작성자 : 김기희 (cupid072@naver.com)

배열

- 학습목표
  - 배열의 특징을 설명할 수 있다.
  - 배열 사용의 장단점에 대해 설명할 수 있다.
  - 1차원 배열을 선언하고 올바른 초기화 방법에 대하여 설명할 수 있다.
  - 1차원 배열 원소의 참조방법을 설명할 수 있다.
  - 1차원 배열 원소의 수를 구할 수 있다.
  - 문자배열과 문자열의 차이점을 설명할 수 있다.
  - 배열 기반의 문자열과 포인터 기반의 문자열을 설명할 수 있다.
  - 2차원 배열을 선언하고 올바른 방법에 대하여 설명할 수 있다.
  - 2차원 배열 원소의 참조 방법을 설명할 수 있다.
  - 2차원 배열을 구성하는 행의 수와 열의 수를 구할 수 있다.

배열의 기본 개념

- 배열(Array)이란? 동일한 자료형의 데이터를 연속적으로 저장할 수 있는 저장소

  int a[10]; // int 형 데이터 10개를 저장할 수 있는 저장소

- 배열의 선언 형식

  자료형 배열명[원소의수];
  자료형 배열명[] = { 초기화목록 }; // Initialization List에 의한 할당

- 배열의 특징
  - 메모리상에 연속적으로 할당된다.
  - 배열을 구성하는 요소는 인덱스(Zero-base)로 구분한다.

배열의 선언과 초기화

- 배열의 선언
  - 배열의 선언은 배열 공간의 할당을 의미한다.

```
int a[5];                    // 선언 시점에 메모리 공간이 할당된다.
int a[5] = { 1, 2, 3, 4, 5 }; // 배열 원소의 수는 초기화 목록의 수 만큼 할당된다.
```

- 배열의 초기화
  - stack에 할당(함수 내에서 선언)되는 배열은 자동으로 초기화 되지 않는다.
  - data영역에 할당(전역으로 선언)되는 배열은 자동으로 초기화 된다.

배열의 선언과 초기화

- 10개의 공간을 할당한 후 모든 배열 원소를 0으로 초기화

  int a[5] = { 0 };

- 10개의 공간을 할당한 후 첫 번째 원소는 1로 나머지는 0으로 초기화

  int a[5] = { 1 }; // 1 0 0 0 0

- 10개의 공간을 할당한 후 첫 번째 원소를 1로 마지막 원소를 10으로 초기화 (선택적 요소 초기화)

  int a[5] = { [0] = 1, [4] = 5 }; // 1 0 0 0 5

- 선택적 요소 초기화 이 후의 값은 선택적 요소 이 후의 값이 된다.

  int a[5] = { [1] = 2, 3, 4, 5 }; // 0 2 3 4 5

※ 선택적 요소 초기화를 사용할 경우 모든 요소에 대한 값을 지정하거나 첫 번째 요소에 대한 값 만을 지정하도록 한다.

배열의 참조

- 배열의 원소는 배열 참조 연산자[]를 사용하여 참조한다.

```
int ary[] = { 10, 20, 30, 40, 50 };
printf("%d\n", ary[2]);
```

ary				
ary[0]	ary[1]	ary[2]	ary[3]	ary[4]
10	20	30	40	50

- 배열 원소의 수 구하기

  배열 원소의 수는 배열의 전체 크기(Byte) / 배열을 구성하는 원소 하나의 크기(Byte)

```
size_t size = sizeof(ary)/sizeof(ary[0])
```

배열의 참조

- 반복문을 사용한 배열의 참조

```
int ary[] = {  5,  1, 10,  4,  9,  6,  3,  7,  8,  2 };
size_t size = sizeof(ary) / sizeof(ary[0]);
for (size_t i=0 ; i<size ; i++) {
   printf("ary[%d] : %d\n", i, ary[i]);
}
```

- size_t 타입은 컴파일러에 따라 다르다.
  - Visual Studio : unsigned int
  - gcc 64bit : unsigned long long

문자배열과 문자열

- char형 배열은 문자들을 연속적인 공간에 저장할 수 있다.

```
char name[5] = { 'K', 'I', 'H', 'E', 'E' };
char name[6] = { 'K', 'I', 'H', 'E', 'E' };
```

K	I	H	E	E

K	I	H	E	E	\0

- 그러므로 char형 배열은 주로 문자열을 저장하는 용도로 사용되며 문자열로 초기화 할 수 있다.

```
char name[10] = "KIHEE";
char name[] = "KIHEE";
```

K	I	H	E	E	\0	\0	\0	\0	\0

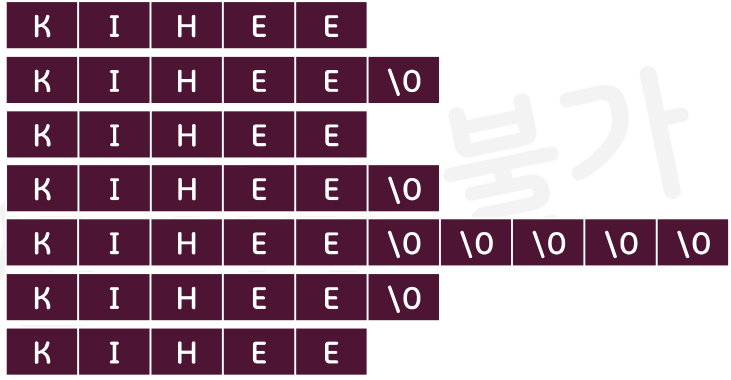
K	I	H	E	E	\0

※ 문자열을 사용하여 문자배열을 초기화 시 배열의 크기는 최소 문자의 수 + 1 이상의 크기이어야 한다.

문자배열과 문자열

- 문자 배열을 문자열로 사용하기 위해서는 반드시 문자 배열의 끝에 널이 포함되어야 한다.

```
char name[5] = { 'K', 'I', 'H', 'E', 'E' };
char name[6] = { 'K', 'I', 'H', 'E', 'E' };
char name[] = { 'K', 'I', 'H', 'E', 'E' };
char name[6] = { 'K', 'I', 'H', 'E', 'E', '\0' };
char name[10] = "KIHEE";
char name[] = "KIHEE";
char name[5] = "KIHEE";
```



└ 컴파일러에 따라 오류 또는 정상처리가 됨

※ 마지막에 널 문자를 포함하지 않는 문자 배열은 문자열로 취급되어서는 안된다.

문자열을 사용한 문자 배열의 초기화

- 문자 배열은 stack 영역에 할당되며 초기화에 사용되는 문자열은 CONST 영역에 저장되어 있다.

  char 배열명[] = 문자열;

  char name[] = "KIHEE";

- 배열 초기화에 사용되는 문자열은 프로그램 구동 시 CONST 영역에 저장된다.

- 배열은 프로그램 실행 중(Runtime)에 배열 공간을 할당 받은 후 CONST 영역에 저장되어 있는 문자열을 Byte by Byte 복사를 통해 배열을 초기화 한다.

- CONST 영역의 데이터를 문자 배열에 복사한 후 여분의 배열 공간은 0 으로 초기화 한다.

※ 0 (integer const zero)은 문자로 표현 시 '\0', 주소로 표현 시 NULL이다.

문자열을 사용한 문자 배열의 초기화

```
char name[] = "KIHEE";
```

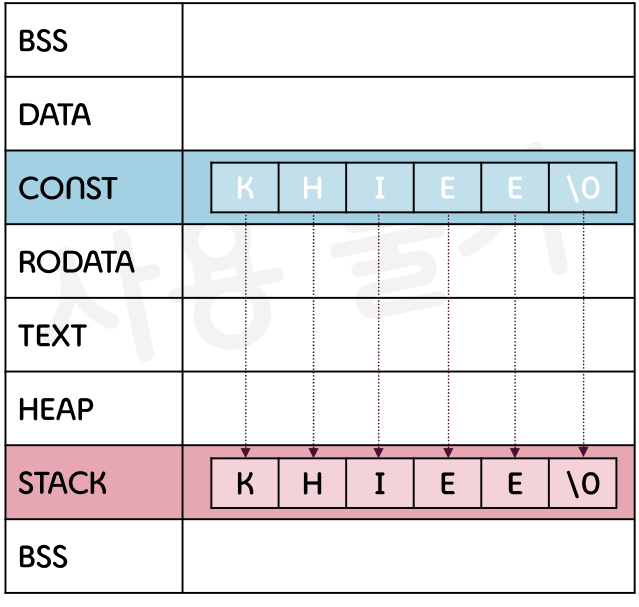
- 문자열의 끝에 NULL 문자가 없으면?

```
char name[5] = { 'K', 'I', 'H', 'E', 'E' };
printf("%s\n", name);
```

- NULL 문자를 만날 때 까지 출력

K	I	H	E	E	?	?	?	.....	\0

- 문자열 복사 함수의 경우 할당된 영역을 벗어난 위치에 데이터를 복사



문자 배열과 문자열 정리

- 배열은 곧 그 배열 공간의 시작 주소 상수이다. (배열의 자료형에 상관 없이)

```
char name[] = "KIHEE";          // OK
name = "KOREA";                 // ERROR
```

- 문자열은 곧 그 문자열의 시작 주소 상수이다.

- 문자 배열은 문자열을 사용하여 초기화가 가능하다.

```
char name[] = "KIHEE";          // OK
name = "KOREA";                 // ERROR
```

- 문자열을 사용하여 배열을 초기화 하는 경우 배열의 크기는 최소 문자열을 구성하는 문자의 수 + 1 만큼의 공간을 가져야 한다.

```
char name[6] = "KIHEE";
char name[5] = "KIHEE";         // 컴파일러에 따라 에러 또는 정상처리
```

- 마지막에 널('\0') 문자를 가지지 않는 문자 배열은 문자열로 취급되어서는 안 된다.

1차원 배열을 함수에 전달하기

```
#include <stdio.h>

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

int main() {
	int ary[] = { 5, 2, 8, 4, 10, 9, 1, 7, 6, 3 };
	int size = (int)(sizeof(ary) / sizeof(ary[0]));
	print(ary, size);

	return 0;
}
```

- 배열을 함수의 인수로 전달 시 타입은 배열의 타입과 동일한 타입을 사용한다.
- 배열타입 파라미터 원소의 수는 기술을 생략할 수 있다.
- 배열타입 파라미터에 원소의 수를 기술하여도 무시된다.
- 배열타입 파라미터는 원소의 수나 배열의 차원에 관계 없이 배열의 시작 주소가 저장되는 포인터 타입이다.
- 배열타입 파라미터는 포인터이므로 sizeof 연산을 통한 배열 원소의 수를 얻어 낼 수 없다.
- 그러므로 1차원 배열을 함수 인수로 전달 시에는 배열 원소의 수를 함께 전달하여야 한다.

다차원 배열의 선언과 초기화

- 2차원 배열의 선언

  자료형 배열명[행의수][열의수];

  int ary[2][3];

ary					
ary[0]			ary[1]		
ary[0][0]	ary[0][1]	ary[0][2]	ary[1][0]	ary[1][1]	ary[1][2]

- 2차원 배열의 초기화는 초기화 리스트의 시작과 끝을 나타내는 중괄호{}를 사용한다.
- 2차원 배열을 구성하는 각 행은 행의 시작과 끝을 나타내는 중괄호{}를 사용한다.
- 행의 초기화 시작과 끝 괄호는 생략이 가능하다.
- 문자열을 저장하기 위한 배열의 열의 크기는 문자열 배열을 초기화 하기 위한 문자열 중 가장 긴 문자열의 수 + 1개 만큼의 크기로 한다.

다차원 배열의 선언과 초기화

- 선언과 동시에 모든 요소를 0으로 초기화

  int ary[2][3] = { 0 };

- 초기화 리스트를 사용하는 경우 행의 수 생략이 가능하다.

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

- 선택적 요소 초기화

  int ary[][3] = { [0] = { 0 }, [1] = { 1, 2, 3} } ;

- 아래의 형식은 불가능한 것은 아니지만 컴파일러나 컴파일러 버전에 따라 경고 메시지가 출력된다.

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

2차원 배열의 참조

- 2차원 배열의 참조는 배열 연산자[]를 사용하여 행과 열의 index를 지정하여 참조한다.

  배열명[행][열]

  ```
  int ary[2][3] = { { 10, 20, 30 }, { 40, 50, 60 } };
  printf("%d\n", ary[1][1]);
  ```

ary					
ary[0]			ary[1]		
ary[0][0]	ary[0][1]	ary[0][2]	ary[1][0]	ary[1][1]	ary[1][2]
10	20	30	40	50	60

- 2차원 배열의 행과 열의 수 구하기

  행의 수 : 2차원 배열의 전체 크기 / 2차원 배열을 구성하는 행의 크기

  sizeof(ary) / sizeof(ary[0])

  열의 수 : 2차원 배열을 구성하는 행의 크기 / 2차원 배열의 행을 구성하는 원소 하나의 크기

  sizeof(ary[0]) / sizeof(ary[0][0]);

2차원 배열의 참조

- 반복문을 사용한 배열의 참조

```
int ary[][3] = { { 10, 20, 30 }, { 40, 50, 60 } };
size_t row = sizeof(ary)/sizeof(ary[0]);                    // 행의 수 구하기
size_t column = sizeof(ary[0])/sizeof(ary[0][1]);           // 열의 수 구하기

for (int i=0 ; i<row ; i++) {
   for (int j=0 ; j<column ; j++) {
      printf("ary[%d][%d] : %d\n", i, j, ary[i][j]);
   }
}
```

2차원 배열을 함수에 전달하기

```
#include <stdio.h>

void print(int arr[][3], int row) {
	for (int i=0 ; i<row ; i++) {
		int col = (int) (sizeof(arr[i]) / sizeof (arr[i][0]));
		for (int j=0 ; j<col ; j++)
			printf("arr[%d][%d] : %d\n", i, j, arr[i][j]);
	}
}

int main() {
	int ary[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
	int row = (int)(sizeof(ary) / sizeof(ary[0]));
	print(ary, row);

	return 0;
}
```

- 2차원 배열을 함수에 전달하고자 하는 경우 함수 파라미터의 타입은 2차원 배열과 동일한 타입이어야 한다.
- 이때 행의 수는 생략이 가능하지만 하나의 행을 구성하는 열의 수는 생략할 수 없다.
- 행의 수는 기술하여도 무시된다.
- 1차원 배열과 마찬가지로 2차원 배열 타입의 함수 파라미터 또한 배열의 시작 주소가 저장되는 포인터이다.
- 배열타입 파라미터는 포인터이므로 sizeof 연산을 통한 배열 원소의 수를 구할 수 없다.
- 그러므로 2차원 배열을 함수에 전달시에는 2차원 배열을 구성하는 행의 수를 함께 전달해야 한다.

2차원 문자배열

- 2차원 문자 배열의 선언

  - 형식1

  char name[2][8] = { {'M', 'r', ' ', 'K', 'I', 'M'}, {'M', 'r', 's', ' ', 'K', 'I', 'M'} };

  - 형식2 : 문자열을 사용한 초기화

  char name[2][8] = {"Mr KIM", "Mrs KIM"};

  2차원 문자 배열은 1차원 문자 배열을 원소로 하는 배열이며, 문자열로 구성된 배열이다.

  2차원 문자배열을 구성하는 원소는 문자열을 사용하여 초기화가 가능하다.

  문자열을 사용하여 초기화하는 경우 각 원소의 크기는 초기화에 사용되는 문자열 중 최대 길이의 문자열의 문자의 수 + 1의 크기이어야 한다.

2차원 문자배열의 사용

- 2차원 문자배열(문자열 배열)

```
void printMenu(char menuStr[][13], int count) {
        ......
        for (int i=0 ; i<count ; i++) {
                printf("%d. %s\n", i+1, menuStr[i]);
        }
        ......
}


......
char menu[][13] = { "입력하기", "출력하기", "종료하기" };
int menuCount = (int)sizeof(menu)/sizeof(menu[0]);
printMenu(menu, menuCount);
```

menuStr의 타입은 char[][13]

menuStr[] 은 menuStr이 배열임을 나타낸다.

menuStr[] 의 각 원소의 타입은 char[3]이다.

menuStr은 배열의 주소를 저장하기 위한 변수이다.

3차원 배열의 선언과 초기화

- 3차원 배열은 2차원 배열을 원소로 하는 배열이다.

- int ary[][2][3] = {
  - { { 1, 2, 3 }, {4, 5, 6 } },
  - { { 10, 20, 30}, { 40, 50, 60 } }
- };

ary[] 는 int[2][3] 타입을 원소로 하는 배열이다.
ary는 3차원 배열
ary[0]은 2차원 배열
ary[0][0]은 1차원 배열
ary[0][0][0]은 0차원으로 값

- 면의 수 : sizeof(ary)/sizeof(ary[0]);
- 행의 수 : sizeof(ary[0]) / sizeof(ary[0][0]);
- 열의 수 : sizeof(ary[0][0]) / sizeof(ary[0][0][0]);