

# 윤성우의 열혈 C 프로그래밍



윤성우 저 열혈강의 C 프로그래밍 개정판

Chapter 11. 1차원 배열

# 윤성우의 열혈 C 프로그래밍



Chapter 11-1. 배열의 이해와 배열의  
선언 및 초기화 방법

윤성우 저 열혈강의 C 프로그래밍 개정판

# 배열이란 무엇인가?

```
int main(void)
{
    int floor101, floor102, floor103, floor104;    // 1층 101호부터 104호까지
    int floor201, floor202, floor203, floor204;    // 2층 201호부터 204호까지
    int floor301, floor302, floor303, floor304;    // 3층 301호부터 304호까지
    . . . . .
}
```

다수의 정보를 저장하기  
위해서는 다수의 배열을  
선언해야 한다.

위와 같이 다수의 변수를 선언해야 하는 경우 매우 번거로울 수 있다. 그래서 **다수의 변수선언을 용이하게 하기 위해서 배열이라는 것이 제공된다.** 배열을 이용하면 하나의 선언을 통해서 둘 이상의 변수를 선언할 수 있다.

배열은 단순히 다수의 변수선언을 대신하지 않는다. **다수의 변수로는 할 수 없는 일을 배열을 선언하면 할 수 있다.**

배열은 1차원의 형태로도 2차원의 형태로도 선언할 수 있다. 이번 Chapter에서는 1차원 형태의 배열에 대해서 학습한다.

# 1차원 배열 선언에 필요한 것 세 가지

```
int oneDimArr [4];
```

1차원 배열 선언의 예

int	배열을 이루는 요소(변수)의 자료형
oneDimArr	배열의 이름
[4]	배열의 길이



생성되는 배열의 형태

```
int arr1[7];           // 길이가 7인 int형 1차원 배열 arr1
float arr2[10];        // 길이가 10인 float형 1차원 배열 arr2
double arr3[12];       // 길이가 12인 double형 1차원 배열 arr3
```

다양한 배열 선언의 예

# 선언된 1차원 배열의 접근

```
arr[0]=10;    // 배열 arr의 첫 번째 요소에 10을 저장해라!
arr[1]=12;    // 배열 arr의 두 번째 요소에 12를 저장해라!
arr[2]=25;    // 배열 arr의 세 번째 요소에 25를 저장해라!
```

1차원 배열 접근의 예



일반화

arr[idx]=20; → "배열 arr의 idx+1번째 요소에 20을 저장해라!"

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

    arr[0]=10, arr[1]=20, arr[2]=30, arr[3]=40, arr[4]=50;

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

    printf("배열요소에 저장된 값의 합: %d \n", sum);
    return 0;
}
```

원편의 예제를 통해서 느낄 수  
있는 배열의 또 다른 매력은?

실행결과

배열요소에 저장된 값의 합: 150

# 배열! 선언과 동시에 초기화하기

```
int arr1[5]={1, 2, 3, 4, 5};
```

초기화 리스트로 초기화



초기화 결과



순서대로 초기화

```
int arr3[5]={1, 2};
```

초기화 값 부족한 경우



부족한 부분 0으로 채워짐



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

초기화 리스트는 존재하고 배열의  
길이정보 생략된 경우



컴파일러가 배열의 길이정보 채움

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

# 1차원 배열의 선언, 초기화 및 접근 관련 예제

```
int main(void)
{
    int arr1[5]={1, 2, 3, 4, 5};
    int arr2[ ]={1, 2, 3, 4, 5, 6, 7};
    int arr3[5]={1, 2};
    int ar1Len, ar2Len, ar3Len, i;

    printf("배열 arr1의 크기: %d \n", sizeof(arr1));
    printf("배열 arr2의 크기: %d \n", sizeof(arr2));
    printf("배열 arr3의 크기: %d \n", sizeof(arr3));

    ar1Len = sizeof(arr1) / sizeof(int); // 배열 arr1의 길이 계산
    ar2Len = sizeof(arr2) / sizeof(int); // 배열 arr2의 길이 계산
    ar3Len = sizeof(arr3) / sizeof(int); // 배열 arr3의 길이 계산

    for(i=0; i<ar1Len; i++)
        printf("%d ", arr1[i]);
    printf("\n");

    for(i=0; i<ar2Len; i++)
        printf("%d ", arr2[i]);
    printf("\n");

    for(i=0; i<ar3Len; i++)
        printf("%d ", arr3[i]);
    printf("\n");

    return 0;
}
```

*sizeof 연산의 결과로  
배열의 바이트 크기 정보 반환*

*배열의 길이를 계산하는 방식  
에 주목!*

*배열이기에 for문을 통한 순차  
적 접근이 가능하다.*

*다수의 변수라면 반복문을 통한  
순차적 접근 불가능!*

*실행결과*

```
배열 arr1의 크기: 20
배열 arr2의 크기: 28
배열 arr3의 크기: 20
1 2 3 4 5
1 2 3 4 5 6 7
1 2 0 0 0
```

# 윤성우의 열혈 C 프로그래밍



Chapter 11-2. 배열을 이용한 문자열  
변수의 표현

윤성우 저 열혈강의 C 프로그래밍 개정판



# char형 배열의 문자열 저장과 널 문자

```
char str[14]="Good morning!";
```

배열에 문자열 저장



저장결과



문자열의 끝에 널 문자라 불리는 \0가 삽입되었음에  
주목! 널 문자는 문자열의 끝을 의미한다.

```
int main(void)
{
    char str[]="Good morning!";
    printf("배열 str의 크기: %d \n", sizeof(str));
    printf("널 문자 문자형 출력: %c \n", str[13]);
    printf("널 문자 정수형 출력: %d \n", str[13]);

    str[12]='?'; // 배열 str에 저장된 문자열 데이터는 변경 가능!
    printf("문자열 출력: %s \n", str);
    return 0;
}
```

실행결과

배열 str의 크기: 14  
 널 문자 문자형 출력:  
 널 문자 정수형 출력: 0  
 문자열 출력: Good morning?

# 널 문자와 공백 문자의 비교

```
int main(void)
{
    char nu = '\0';    // 널 문자 저장
    char sp = ' ';     // 공백 문자 저장
    printf("%d %d", nu, sp);    // 0과 32 출력
    return 0;
}
```

널 문자를 %c를 이용해서 출력 시 아무것도 출력되지 않는다. 그렇다고 해서 널 문자가 공백 문자는 아니다.

널 문자의 아스키 코드 값은 0이고, 공백 문자의 아스키 코드 값은 32이다.

널 문자는 모니터 출력에서 의미를 갖지 않는다. 그래서 아무것도 출력이 되지 않을 뿐이다.



# scanf 함수를 이용한 문자열의 입력

```
int main(void)
{
    char str[50];
    int idx=0;

    printf("문자열 입력: ");
    scanf("%s", str); // 문자열을 입력 받아서 배열 str에 저장!
    printf("입력 받은 문자열: %s \n", str);

    printf("문자 단위 출력: ");
    while(str[idx] != '\0')
    {
        printf("%c", str[idx]);
        idx++;
    }
    printf("\n");
    return 0;
}
```

*scanf* 함수의 호출을 통해서 입력 받은  
문자열의 끝에도 널 문자가 존재함을 확  
인하기 위한 문장

*scanf* 함수를 이용해서 문자열 입력 시  
서식문자 %s를 사용한다.

***scanf("%s", str);***

위와 같이 배열이름 *str*의 앞에는  
& 연산자를 붙이지 않는다.

실행결과

문자열 입력: Simple  
입력 받은 문자열: Simple  
문자 단위 출력: Simple

```
char arr1[ ] = {'H', 'i', '~'};
char arr2[ ] = {'H', 'i', '~', '\0'};
```

*arr1*은 문자열이 아닌 문자 배열, 반면 *arr2*는 문자열!  
널 문자의 존재여부는 문자열의 판단여부가 된다.

# 문자열의 끝에 널 문자가 필요한 이유

문자열의 시작은 판단할 수 있어도 문자열의 끝은 판단이 불가능하다! 때문에 문자열의 끝을 판단할 수 있도록 널 문자가 삽입이 된다.

```
int main(void)
{
    char str[50]="I like C programming";
    printf("string: %s \n", str);

    str[8]='\0';    // 9번째 요소에 널 문자 저장
    printf("string: %s \n", str);

    str[6]='\0';    // 7번째 요소에 널 문자 저장
    printf("string: %s \n", str);

    str[1]='\0';    // 2번째 요소에 널 문자 저장
    printf("string: %s \n", str);
    return 0;
}
```

배열의 시작위치에 문자열이 저장되기 시작한다. 따라서 시작위치는 확인이 가능하다. 하지만 배열의 끝이 문자열의 끝은 아니므로 널 문자가 삽입되지 않으면 문자열의 끝은 확인이 불가능하다.

## 실행결과

```
string: I like C programming
string: I like C
string: I like
string: I
```

위 예제에서 보이듯이 printf 함수도 배열 str의 시작위치를 기준으로해서 널 문자를 만날 때까지 출력을 진행한다. 따라서 널 문자가 없으면 printf 함수도 문자열의 끝을 알지 못한다.

# scanf 함수의 문자열 입력 특성

```
int main(void)
{
    char str[50];
    int idx=0;

    printf("문자열 입력: ");
    scanf("%s", str); // 문자열을 입력 받아서 배열 str에 저장!
    printf("입력 받은 문자열: %s \n", str);

    printf("문자 단위 출력: ");
    while(str[idx] != '\0')
    {
        printf("%c", str[idx]);
        idx++;
    }
    printf("\n");
    return 0;
}
```

앞서 보인 원편의 예제를 실행할 때 다음과 같이 문자열을 입력하면

**He is my friend**

다음의 실행결과를 보인다.

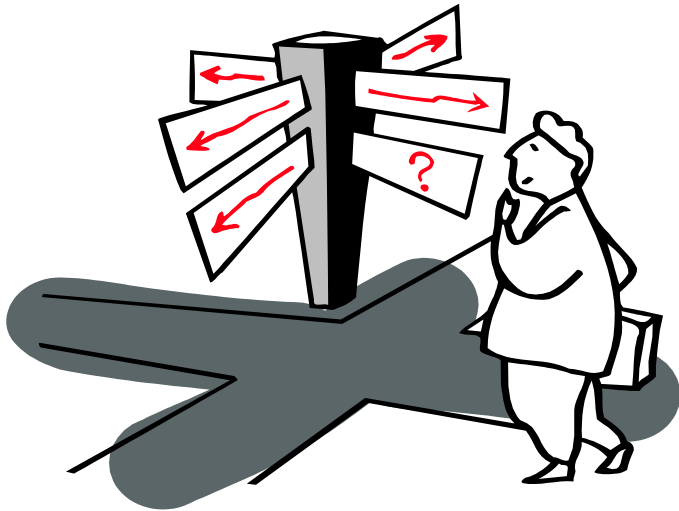
**입력 받은 문자열: He**

**문자 단위 출력: He**

scanf 함수는 공백을 기준으로 데이터의 수를 구분한다. 따라서 공백을 포함하는 문자열을 한번의 scanf 함수호출을 통해서 읽어 들이지는 못한다.

공백을 포함하는 문자열의 입력에 사용되는 함수는 이후에 별도로 설명합니다.





Chapter 11이 끝났습니다. 질문 있으신지요?