이해한다.
- 1차원 배열을 선언하고 초기화한다.
- 함수에 배열을 전달하는 방법을 학습한다.
- 다차원 배열을 이해한다.

목차

01 배열의 필요성과 개념

02 1차원 배열

03 함수와 배열

04 다차원 배열

05 C 언어의 다차원 배열

01

배열의 필요성과 개념

1. 배열의 필요성과 개념

■ 배열의 필요성

- 보안번호가 30개라고 가정

```
int securityNum0;  
int securityNum1;  
int securityNum2;  
...  
int securityNum29;
```

- 번호를 한 개씩 30회 입력받기

```
scanf("%d", &securityNum0);  
scanf("%d", &securityNum1);  
...  
scanf("%d", &securityNum29);
```

1. 배열의 필요성과 개념

■ 배열의 필요성

- 번호 검색도 반복문 등을 활용할 수 없어 코드가 길어지는 문제가 발생

```
switch (num) {  
    case 0:  
        printf("%d\n", securityNum0);  
        break;  
  
    case 1:  
        printf("%d\n", securityNum1);  
        break;  
  
    ...  
  
    case 29:  
        printf("%d\n", securityNum29);  
        break;  
}
```

1. 배열의 필요성과 개념

■ 배열

- 보안번호를 저장한 배열



그림 7-1 보안번호를 저장한 배열이 메모리에 할당됨

1. 배열의 필요성과 개념

■ 배열

- 배열의 특징

- 같은 종류의 자료들을 구조화된 형태로 메모리 공간에 저장하고 한 개의 이름으로 사용할 수 있다.
- 인덱스(index) 번호를 이용해서 자료들에 빠른 접근이 가능하다.
- 반복문과 함께 사용하면 각 요소에 접근하는 효율적인 코드를 작성할 수 있다.
- C 언어의 배열은 포인터 연산과 아주 밀접한 관계를 가지고 있다.

02

1차원 배열

2. 1차원 배열

■ 배열 선언

■ 배열을 선언하는 방법

```
TYPE 배열_이름[요소_개수];
```

- TYPE은 배열 요소에 저장되는 값의 자료형을 나타냄
- C 언어의 배열은 순수형이라서 모든 요소는 같은 자료형으로 구성
- 동일한 크기의 공간을 사용

2. 1차원 배열

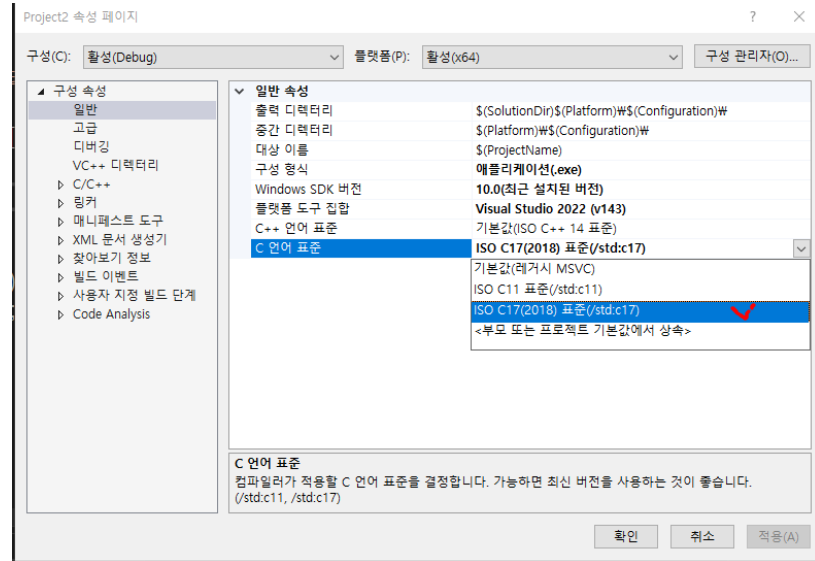
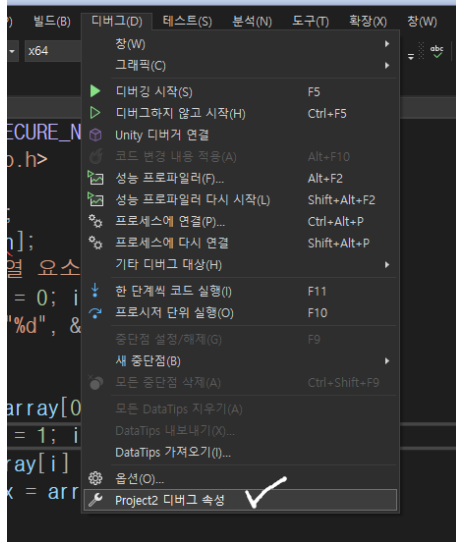
■ 배열 선언

- 배열 선언 코드
- ANSI C에서는
배열 크기는 상수
로만 지정 가능

코드 7-1 ConstArray.c

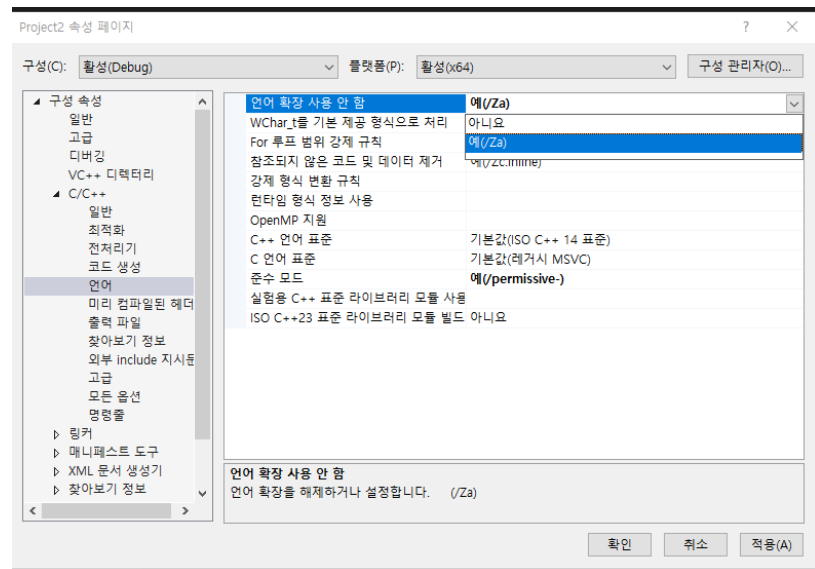
```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define SIZE 10      // 상수 선언
5
6  int main(void)
7  {
8      const int n = 15; // 상수 변수
9      int m = 10;       // 변수
10     int a[m];          ← 변수로 배열 크기 지정 가능(C99 이상)
11     int b[n];          ← 상수 변수로 배열 크기 지정 가능(C99 이상)
12     int c[SIZE];       ← 상수로 배열 크기 지정(ANSI C 이상)
13     int securityNums[30]; ← 네 자리 정수의 보안번호를 저장하는 배열인 securityNums 배열
14                                     (그림 7-1)은 int 형으로 선언
15
16     a[0] = 1;
17     printf("%d\n", a[0]);
18     return 0;
19 }
```

Window 경우 추가설정(여러 문제 발생)- 추후 동적배열권장



<https://learn.microsoft.com/ko-kr/cpp/overview/install-c17-support?view=msvc-170>

<https://seolin.tistory.com/67>



2. 1차원 배열

■ 배열 사용

- 배열의 요소는 변수와 비슷

```
배열_이름[인덱스] = 값;
```

- 일반 변수처럼 표현식의 일부에 사용할 수 있고 함수에 값으로 전달 가능

```
변수 = 배열_이름[인덱스];  
func(배열_이름[인덱스]);
```

- 배열 요소에 저장되는 값이나 배열 요소를 저장하는 변수는 같은 자료형이거나 함께 사용할 수 있는 자료형
 - 예: 정수(int) 배열의 요소에 정수값(int, short, char 등)을 저장

2. 1차원 배열

■ 배열 사용

- 보안번호를 저장하고 화면에 출력

```
securityNums[0] = 1245;  
securityNums[1] = 3152;  
printf("securityNums[0] = %d, securityNums[1] = %d\n", securityNums[0],  
securityNums[1]);
```

- 인덱스로 사용할 수 있는 값의 예시

표 7-1 인덱스로 사용할 수 있는 것들과 설명

예시 코드	설명
securityNums[1]	상수
securityNums[n]	정수형(int, short, char, unsigned int 등) 변수
securityNums[n + 2]	정수형 값을 생성하는 표현식
securityNums[getIdx()]	정수형 값을 반환하는 함수
securityNums['a']	문자(char도 정수형으로 취급)
securityNums[AUG]	enum 자료형 값

2. 1차원 배열

■ 배열 사용

- 정수형이 아닌 값은 인덱스로 사용 불가

```
securityNums[1.0] = 3; // 컴파일 오류
```

- 반복문을 이용해서 securityNums의 값을 화면에 출력

코드 7-2

```
for (int i = 0; i < 30; i++) {  
    printf("보안카드 %d번째 숫자: %d\n", securityNums[i]);  
}
```

2. 1차원 배열

■ C 컴파일러

- 문제가 발생할 수 있는 코드

코드 7-3 ErrorIndex.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define SIZE 3 ← 배열의 크기를 상수로 지정
5
6  int main(void)
7  {
8      int nums[SIZE]; ← SIZE 매크로 상수를 이용해서 요소가 세 개인 배열을 생성
9      nums[0] = 4;
10     nums[1] = 3; ← 배열 초기화
11     nums[2] = 9;
12     //  nums[9] = 8; ← 인덱스가 배열의 범위를 아예 벗어남. 오류 발생 가능성 높음
13     for (int i = 0; i < SIZE; i++) {
14         printf("nums[%d] = %d\n", i, nums[i]); ← 인덱스 번호와 요소 출력
15     }
16     return 0;
17 }
```

〈실행 결과〉

```
nums[0] = 4
nums[1] = 3
nums[2] = 9
```


2. 1차원 배열

■ C 컴파일러

- 12~15줄을 수정하고 파일 이름을 'ErrIndex2.c'로 저장하고 실행
 - 인덱스가 범위를 살짝 벗어남

코드 7-4 ErrorIndex2.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #pragma warning(disable: 4789)
3  #include <stdio.h>
4
5  #define SIZE 3
6
7  int main(void)
8  {
9      int nums[SIZE];
10     int n = 4;
11     nums[0] = 4;
12     nums[1] = 3;
13     nums[2] = 9;
14     nums[3] = 10;
15     for (int i = 0; i < 10; i++) {
16         printf("nums[%d] = %d\n", i, nums[i]);
17     }
18     printf("n = %d\n", n);
19     return 0;
20 }
```

〈실행 결과〉

```
nums[0] = 4
nums[1] = 3
nums[2] = 9
nums[3] = 10
nums[4] = -20877296
n = 10
```

2. 1차원 배열

■ C 컴파일러

- nums 배열의 메모리 사용 현황
- 오류가 나타나지 않을 수도 있음(하지만 버그)

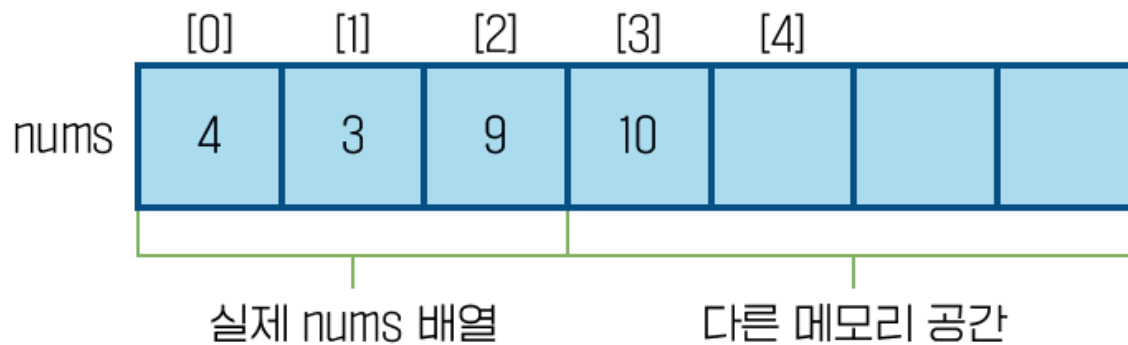


그림 7-2 잘못된 인덱스를 사용하는 상황의 메모리 공간

2. 1차원 배열

■ sizeof() 연산자와 배열

- sizeof(배열_이름)을 사용하면 배열의 모든 요소가 차지하는 메모리 공간의 크기를 반환
- 배열의 크기가 N이라면 sizeof(배열 요소) * N이 반환
- sizeof(배열_이름[인덱스])를 사용하면 배열 요소 한 개의 크기가 바이트 단위로 반환

코드 7-5 SizeArr.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define SIZE 5
5
6  int main(void)
7  {
8      int nums[SIZE];
9      printf("size of nums[0] = %d\n", sizeof(nums[0]));
10     printf("size of nums = %d\n", sizeof(nums));
11     return 0;
12 }
```

배열 요소 한 개의 크기 출력

배열 전체 공간의 크기 출력

〈실행 결과〉

size of nums[0] = 4

size of nums = 20

2. 1차원 배열

■ 배열 초기화

- 배열을 선언할 때 전역이나 지역 정적 배열로 선언하면 자동으로 초기화
- 하지만 일반 지역변수로 선언되는 배열은 초기화되지도 않고 쓰레기 (Garbage) 값이 존재

코드 7-6 InitArr1.c

```
1  #include <stdio.h>
2
3  int globalNums[5];
4
5  int main(void)
6  {
7      static int staticNums[5];
8      int localNums[5];
9
10     for (int i = 0; i < 5; i++) {
11         printf("globalNums[%d] = %d, staticNums[%d] = %d, localNums[%d] = %d\n", i, globalNums[i], i, staticNums[i], i, localNums[i]);
12     }
13     return 0;
14 }
```

2. 1차원 배열

■ 배열 초기화

- 전역 배열, 정적 배열, 지역 배열 변수들을 선언하고 있는 그대로의 배열 요소 값들을 출력

〈실행 결과〉

```
globalNums[0] = 0, staticNums[0] = 0, localNums[0] = 1  
globalNums[1] = 0, staticNums[1] = 0, localNums[1] = 0  
globalNums[2] = 0, staticNums[2] = 0, localNums[2] = 2  
globalNums[3] = 0, staticNums[3] = 0, localNums[3] = 0  
globalNums[4] = 0, staticNums[4] = 0, localNums[4] = -679603328
```

- globalNums와 staticNums는 몇 번 실행해도 0을 출력
- localNums는 실행할 때마다 값이 조금 변경되기도 하고 컴파일러에 따라 다른 값을 출력

2. 1차원 배열

■ 배열 초기화

- 전역 또는 정적 배열을 0이 아닌 다른 값으로 초기화시킬 때 사용
- 지역변수로 선언되는 배열을 초기화시킬 때 사용

- 선언한 뒤에 초기화
- 선언하면서 초기화

2. 1차원 배열

■ 배열 초기화

■ 선언 후 초기화

- 선언한 뒤에 초기화시키는 것은 요소 값을 직접 초기화
- 반복문 이용 가능

```
1 securityNums[0] = 1234;
2 securityNums[28] = 5532;
3 for (int i = 0; i < 30; i++) {
4     securityNums[i] = 0;
5 }
```

첫 번째와 29번째 요소의 값 저장

반복문을 이용해서 배열의 모든 요소를 0으로 초기화

※ 보안번호 1000부터 9999까지의 정수를 무작위로 생성해 초기화하고 화면 출력하는 프로그램

➔ [코드 7-7](#)
➔ [실행 결과](#)

2. 1차원 배열

■ 배열 초기화

■ 선언하면서 초기화

- 배열을 선언하고 중괄호 안에 배열 요소의 값 지정

```
TYPE 배열_이름[] = { 값0, 값1, ..., 값n-1 };
```

```
TYPE 배열_이름[요소_개수] = { 값0, 값1, ..., 값n-1 }; // 요소_개수가 n
```

- nums 배열을 5개 정수를 요소로 가지는 배열로 선언

코드 7-8

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

- 코드 7-8은 다음 형태와 같음

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


2. 1차원 배열

■ 배열 초기화

■ 선언하면서 초기화

- 배열의 개수를 지정하지 않았을 때 배열 크기 확인

```
배열_크기 = sizeof(배열_이름) / sizeof(TYPE)  
배열_크기 = sizeof(배열_이름) / sizeof(배열_이름[0])
```

- 요소 크기로 나누는 방법을 이용해 매크로 함수를 만들어 사용

```
#define ARRAY_SIZE(arr, element) (sizeof((arr)) / sizeof((element)))
```

※ 배열을 크기 없이 초기화하고 ARRAY_SIZE()
매크로 함수를 이용해 화면에 출력하는 프로그램

→ [코드 7-9](#)
→ [실행 결과](#)

2. 1차원 배열

■ 배열 초기화

- 배열의 크기를 정하고 초기화하는 방법

표 7-2 배열을 초기화할 때 요소 개수와 값의 개수 관계 및 설명

요소 개수와 초깃값 개수의 관계	설명
요소 개수 == 초깃값 개수	첫 번째 배열 요소(인덱스 0)부터 초기자의 값을 순서대로 저장한다.
요소 개수 > 초깃값 개수	첫 번째 배열 요소부터 초기자의 값을 순서대로 저장하고, 남은 공간은 0에 해당하는 값 (float이나 double 형은 0.0f 또는 0.0, char 형은 널 문자('\0'), 포인터는 NULL 등)으로 저장한다.
요소 개수 < 초깃값 개수	배열 요소를 순서대로 초깃값으로 채우지만, 컴파일러에 따라 경고 또는 오류가 발생할 수 있다.

2. 1차원 배열

■ 배열 초기화

- 배열 요소와 초기자 개수가 일치할 때
 - 순서대로 값을 배열 요소에 저장

```
int num1[5] = { 1, 2, 3, 4, 5 }; // 요소_개수 == 초깃값 개수
```

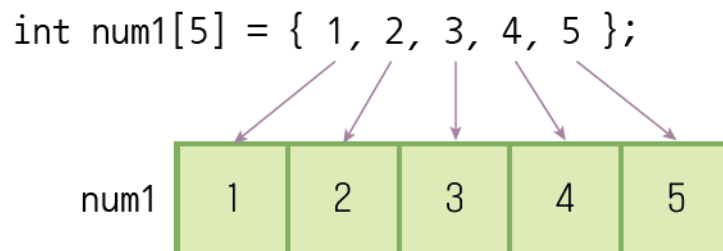


그림 7-3 요소_개수와 초깃값 개수가 일치

2. 1차원 배열

■ 배열 초기화

- 요소 개수가 초기화 개수보다 클 때
 - 요소를 순서대로 채우고 남은 공간은 0에 근접한 값으로 채움

```
int num2[5] = { 1, 2, 3 };
```

```
int num2[5] = { 1, 2, 3 };
```

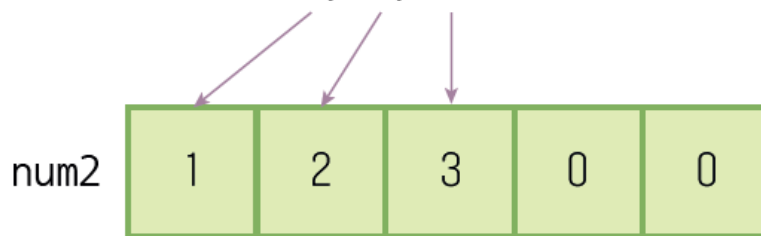


그림 7-4 요소_개수 > 초기값 개수

2. 1차원 배열

■ 배열 초기화

- 초기자 개수가 요소 개수보다 클 때
 - 배열의 모든 요소를 채울 때까지 순서대로 초기자의 값을 배열 요소에 저장

```
int num2[5] = { 1, 2, 3, 4, 5, 6 };
```

```
int num3[5] = { 1, 2, 3, 4, 5, 6 };
```

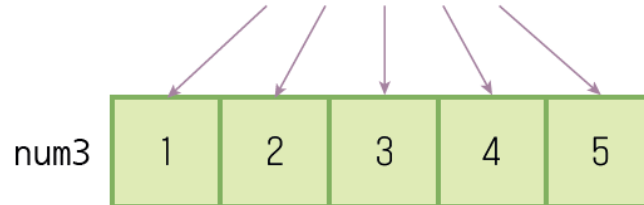


그림 7-5 요소_개수 < 초깃값 개수

2. 1차원 배열

■ 배열 초기화

- 코드 7-10 실행 후 표 7-2 다시 확인

코드 7-10 InitArr3.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define SIZE 5
5
6  int main(void)
7  {
8      int num1[SIZE] = { 1, 2, 3, 4, 5 }; // 요소_개수 == 초깃값 개수
9      int num2[SIZE] = { 1, 2, 3 };       // 요소_개수 > 초깃값 개수
10     int num3[SIZE] = { 1, 2, 3, 4, 5, 6 }; // 요소_개수 < 초깃값 개수
11
12     for (int i = 0; i < SIZE; i++) {
13         printf("num1[%d] = %d, num2[%d] = %d, num3[%d] = %d\n", i, num1[i], i,
14             num2[i], i, num3[i]);
15     }
16     return 0;
17 }
```

〈실행 결과〉

```
num1[0] = 1, num2[0] = 1, num3[0] = 1
num1[1] = 2, num2[1] = 2, num3[1] = 2
num1[2] = 3, num2[2] = 3, num3[2] = 3
num1[3] = 4, num2[3] = 0, num3[3] = 4
num1[4] = 5, num2[4] = 0, num3[4] = 5
```

2. 1차원 배열

■ 배열과 대입 연산

- 배열은 요소별로 값을 저장하거나 접근하는 것만 가능
- 배열 전체를 대입 연산에 사용 불가
- 배열을 선언하고 초기화할 때 중괄호에 초기자를 지정해서 배열 요소 각각에 값을 저장 가능

```
int arr[2] = { 1, 2 };
```

- 일반 대입 연산에서는 요소별로 값을 저장하거나 사용해야 함

```
int arr[2];  
arr = { 1, 2 }; // 절대 불가
```

2. 1차원 배열

■ 배열과 대입 연산

- 오류가 발생하는 몇 가지 경우

코드 7-11

```
1  #define SIZE 2
2
3  int arr1[SIZE] = { 1, 2 };
4  int arr2[SIZE] = arr1; // 컴파일 오류 발생
5  int arr3[SIZE];
6  arr3 = arr1;           // 컴파일 오류 발생
```


2. 1차원 배열

■ 배열과 대입 연산

- 코드 7-12가 제대로 동작하도록 수정

코드 7-12

```
1  #define SIZE 2
2
3  int arr1[SIZE] = { 1, 2 };
4  int arr2[SIZE] = { arr1[0], arr1[1] };
5  int arr3[SIZE];
6  arr3[0] = arr1[0];
7  arr3[1] = arr1[1];
```

2. 1차원 배열

■ 배열과 대입 연산

- 배열의 요소 개수가 많다면 반복문을 이용

코드 7-13

```
1  #define SIZE 5
2
3  int arr1[SIZE] = { 1, 2, 3, 4, 5 };
4  int arr2[SIZE];
5
6  for (int i = 0; i < SIZE; i++) {
7      arr2[i] = arr1[i];
8  }
```

03

함수와 배열

3. 함수와 배열

■ 함수에 배열 요소 전달

- 함수에 배열의 요소를 전달하는 것은 일반 변수를 전달하는 것과 동일

■ 예시 프로그램 작성

- 1 인치(inch)는 2.54cm로 계산하고, 배열을 선언하면서 값을 초기화

```
double tvSizes[] = { 32 * 2.54, 42 * 2.54, 55 * 2.54, 65 * 2.54, 75 * 2.54 };
```

- double 값 한 개를 전달받아 화면에 출력하는 함수 작성

```
void printTVSize(double size) { printf("TV size: %f cm\n", size); }
```

- 함수에 배열 요소를 한 개씩 전달해서 화면에 출력

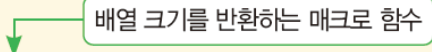
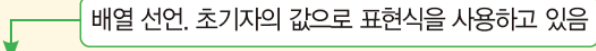
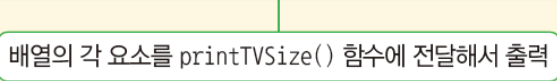
```
printTVSize(tvSizes[0]); // 첫 번째 요소의 값을 함수에 복사해서 전달
```

3. 함수와 배열

■ 함수에 배열 요소 전달

■ 예시 프로그램 작성

코드 7-14 PrintTVSize.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3   배열 크기를 반환하는 매크로 함수
4  #define ARRAY_SIZE(arr, element) (sizeof((arr)) / sizeof((element)))
5
6  void printTVSize(double size) { printf("TV size: %f cm\n", size); }
7
8  int main(void)
9  {  배열 선언. 초기자의 값으로 표현식을 사용하고 있음
10     double tvSizes[] = { 32 * 2.54, 42 * 2.54, 55 * 2.54, 65 * 2.54, 75 * 2.54 };
11
12     for (int i = 0; i < ARRAY_SIZE(tvSizes, tvSizes[0]); i++) {  배열의 각 요소를 printTVSize() 함수에 전달해서 출력
13         printTVSize(tvSizes[i]);
14     }
15     return 0;
16 }
```

3. 함수와 배열

■ 함수에 배열 요소 전달

- 예시 프로그램 작성
 - 프로그램이 동작하는 방법

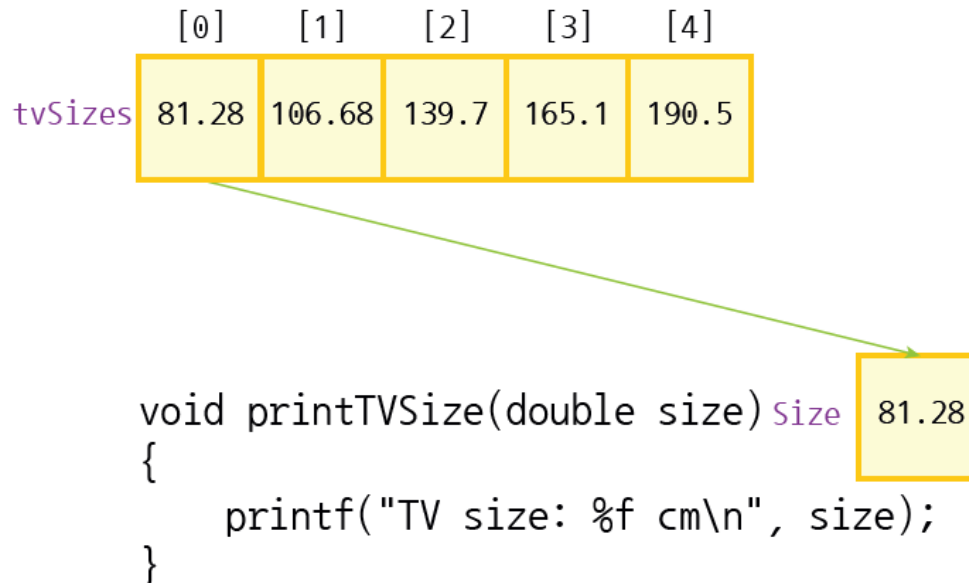


그림 7-6 코드 7-14 동작 방법

3. 함수와 배열

■ 함수에 배열 요소 전달

■ 예시 프로그램 작성

- printTVSize()의 size 매개변수에 전달
- printTVSize()에서는 전달된 값을 화면에 출력

<실행 결과>

```
TV size: 81.280000 cm  
TV size: 106.680000 cm  
TV size: 139.700000 cm  
TV size: 165.100000 cm  
TV size: 190.500000 cm
```

3. 함수와 배열

■ 함수에 배열 전달

- 함수에 배열을 인자로 전달할 때는 배열의 이름을 사용

■ 예시 코드

- printTVSizes() 함수는 배열을 전달받고 모든 요소의 내용을 화면에 출력한다고 가정

```
printTVSizes(tvSizes); // 함수를 호출하면서 배열을 입력으로 전달
```

- printTVSizes() 함수 구현

코드 7-15

```
1 void printTVSizes(double sizes[])
2 {
3     for (int i = 0; i < 5; i++) {
4         printf("TV size: %f cm\n", sizes[i]);
5     }
6 }
```


※ 함수에 배열을 전달하는 코드 완성

➔ [코드 7-16](#)

➔ [실행 결과](#)

3. 함수와 배열

■ 함수에 배열 크기 함께 전달

- 매개변수로 전달된 배열의 크기 출력

코드 7-17 PrintArrSize.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  void printArrSize(int arr[])
5  {
6      printf("sizeof(arr) = %zu\n", sizeof(arr));
7      printf("sizeof(arr[1]) = %zu\n", sizeof(arr[1]));
8  }
9
10 int main(void)
11 {
12     int nums[] = { 1, 2, 3, 4, 5 };
13     printf("sizeof(nums) = %zu\n", sizeof(nums));
14     printArrSize(nums);
15     return 0;
16 }
```

〈실행 결과〉

```
sizeof(nums) = 20
sizeof(arr) = 8
sizeof(arr[1]) = 4
```

3. 함수와 배열

■ 함수에 배열 크기 함께 전달

- 함수에 배열이 전달되는 과정
 - 배열 변수 이름은 배열의 시작 주소를 나타내는 상수 변수
 - 배열은 선언할 때 첫 번째 요소의 메모리 주소가 저장되는 변수
 - 배열의 시작 주소(첫 번째 배열 요소의 주소)만 전달

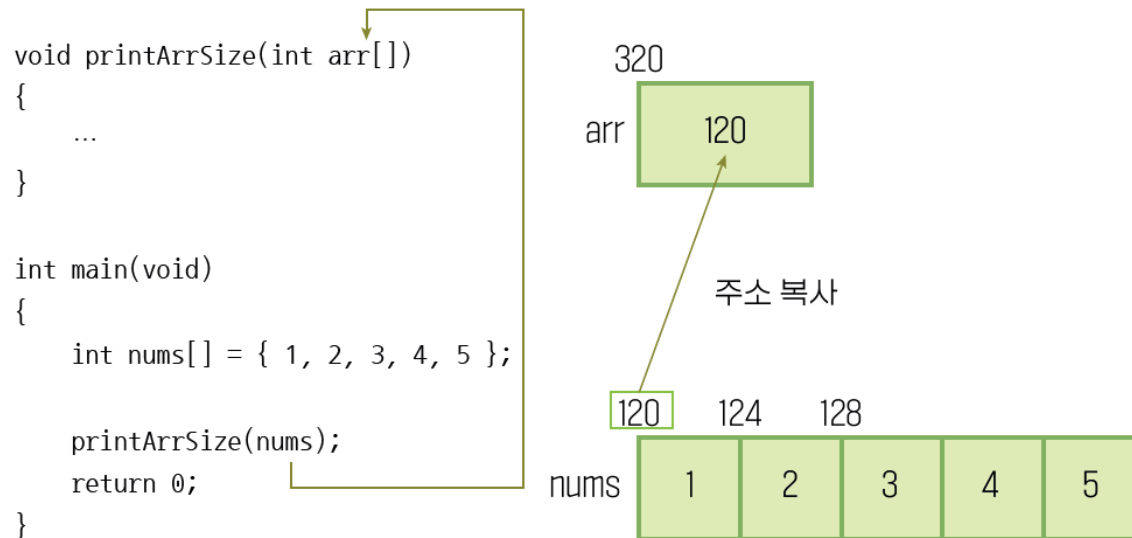


그림 7-7 인자로 배열을 함수에 전달하는 과정

3. 함수와 배열

■ 함수에 배열 크기 함께 전달

- 주소가 복사됨을 확인

코드 7-18 ArrayAddress.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  void printArrAddr(int arr[])
5  {
6      printf("address of arr = %p\n", arr); ← 매개변수 arr에 저장된 값이 출력됨
7  }
8
9  int main(void)
10 {
11     int nums[] = { 1, 2, 3, 4, 5 };
12     printf("address of nums = %p\n", nums); ← 배열 이름 nums의 값을 출력  
배열의 시작 주소가 출력됨
13     printf("address of nums[0] = %p\n", &nums[0]);
14     printArrAddr(nums); ← 배열의 시작 주소(배열 첫 번째 요소의 시작 주소)를 출력
15     return 0;
16 }
```

<실행 결과>

```
address of nums = 000000c7ecbffa0
address of nums[0] = 000000c7ecbffa0
address of arr = 000000c7ecbffa0
```

3. 함수와 배열

■ 함수에 배열 크기 함께 전달

■ 함수에 배열과 크기 함께 전달

- 코드 7-16에서 만들었던 printTVSizes() 함수를 다시 구현하고 사용

코드 7-19 PrintTVSizes2.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define ARRAY_SIZE(arr, element) (sizeof((arr)) / sizeof((element)))
5
6  void printTVSizes(double sizes[], int size)
7  {
8      for (int i = 0; i < size; i++) { // 배열의 크기만큼 출력
9          printf("TV size: %f cm\n", sizes[i]);
10     }
11 }
12
13 int main(void)
14 {
15     double tvSizes[] = { 81.28, 106.68, 139.70, 165.10, 190.50 };
16
17     printTVSizes(tvSizes, ARRAY_SIZE(tvSizes, tvSizes[0]));
18     return 0;
19 }
```

3. 함수와 배열

■ 함수에서 전달받은 배열 수정

- 기본 자료형 변수와 배열이 함수에 전달

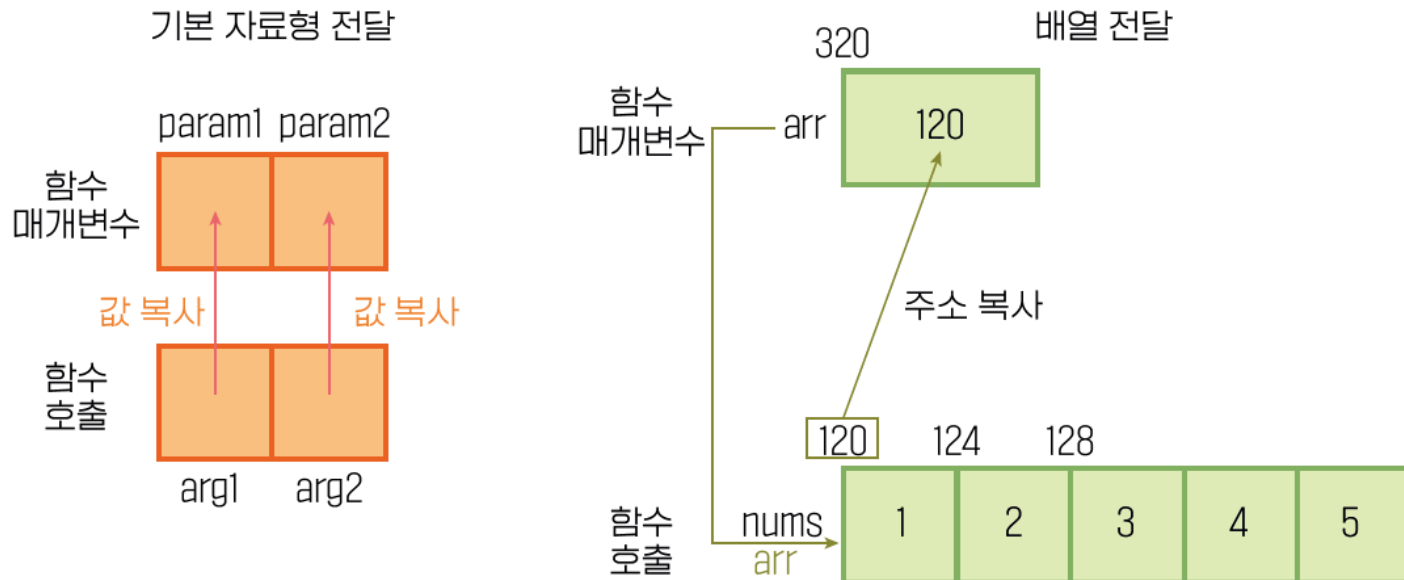


그림 7-8 기본 자료형 변수와 배열이 함수에 전달될 때 발생하는 모습

3. 함수와 배열

■ 함수에서 전달받은 배열 수정

■ 예제 프로그램

코드 7-20 IncreaseArr.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  void increaseArray(int arr[], int size)
5  {
6      for (int i = 0; i < size; i++) {
7          arr[i]++; // nums 요소를 1씩 증가시킴
8      }
9  }
10
11 int main(void)
12 {
13     int nums[5] = { 1, 2, 3, 4, 5 };
14     increaseArray(nums, 5);
15     for (int i = 0; i < 5; i++) {
16         printf("%d ", nums[i]);
17     }
18     return 0;
19 }
```

〈실행 결과〉

2 3 4 5 6

3. 함수와 배열

■ 함수에서 전달받은 배열 수정

■ 배열 요소 교환

코드 7-21

```
1 void swapArrayElements(int arr[], int idx1, int idx2)
2 {
3     int temp = arr[idx1]; ← 교환할 배열의 요소 한 개(arr[idx1])를 임시 변수 temp에 저장
4     arr[idx1] = arr[idx2]; ← 임시 변수에 값을 복사한 요소 공간에 다른 요소 값(arr[idx2])을 저장
5     arr[idx2] = temp; ← arr[idx2]에 temp 값을 저장
6 }
```


※ 배열 요소 교환 함수를 사용하는 코드

➔ [코드 7-22](#)

➔ [실행 결과](#)

3. 함수와 배열

■ 함수에서 매개변수 배열 수정 방지

- 함수 안에서 배열 요소의 값을 실수로 변경하지 못하도록 사전에 방지
 - const를 붙임

코드 7-23

```
1 void printArrayElements(const int arr[], int size)
2 {
3     for (int i = 0; i < size; i++) {
4         printf("%d ", arr[i]);
5     }
6     printf("\n");
7 }
```

3. 함수와 배열

■ 매개변수를 이용한 배열 선언(C99, 비주얼 스튜디오에서 안됨)

- 요소 이동 과정

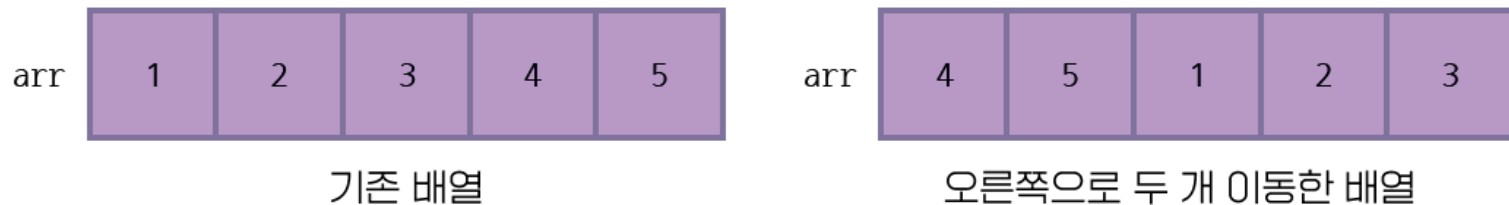


그림 7-9 배열의 요소를 오른쪽으로, 마지막 요소를 첫 번째 위치로 이동

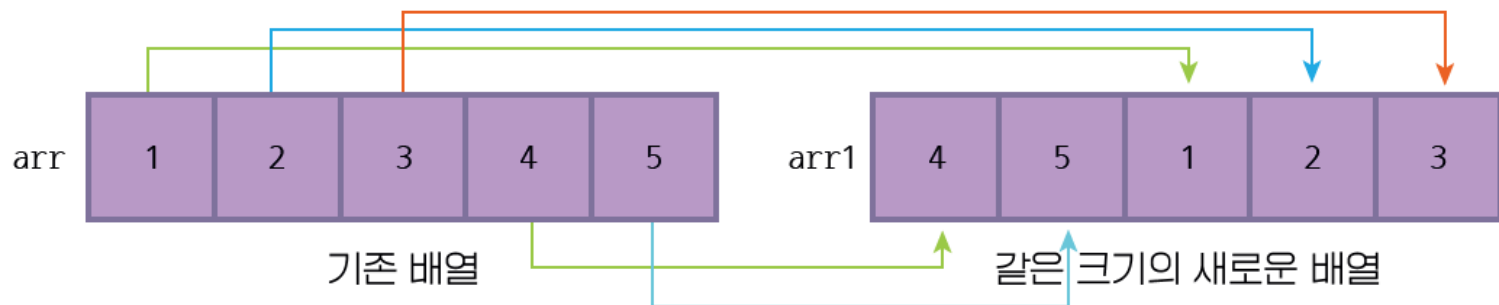


그림 7-10 함수 구현 방법

3. 함수와 배열

■ 매개변수를 이용한 배열 선언(C99, 비주얼 스튜디오 안됨)

- 새로 생성한 배열의 내용을 기존 배열에 다시 복사하여 배열 요소를 이동하는 함수

코드 7-24

```
1  int shiftArray(int arr[], int size, int n)
2  {
3      int arr1[size];
4
5      for (int i = 0; i < size - n; i++) {
6          arr1[i + n] = arr[i];
7      }
8      for (int i = size - n, j = 0; i < size; i++, j++) {
9          arr1[j] = arr[i];
10     }
11     for (int i = 0; i < size; i++) {
12         arr[i] = arr1[i];
13     }
14 }
```

원본 배열과 배열 크기, 이동할 요소 개수를 전달받음

원본 배열의 크기와 동일한 배열을 생성

arr 처음 요소부터 새로운 배열 arr1의 n 위치로 요소 복사 시작. arr1의 끝에 도달할 때까지 복사

arr에 남은 요소가 있다면 arr1의 처음부터 복사 시작

arr1의 모든 내용을 arr에 다시 순차적으로 복사 시작

※ 배열의 요소를 출력하는 함수와 검수 코드

➔ [코드 7-25](#)

➔ [실행 결과](#)

3. 함수와 배열

■ 문자를 입력받고 출력하는 프로그램(C99, 비주얼 스튜디오 안 됨)

- 문자 배열을 출력하는 함수

코드 7-26

전달된 문자 배열과 배열 크기에 맞춰 배열에 있는 글자들을 한 개씩 출력하는 함수를 구현
함수 내부에서 s 배열의 내용을 수정하지 않으므로 s 배열에 const를 붙임

```
1 void printChars(const char s[], int size)
2 {
3     printf("문자 배열 출력: ");
4     for (int i = 0; i < size; i++) {
5         printf("%c ", s[i]);
6     }
7     printf("\n");
8 }
```

함수에 주어진 배열에서 size - 1까지
문자를 한 개씩 공백을 두고 출력

- '0'을 입력받거나 정해진 개수(n)의 문자를 입력받는 알고리즘

count = 0 // 배열의 첫 번째 요소 인덱스

글자 입력받고 배열의 count 인덱스에 한 개씩 저장(인덱스 증가 포함)

count를 1 증가

만약 글자가 '0'이 아니거나 count가 n 미만이라면 처음부터 반복

3. 함수와 배열

■ 문자를 입력받고 출력하는 프로그램(C99, 비주얼 스튜디오 안 됨)

- 알고리즘을 구현한 코드

코드 7-27

```
1 void getAndPrintChars(int n)
2 {
3     char chars[n]; // n개의 문자를 저장할 수 있는 배열 선언
4     int count = 0; // 사용자가 입력하는 문자의 개수
5     do {
6         printf("문자 한 개를 입력하세요: ");
7         scanf("%c", &chars[count]);
8         getchar(); // 엔터 키의 줄 바꿈 문자 제거
9         count++;
10    } while (chars[count - 1] != '0' && count < n);
11    printChars(chars, count);
12 }
```

사용자가 입력한 최대 문자 개수만큼 배열 선언

사용자가 입력한 문자 개수를 세는 변수

문자 입력받고 배열에 저장
사용자가 '0'을 입력하거나
최대 개수만큼 문자가 입력
되면 종료

입력한 문자들을 화면에 출력

3. 함수와 배열

■ 문자를 입력받고 출력하는 프로그램(C99, 비주얼 스튜디오 안 됨)

- 문자를 읽을 때 입력 버퍼



그림 7-11 문자를 읽을 때의 입력 버퍼 모습

※ 사용자로부터 문자를 입력받고 출력하는 프로그램 완성

➔ [코드 7-28](#)
➔ [실행 결과](#)

3. 함수와 배열

■ 약수를 생성해서 출력하는 프로그램

- 함수에 전달한 배열에 약수를 구해서 저장

코드 7-29

```
1  int findDivisors(int num, int divisors[], int size)
2  {
3      int count = 0;
4      for (int i = 1; i <= num; i++) {
5          if (num % i == 0) {
6              divisors[count] = i;
7              count++;
8              if (count >= size) { return 0; }
9          }
10     }
11     return count;
12 }
```

num은 약수를 찾을 정수
divisors[]는 약수를 저장할 배열
size는 배열 크기

약수를 저장한 위치를 기록할 변수

약수이면 배열에 추가하고 위치 변수를 1 증가시킴

약수를 배열을 꽉 채울 정도로 찾았으면 0 반환

※ 약수를 생성해서 출력하는 프로그램 완성

- ➔ [코드 7-30](#)
- ➔ [실행 결과](#)
- ➔ [SIZE가 10일 때 실행 결과](#)

04

다차원 배열

4. 다차원 배열

■ 2차원 배열

- 5개의 정수를 저장할 수 있는 1차원 배열

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

- arr 배열의 메모리 구조

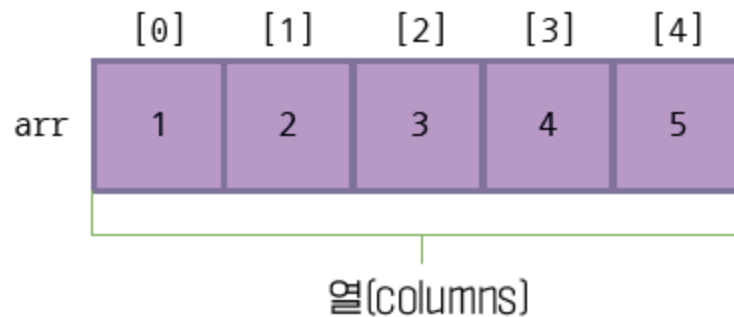


그림 7-12 1차원 배열의 메모리 구조

4. 다차원 배열

■ 2차원 배열

- 2차원 배열은 열(column)과 행(row)으로 구성된 표

	<code>[] [0]</code>	<code>[] [1]</code>	<code>[] [2]</code>	<code>[] [3]</code>	<code>[] [4]</code>	
<code>arr2[0][]</code>	1	2	3	4	5	행(rows)
<code>arr2[1][]</code>	6	7	8	9	10	
<code>arr2[2][]</code>	11	12	13	14	15	
열(columns)						

그림 7-13 2차원 배열의 개념도

4. 다차원 배열

■ 2차원 배열의 선언과 메모리 구조

- 2차원 배열 선언 방법

```
TYPE 배열_이름[ROWS][COLUMNS];
```

- 정수형 2차원 배열 선언

```
int arr2[3][5];
```

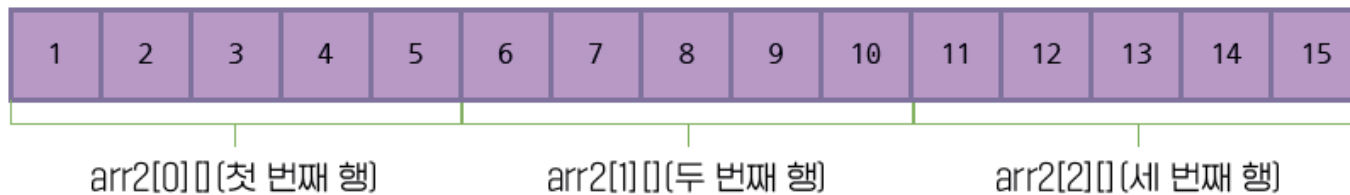


그림 7-14 2차원 배열의 메모리 저장 구조

4. 다차원 배열

■ 2차원 배열의 선언과 메모리 구조

- C 언어의 2차원 배열
- 2차원 배열은 1차원 배열을 요소로 가지는 배열

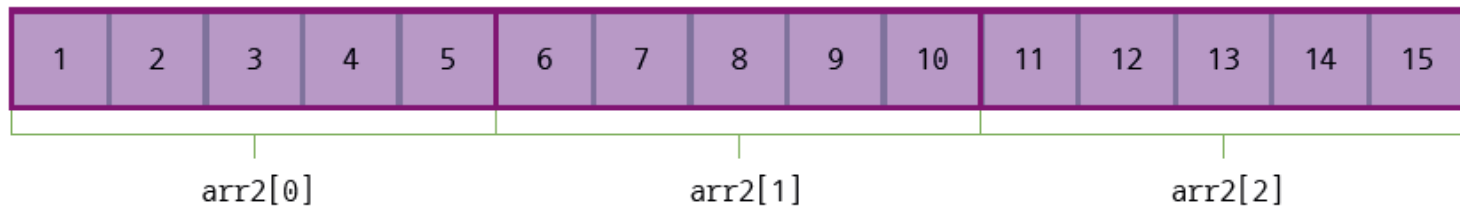


그림 7-15 2차원 배열은 1차원 배열의 배열

4. 다차원 배열

■ 2차원 배열 사용

- 중첩 반복문의 두 개 for 반복문이 2차원 배열의 행과 열에 적용되는 방법

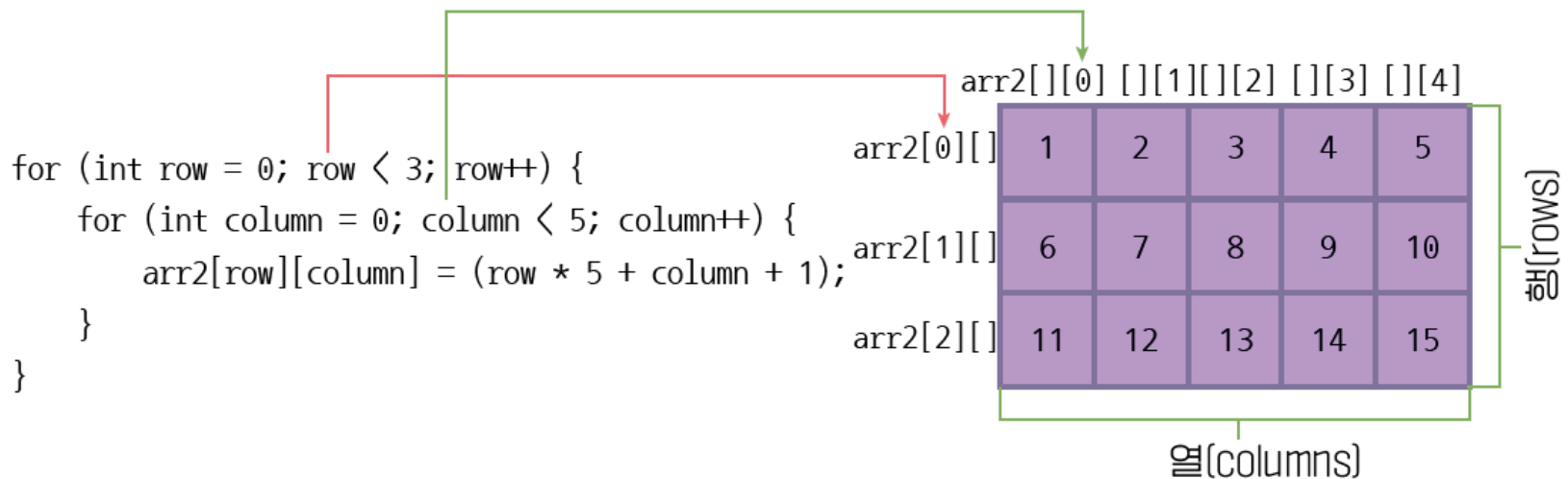


그림 7-16 2차원 배열 요소에 접근하는 중첩 반복문의 사용법

4. 다차원 배열

■ 2차원 배열 사용

- arr2를 화면에 출력하는 코드

코드 7-31

```
1  for (int row = 0; row < 3; row++) {  
2      for (int column = 0; column < 5; column++) {  
3          arr2[row][column] = (row * 5 + column + 1);  
4      }  
5  }
```

간단한 수식을 이용해서
1~15까지의 값을 지정

- 대입 연산 대신 출력 코드 사용

코드 7-32

```
1  for (int i = 0; i < 3; i++) {  
2      for (int j = 0; j < 5; j++) {  
3          printf("%d\n", arr2[i][j]); // 대입 대신 출력 코드 사용  
4      }  
5  }
```

※ arr2 배열에 1~15를 저장하고 arr2[0][0]과
arr2[1][3]를 출력하는 코드

➔ [코드 7-33](#)

➔ [실행 결과](#)

4. 다차원 배열

■ 2차원 배열 초기화

- arr2를 1~15의 값으로 초기화

코드 7-34

```
int arr2[3][5] = { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 }, { 11, 12, 13, 14, 15 } };
```

```
int arr2[3][5] = { { 1, 2, 3, 4, 5 },  
                  { 6, 7, 8, 9, 10 },  
                  { 11, 12, 13, 14, 15 } };
```

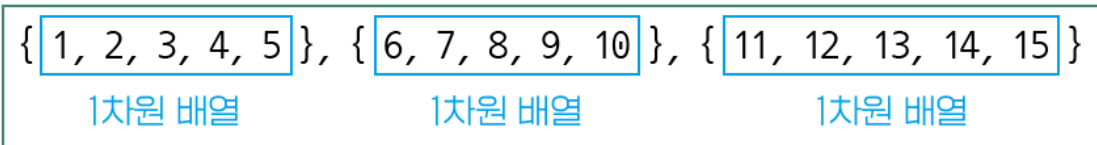
그림 7-17 2차원 배열을 행(row)별로 초기화

4. 다차원 배열

■ 2차원 배열 초기화

- 1차원 배열을 초기화한 것들을 이용해서 2차원 배열을 초기화

```
int arr2[3][5] = { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 }, { 11, 12, 13, 14, 15 } };
```



1차원 배열의 배열 → 2차원 배열

그림 7-18 2차원 배열을 1차원 배열의 조합으로 초기화

4. 다차원 배열

■ 2차원 배열 초기화(값을 모두 지정하지 않는 경우)

- 첫 번째 행은 1, 2, 3, 4, 5, 두 번째 행은 6, 7, 8, 0, 0, 세 번째 행은 0, 0, 0, 0, 0으로 초기화

코드 7-35

```
int arr2[3][5] = { { 1, 2, 3, 4, 5 },  
                  { 6, 7, 8 }  
                };
```

- arr2를 개수를 정하지 않고 초기화

코드 7-36

```
int arr2[][5] = { { 1, 2, 3, 4, 5 },  
                 { 6, 7, 8 }  
               };
```

4. 다차원 배열

■ 2차원 배열 초기화

- 인덱스 크기가 정해지지 않았을 때 사용하는 공식

행의 개수 = `sizeof(배열_이름) / sizeof(배열_요소의_자료형) / (열의 개수)`

행의 개수 = `sizeof(배열_이름) / sizeof(가장 작은 배열_요소 한 개) / (열의 개수)`

- 코드 7-37의 arr2의 행 개수를 찾는 코드

```
sizeof(arr2) / sizeof(arr2[0][0]) / 5;
```

4. 다차원 배열

■ 2차원 배열 초기화

- 행의 개수를 확인하는 코드

코드 7-37 RowsOf2DArr.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6      int arr2[][5] = { { 1, 2, 3, 4, 5 }, { 6, 7, 8 }, { 11, 12 } };
7      int rows = sizeof(arr2) / sizeof(arr2[0][0]) / 5;
8      printf("rows = %d\n", rows);
9      return 0;
10 }
```

<실행 결과>

rows = 3

4. 다차원 배열

■ 2차원 배열 초기화

- arr2 배열을 동일한 값으로 초기화

코드 7-38

```
int arr2[3][5] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
```

- 초기화할 때 첫 번째 인덱스는 생략 가능

```
int arr2[][5] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
```


※ 한 줄에 한 행의 배열을 모두 출력하는 코드

➔ [코드 7-39](#)

➔ [실행 결과](#)

4. 다차원 배열

■ 함수에 2차원 배열 전달

- 함수에 2차원 배열을 전달하는 방법

- 매개변수에 2차원 배열의 크기를 정확하게 표기하기
- 매개변수에 2차원 배열의 첫 번째 인덱스 부분을 비우기

- arr2를 전달받아 화면에 출력하는 함수 원형

```
void print2DArray(const int arr[3][5], int size); // 배열 크기를 명확하게 명시  
void print2DArray(const int arr[][5], int size); // 첫 번째 인덱스를 생략
```

※ arr2를 전달받아 화면에 출력하는 함수

➔ [코드 7-40](#)

4. 다차원 배열

■ 함수에 다차원 배열의 부분 배열 전달

- 부분 배열의 주소를 나타내는 변수로 사용

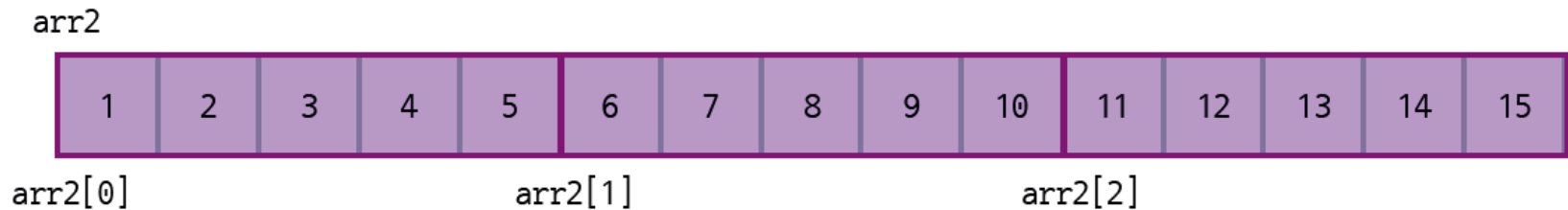


그림 7-19 2차원 배열의 이름[인덱스] 1차원 부분 배열의 주소를 나타냄

4. 다차원 배열

■ 함수에 다차원 배열의 부분 배열 전달

- 부분 배열의 메모리 주소를 확인하는 프로그램

코드 7-41 TwoDArrayAddr.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6      int arr2[3][5];
7
8      printf("arr2 = %p\n", arr2);
9      printf("address of arr2[0][0] = %p\n", &arr2[0][0]);
10     printf("arr2[0] = %p\n", arr2[0]);
11     printf("arr2[1] = %p\n", arr2[1]);
12     printf("address of arr2[1][0] = %p\n", &arr2[1][0]);
13     printf("arr2[2] = %p\n", arr2[2]);
14     printf("address of arr2[2][0] = %p\n", &arr2[2][0]);
15     return 0;
16 }
```

arr2, arr2[0][0]의 주소, arr2[0]은 모두 동일한 주소

arr2[1]과 arr2[1][0]의 주소는 동일

arr2[2]과 arr2[2][0]의 주소는 동일

<실행 결과>

```
arr2 = 0000002bc37ff920
address of arr2[0][0] = 0000002bc37ff920
arr2[0] = 0000002bc37ff920
arr2[1] = 0000002bc37ff934
address of arr2[1][0] = 0000002bc37ff934
arr2[2] = 0000002bc37ff948
address of arr2[2][0] = 0000002bc37ff948
```

※ printArrayElements() 함수를 활용해 2차원 배열을 출력하는 코드

➔ [코드 7-42](#)

➔ [실행 결과](#)

4. 다차원 배열

■ 3차원 배열

- 3차원 배열을 선언하는 방법

```
TYPE 배열_이름[DEPTH][ROWS][COLUMNS];
```

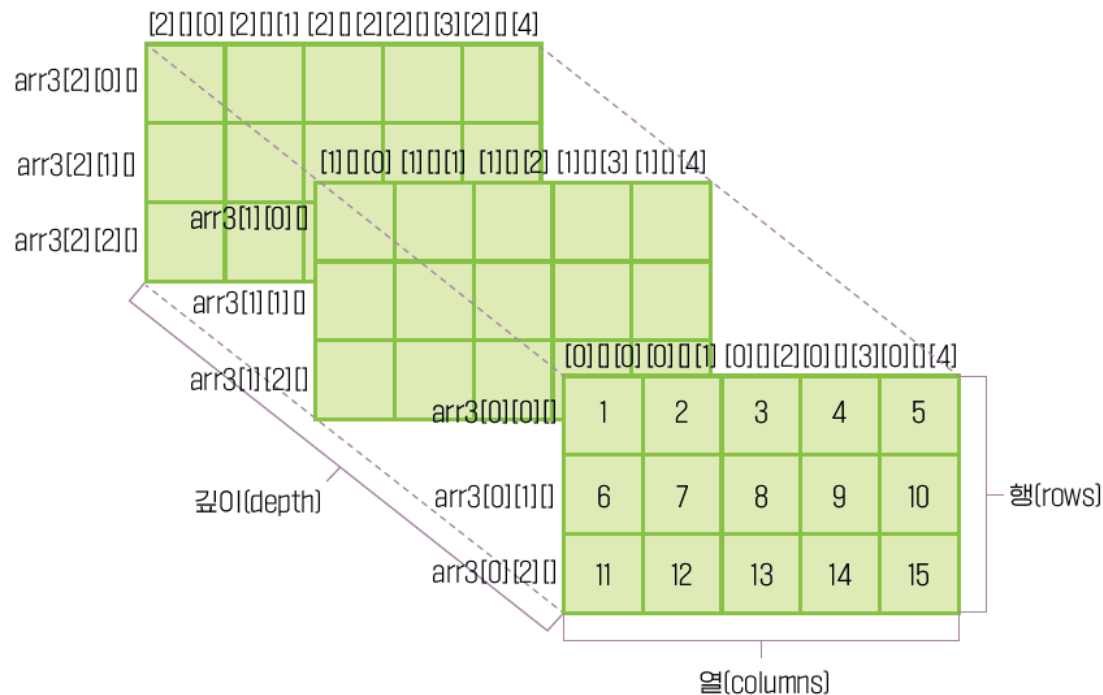


그림 7-20 3차원 배열의 개념도

4. 다차원 배열

■ 3차원 배열

- 메모리 저장 구조

```
int arr3[3][3][5];
```

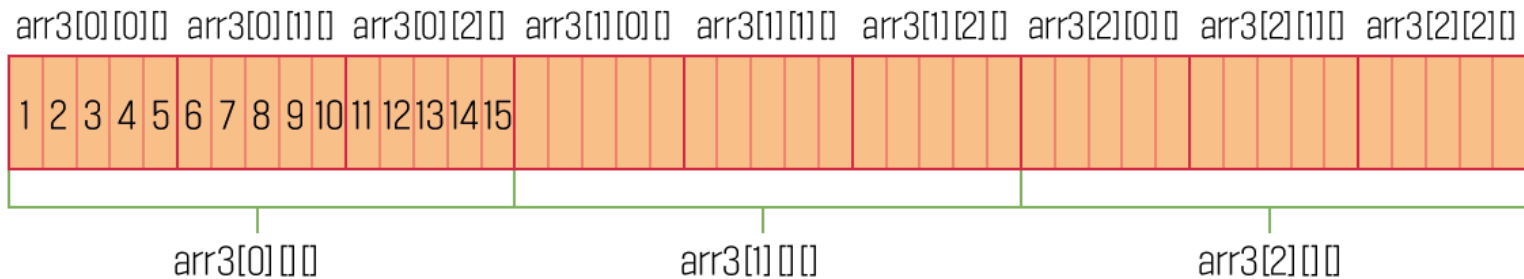


그림 7-21 3차원 배열의 메모리 저장 구조

4. 다차원 배열

■ 3차원 배열

- 3차원 배열을 선언한 후 1~45까지의 값으로 채우고 일부를 화면 출력하는 프로그램

코드 7-43 Init3DArr.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define DEPTH 3
5  #define ROWS 3
6  #define COLUMNS 5
7
8  int main(void)
9  {
10     int arr3[DEPTH][ROWS][COLUMNS];
11     for (int depth = 0; depth < DEPTH; depth++) {
12         for (int row = 0; row < ROWS; row++) {
13             for (int column = 0; column < COLUMNS; column++) {
14                 arr3[depth][row][column] = depth * ROWS * COLUMNS + row *
15                 COLUMNS + column + 1;
16             }
17         }
18     }
19     printf("arr3[0][0][1] = %d, arr3[1][0][4] = %d\n", arr3[0][0][1],
20           arr3[1][0][4]);
21     return 0;
22 }
```

간단한 계산을 통해 1~45까지의 값 지정

3차원 배열의 모든 요소를 사용하려면 3번 중첩되는 반복문 사용

〈실행 결과〉

arr3[0][0][1] = 2, arr3[1][0][4] = 20

4. 다차원 배열

■ 3차원 배열

- 3차원 배열 초기화

```
int arr3[3][3][5] = { { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 }, { 11, 12, 13, 14, 15 } },  
                      { { 16, 17, 18, 19, 20 }, { 21, 22, 23, 24, 25 }, { 26, 27, 28, 29, 30 } },  
                      { { 31, 32, 33, 34, 35 }, { 36, 37, 38, 39, 40 }, { 41, 42, 43, 44, 45 } } };
```

- 배열의 크기 지정

```
int arr3[][3][5] = { { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 }, { 11, 12, 13, 14, 15 } },  
                    { { 16, 17, 18, 19, 20 }, { 21, 22, 23, 24, 25 }, { 26, 27, 28, 29, 30 } },  
                    { { 31, 32, 33, 34, 35 }, { 36, 37, 38, 39, 40 }, { 41, 42, 43, 44, 45 } } };
```

4. 다차원 배열

■ 3차원 배열

- 3차원 배열 출력

<실행 결과>

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15

16 17 18 19 20
21 22 23 24 25
26 27 28 29 30

31 32 33 34 35
36 37 38 39 40
41 42 43 44 45
```

코드 7-44 Print3DArr.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define DEPTH 3
5  #define ROWS 3
6  #define COLUMNS 5
7
8  int main(void)
9  {
10     int arr3[][ROWS][COLUMNS] =
11         { { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 }, { 11, 12, 13, 14, 15 } },
12           { { 16, 17, 18, 19, 20 }, { 21, 22, 23, 24, 25 }, { 26, 27, 28, 29, 30 } },
13           { { 31, 32, 33, 34, 35 }, { 36, 37, 38, 39, 40 }, { 41, 42, 43, 44, 45 } } };
14     for (int depth = 0; depth < DEPTH; depth++) { // arr3[]
15         for (int row = 0; row < ROWS; row++) {
16             for (int column = 0; column < COLUMNS; column++) {
17                 printf("%d ", arr3[depth][row][column]);
18             }
19             printf("\n");
20         }
21         printf("\n");
22     }
23     return 0;
24 }
```

arr3[depth][row][0]-arr3[depth][row][4]까지 출력

1차원 배열 출력이 끝나면 줄바꿈

2차원 배열의 출력이 끝나면 줄바꿈을 한 번 더 진행

※ 함수에 3차원 배열을 전달하는 코드

➔ [코드 7-45](#)

➔ [실행 결과](#)

4. 다차원 배열

■ 3차원 배열

- 3차원 배열도 1차원 배열처럼 초기화 가능

```
int arr3[3][3][5] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,  
18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,  
39, 40, 41, 42, 43, 44, 45 };
```

05

C 언어의 다차원 배열

5. C 언어의 다차원 배열

- C 언어의 배열은 void를 제외한 어떤 종류의 자료형도 포함 가능
- 배열을 요소로 포함하는 것도 가능
- 2차원 배열은 1차원 배열을 요소로 포함하는 배열
- 3차원 배열은 2차원 배열을 요소로 포함하는 배열
- 1차원 배열은 요소들을 나열해 놓은 것
- 2차원 배열은 1차원 배열을 연속적으로 배치한 것
- 3차원 배열은 비슷하게 2차원 배열을 연속적으로 나열한 것
- C 언어의 N차원 배열은 N - 1차원 부분 배열의 배열에 불과