Kumoh National Institute of Technology

C Programming

Ch.7-2 포인터와 배열

컴퓨터공학과

C프로그래밍

Contents

1. (Review) 배열, 포인터 (추가)
2. 포인터와 배열의 관계
3. 포인터 연산
4. 상수 형태의 문자열을 가리키는 포인터
5. 포인터 변수로 이뤄진 배열 : 포인터 배열

- 추가 : 교재 외 추가 자료

(Review) 배열

- 문제 : 10개의 int 원소를 가진 1차원 배열을 만들고 각 원소의 값으로 0~100 사이의 난수를 대입한 후 각 원소의 값을 출력해 보라.

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

int main(void)
{
    srand(time(NULL));

    int ary[10];

    for (int i = 0; i < 10; i++)
        ary[i] = rand() % 101;

    for (int i = 0; i < 10; i++)
        printf("[%d] %d\n", i, ary[i]);
}
```

```
[0] 68
[1] 19
[2] 51
[3] 2
[4] 99
[5] 63
[6] 63
[7] 1
[8] 58
[9] 72
```

2차원 배열, 3차원 배열, ... 도 다 알고 있다!

(Review) 포인터

- 문제 : int num = 3;이 있다. 포인터 변수 p를 만들어 num을 가리킨 후(num의 주소 저장), p를 통해 num의 값을 100으로 변경하고 값을 출력해 보라.

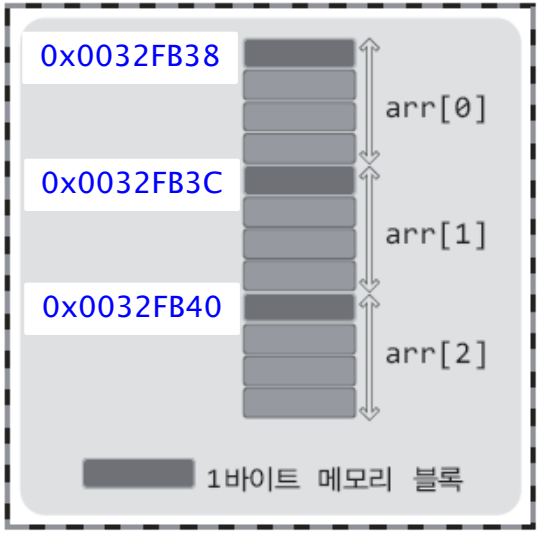
```
int main(void)
{
    int num = 3;
    int* p = &num;
    *p = 100;

    printf("num의 값은 %d\n", *p);
}
```

```
num의 값은 100
```

포인터의 시작 → 시작이 반이다! (개념 이해 중요)

배열의 이름은 무엇을 의미하는가?

- 배열 이름 : 첫 번째 요소의 주소값 = 첫 번째 요소를 가리키는 포인터
  - 단, 그 값을 바꿀 수 없는 상수 개념




```
int main(void)
{
    int arr[3] = { 0, 1, 2 };

    printf("배열의 이름 : %p \n", arr);
    printf("첫 번째 요소: %p \n", &arr[0]);
    printf("두 번째 요소: %p \n", &arr[1]);
    printf("세 번째 요소: %p \n", &arr[2]);
}
```

```
배열의 이름 : 0032FB38
첫 번째 요소: 0032FB38
두 번째 요소: 0032FB3C
세 번째 요소: 0032FB40
```

배열 이름의 진짜 정체는 자동 형변환이다!
→ 첫 번째 요소의 주소값(포인터)으로 형변환됨

배열 이름과 진짜 포인터 변수의 비교

비교조건 \ 비교대상	포인터 변수	배열 이름
이름이 존재하는가?	존재	존재
무엇을 나타내거나 저장하는가?	메모리의 주소 값	메모리의 주소 값
주소 값의 변경이 가능한가?	가능	불가능

- 배열 이름은 상수 ⇒ 값을 변경할 수 없음

```
int main(void)
{
    int a[5] = { 0, 1, 2, 3, 4 };
    int b = 10;
    a = &b;       // a는 상수이므로 오류
}
```

1차원 배열 이름의 타입 (1)

- 배열 이름도 포인터이므로 타입이 존재함
  - 배열 이름이 뭐라고? 첫 번째 원소의 주소값!
    - int ary[5]의 첫 번째 원소의 타입은 int형

      → int의 주소값의 타입은 int *
  - Examples
    - int ary1[10]; // ary1의 타입은? int *
    - double ary2[10]; // ary2의 타입은?
    - char ary3[5]; // ary3의 타입은?

1차원 배열 이름의 타입 (2)

- int ary[5];에서 ary의 타입은 int *
  - int* p와 동일하게 사용 가능

```
int main(void)
{
    int arr1[3] = { 1, 2, 3 };
    double arr2[3] = { 1.1, 2.2, 3.3 };

    printf("%d \t %f \n", *arr1, *arr2);
    *arr1 += 100;
    *arr2 += 120.5;
    printf("%d \t %f \n", arr1[0], arr2[0]);
}
```

```
1        1.100000
101      121.600000
```

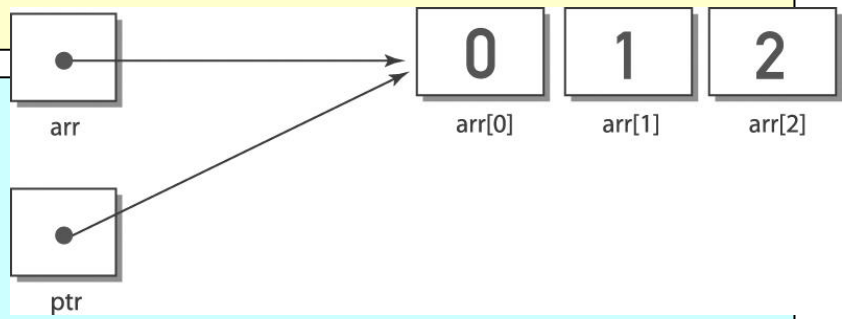
앗! *arr1과 arr1[0]이 같네.
포인터도 배열처럼 사용할 수 있을까?

배열 이름의 활용

- 배열 이름을 포인터처럼(사실 포인터) 사용 가능!
- 포인터를 배열 이름처럼 사용 가능!

```
int main(void)
{
    int arr[3] = { 15, 25, 35 };
    int* ptr = &arr[0];     // int *ptr = arr; 와 동일한 문장

    printf("%d %d \n", ptr[0], arr[0]);
    printf("%d %d \n", ptr[1], arr[1]);
    printf("%d %d \n", ptr[2], arr[2]);
    printf("%d %d \n", *ptr, *arr);
}
```

```
15 15
25 25
35 35
15 15
```



포인터 연산이란?

- 포인터 변수의 값(주소)을 증가 또는 감소시키는 연산
- 원리 : ptr = ptr + 1의 의미?
  - ptr의 값(가리키는 번지)이 1000번지라면 1001이 될까?
  - 아니다! → 해당 타입의 다음 변수를 가리키게 됨(시작 주소)
    - int* ptr; : 다음 int 변수는 4바이트 뒤에 있으므로 1004
    - char* ptr; : 다음 char 변수는 1바이트 뒤, 1001
- 포인터 연산의 예

```
ptr1++;
ptr1 += 3;
--ptr1;
ptr2 = ptr1 + 2;
```

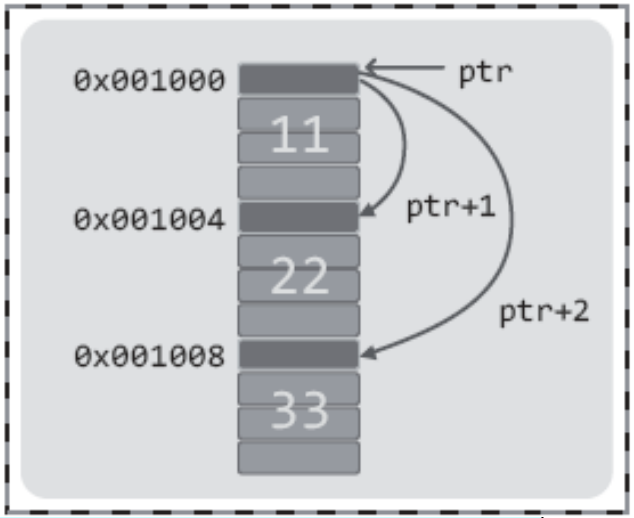
다음 포인터 연산의 실행 결과는?

```
int main(void)
{
    int* ptr1 = NULL;
    double* ptr2 = NULL;

    printf("%d %d \n", ptr1 + 1, ptr1 + 2);
    printf("%d %d \n", ptr2 + 1, ptr2 + 2);

    printf("%d %d \n", ptr1, ptr2);

    ptr1++;
    ptr2++;
    printf("%d %d \n", ptr1, ptr2);
}
```

int형 포인터 변수의 값을 1만큼 증가(또는 감소)하면, 실제로 sizeof(int)의 크기만큼 값이 증가(또는 감소)됨
double형 포인터 변수의 경우 → sizeof(double)만큼 증감함

포인터 연산으로 배열 요소에 접근

```
int main(void)
{
    int arr[3] = { 11, 22, 33 };
    int* ptr = arr;     // int * ptr=&arr[0]; 과 같은 문장

    printf("%d %d %d \n", *ptr, *(ptr + 1), *(ptr + 2));

    printf("%d ", *ptr);  ptr++;
    printf("%d ", *ptr);  ptr++;
    printf("%d ", *ptr);  ptr--;
    printf("%d ", *ptr);  ptr--;
    printf("%d ", *ptr);  printf("\n");
}
```




```
11 22 33
11 22 33 22 11
```

int형 포인터 변수가 int형 배열을 가리키면, int형 포인터 변수의 값을 1씩 증가/감소시켜 다음(이전) 요소에 순차적으로 접근 가능

포인터와 배열을 통해 얻을 수 있는 중대한 결론

- arr[i] == *(arr + i)

```
int main(void)
{
    int arr[3] = { 11, 22, 33 };
    int* ptr = arr;     // int* ptr = &arr[0]; 과 같은 문장

    printf("%d %d %d \n", *ptr, *(ptr + 1), *(ptr + 2));
    printf("%d %d %d \n", *(ptr + 0), *(ptr + 1), *(ptr + 2));
    printf("%d %d %d \n", ptr[0], ptr[1], ptr[2]);
    printf("%d %d %d \n", arr[0], arr[1], arr[2]);
    printf("%d %d %d \n", *(arr + 0), *(arr + 1), *(arr + 2));
                        // *(arr + 0)는 *arr와 동일
}
```

arr이 배열 이름이든 포인터 변수이든 결론은
arr[i] == *(arr + i)

배열은 내부적으로 포인터로 처리됨

두 가지 형태의 문자열 표현

- 배열 기반의 문자열 변수
- 포인터 기반의 문자열 상수

```
int main(void)
{
    char str1[10] = "My String";    // char str1[] = "My String";
    char *str2 = "Your String";

    printf("%s \n", str1);
    printf("%s \n", str2);

    str1[0] = 'm';
    // str2[0] = 'y';    // 에러 발생!

    printf("%s \n", str1);
    printf("%s \n", str2);
}
```


```
My String
Your String
my String
Your String
```

문자열 상수에 대한 이해(1)

```
char * str = "Const String";
```

문자열 저장 후 주소 값 반환

```
char * str = 0x1234;
```

문자열이 먼저 할당된 이후에
그 때 반환되는 주소 값이 저장되는 방식이다.

```
printf("Show your string");
```

문자열 저장 후 주소 값 반환

```
printf(0x1234);
```

위와 동일하다.
문자열은 선언 된 위치로 주소 값이 반환된다.

```
WhoAreYou("Hong");
```

문자열을 전달받는 함수의 선언

```
void WhoAreYou(char * str) { . . . . }
```

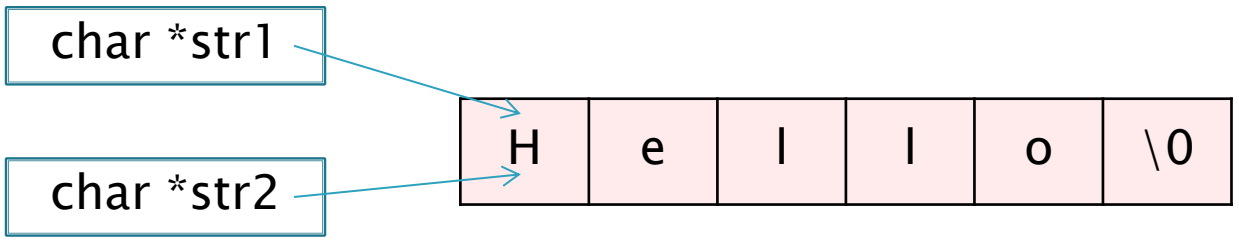
문자열의 전달만 보더라도
함수의 매개변수 형(type)을 짐작할 수 있다.

문자열 상수에 대한 이해(2)

- 동일한 문자열 상수에 대해서는 한 번만 메모리 공간 할당
  - 컴파일러마다 다를 수 있음

```
int main()
{
    char* str1 = "Hello";
    char* str2 = "Hello";

    printf("%d, %d\n", str1, str2);
}
```

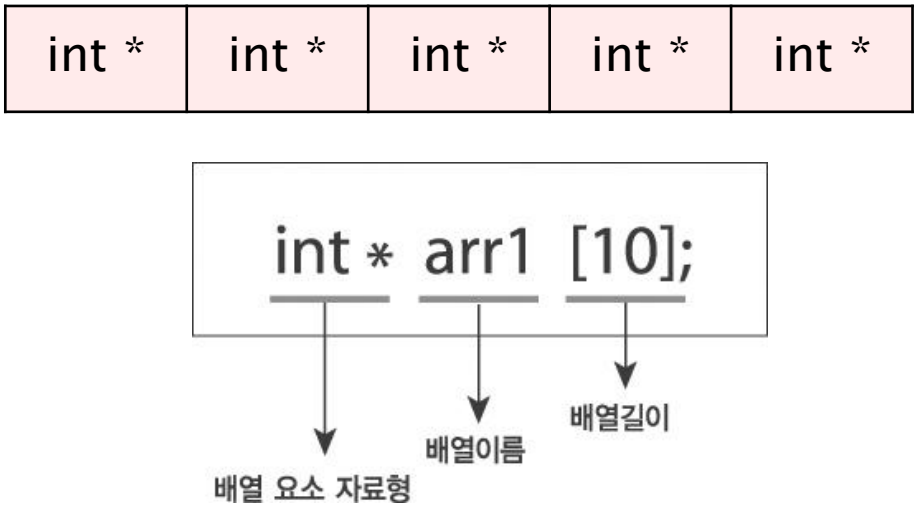
```
9263192, 9263192
```



포인터 배열의 이해

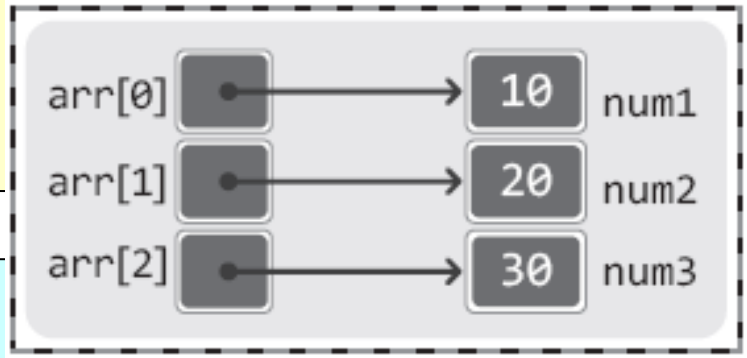
- 배열의 요소가 int 5개라면? int arr[5];
- 포인터 배열이란?
  - 배열의 요소가 포인터인 배열
  - int* arr[5] : arr[0], arr[1], arr[2], arr[3], arr[4]
    각각이 int * 타입의 변수

```
int* arr1[10];
double* arr2[20];
char* arr3[30];
```



포인터 배열 예제 (1)

```
int main(void)
{
    int num1 = 10, num2 = 20, num3 = 30;
    int* arr[3] = { &num1, &num2, &num3 };

    printf("%d \n", *arr[0]);
    printf("%d \n", *arr[1]);
    printf("%d \n", *arr[2]);
}
```



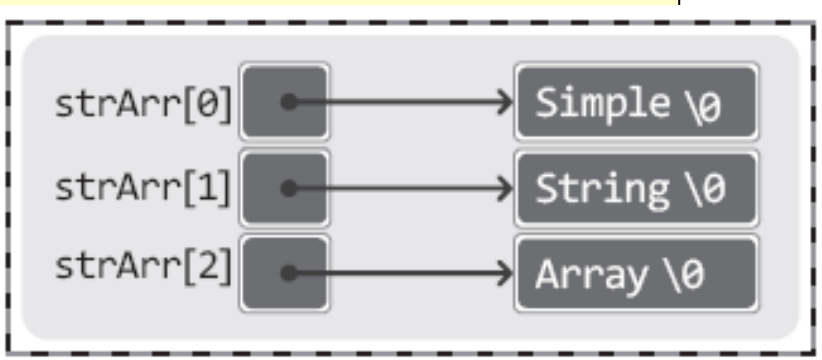

```
10
20
30
```

arr[0], arr[1], arr[2] 각각이 int * 변수이다!

포인터 배열 예제 (2)

- 문자열을 저장하는 포인터 배열

```
int main(void)
{
    char* strArr[3] = { "Simple", "String", "Array" };

    printf("%s \n", strArr[0]);
    printf("%s \n", strArr[1]);
    printf("%s \n", strArr[2]);
}
```




```
Simple
String
Array
```

점검 문제 (1)

```
int main(void)
{
    int a, b;
    int* p;

    a = b = 7;
    p = &a;

    printf("*p = %d\n", *p);

    *p = 3;
    printf("a = %d\n", a);

    p = &b;

    *p = 2 * *p - a;
    printf("b = %d\n", b);

    p = &a;
    printf("정수 입력 : ");
    scanf("%d", p);
    printf("a = %d\n", a);
}
```

- 예제 프로그램의 실행 결과 및 메모리(스택) 변화를 그림으로 표현하라.

점검 문제 (2)

- 예제 프로그램의 실행 결과는?

```
#include <stdio.h>
int main(void)
{
    int ary[] = { 1, 2, 3, 4, 5 };
    int* p;
    p = ary;

    printf("%d\n", *(p + 1));
    printf("%d\n", p[1]);
    printf("%d\n", ary[*p]);
    printf("%d\n", *(ary + ary[2]));
}
```

이번 장에서 배운 것

- 배열 이름은 첫 번째 요소의 주소값을 가지고 있다. 단, 상수 개념이므로 값을 변경할 수는 없다.
- 배열 이름의 타입은 첫 번째 요소 타입의 포인터가 된다. int arr[5]인 경우 arr의 타입은 int 포인터가 된다.
- 포인터 변수의 값을 1 증가시키면 가리키는 대상에 따라 해당 바이트만큼 값이 증가한다. int *p;인 경우 p++는 4씩 증가.
- 포인터는 배열처럼, 배열(이름)은 포인터처럼 사용 가능하다.
- 문자열 변수는 char 배열로 표현하고 문자열 상수는 char 포인터로 표현될 수 있다. char *str = "String";
- 배열의 각 요소로 포인터 변수가 들어갈 수 있다. 이를 포인터 배열이라 한다. int *arr[3];
- 본 장의 내용은 다차원 배열에도 똑 같이 적용될 수 있으므로 확실한 이해가 필요하다.