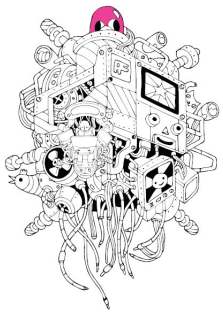


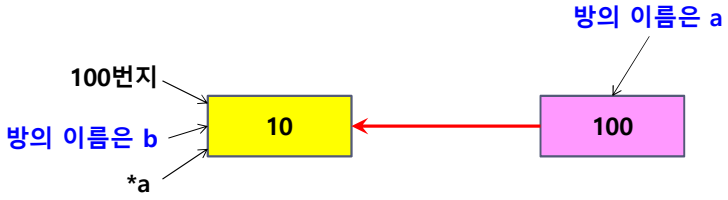
Chapter 13. 포인터와 배열!

Chapter 13-1. 포인터와 배열의 관계

Pointer 복습

- 포인터 변수 선언 : int * a;
  - 정수 포인터 변수 (방) a를 만들자.
  - a에는 어떤 정수 방의 주소가 저장될 것이다.
- 포인터 변수에 주소 저장 : a = &b;
  - int b = 10;은 미리 선언되어 있다고 가정하자.
  - 주소 방 a에 정수 방 b의 주소를 저장하라는 명령이다.
- a가 &b와 같으면, *a는 b와 완전히 동일하다.

```
int b = 10;
Int *a = &b;
```



<그림> 변수 a와 b의 관계 (ex. a == 100)

배열의 이름은 무엇을 의미하는가? (1)

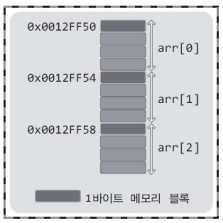
아래의 예제에서 보이듯이 배열의 이름은 배열의 시작 주소 값을 의미하는(배열의 첫 번째 요소를 가리키는) 포인터이다. 단순히 주소 값이 아닌 포인터인 이유는 메모리 접근에 사용되는 * 연산이 가능하기 때문이다.

ArrayNameType.c

```
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]);
    // arr = &arr[i];  // 이 문장은 컴파일 에러를 일으킨다.
    return 0;
}
```

실행결과

```
배열의 이름: 0012FF50
첫 번째 요소: 0012FF50
두 번째 요소: 0012FF54
세 번째 요소: 0012FF58
```



배열의 이름은 변수가 아닌 상수 형태의 포인터이기에 대입연산이 불가능하다.

배열 요소간 주소 값의 크기는 4바이트임을 알 수 있다(모든 요소가 붙어있다는 의미).

배열의 이름은 무엇을 의미하는가? (2)

- 배열 이름은 **배열의 시작 주소** 값을 의미
- 그 형태는 값의 저장이 불가능한 **상수**임

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

- 따라서 위와 같은 정수형 배열에서,"**a = &a[0]**"와 같은 연산은 불가함

- **<포인터 변수와 배열 이름의 비교>**

비교 조건	포인터 변수	배열 이름
이름이 존재하나?	존재함	존재함
무엇을 나타내거나 저장하나?	메모리의 주소 값	메모리의 주소 값
주소 값의 변경이 가능하나?	가능함	**불가능**

1차원 배열 이름의 포인터 형

1차원 배열 이름의 포인터 형 결정하는 방법
- 배열의 이름이 가리키는 변수의 자료형을 근거로 판단
- int형 변수를 가리키면 int * 형
- double형 변수를 가리키면 double * 형

int arr1[5]; 에서 arr1은 int * 형
double arr2[7]; 에서 arr2는 double * 형

ArrayNamePointerOperation.c

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

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

배열 이름을 대상으로 포인터 연산을 하고 있음에 주목!

실행결과

```
1 1.1
101 121.6
```

arr1이 int형 포인터이므로 * 연산의 결과로 4바이트 메모리 공간에 정수를 저장
arr2는 double형 포인터이므로 * 연산의 결과로 8바이트 메모리 공간에 실수를 저장

포인터를 배열의 이름처럼 사용할 수도 있다.

```
int main(void)
{
    int arr[3]={1, 2, 3};
    arr[0] += 5;
    arr[1] += 7;
    arr[2] += 9;
    . . . .
}
```

arr은 int형 포인터이니 int형 포인터를 대상으로 배열접근을 위한 [idx] 연산을 진행한 셈이다.

실제로 포인터 변수 ptr을 대상으로 ptr[0], ptr[1], ptr[2]와 같은 방식으로 메모리 공간에 접근이 가능하다.

ArrayNamePointer.c

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

포인터 변수를 이용해서 배열의 형태로 메모리 공간에 접근하고 있음에 주목!

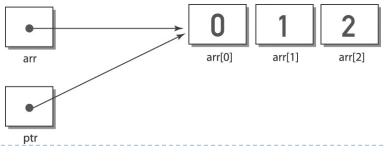
실행결과

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

배열 이름의 활용 (1/2)

- 배열 이름의 활용
  - 배열 이름을 포인터처럼, 포인터를 배열 이름처럼 활용

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

    ptr = arr; // ptr에는 arr[0]의 주소 &arr[0]이 저장됨

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



배열 이름의 활용 (2/2)

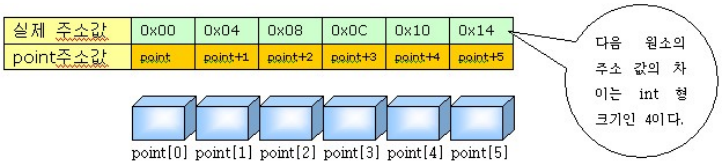
- 배열 이름의 활용
  - sizeof(배열명)은 배열 전체 크기
  - pointer 변수의 크기는 항상 8(4) 바이트 : 64비트 (32비트) OS

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

    ptr = arr; // ptr에는 arr[0]의 주소가 저장됨

    printf("%d ₩n", sizeof(arr));      // 12 출력
    printf("%d ₩n", sizeof(ptr));      // 8 (4) 출력

    printf("%d, %d, %d ₩n", ptr[0], ptr[1], ptr[2]);
    return 0;
}
```

Chapter 13-2. 포인터 연산

(배열이름 + 1)은 무엇?

- 그렇다면 point + 1은 무엇일까?
  - 배열 이름 point에 1을 더한 (point + 1)은 주소값 point의 다음 원소의 주소값
  - (point + 1)은 &point[1]와 같으며, point[1]의 주소값



  - (point + 1)은 point의 실제 주소 값에 int 형의 크기인 4만큼 더한 주소 값
  - (point + 10)은 &point[10]과 같음
  - *(point + 10)은 point[10]과 같음

```
*point == point[0]
*(point + 1) == point[1]
*(point + 2) == point[2]
```

배열에 대한 pointer 표현

- array를 위한 pointer 표현

```
*(point + i) == point[i]
```

연산식 *(point + 1)는 *point + 1 과는 다른 결과이므로 반드시 괄호를 이용해야 함

  - pointer 변수인 point의 덧셈은 단순히 i 바이트 만큼을 증가시키는 것이 아니라 point가 가리키는 변수인 int형 크기를 i 번째 만큼 증가한 주소 값을 결과로 가짐

포인터를 대상으로 하는 증가 및 감소 연산 (1/2)

PointerOperationResult.c

```
int main(void)
{
    int * ptr1=0x0010;
    double * ptr2=0x0010;
    printf("%p %p \n", ptr1+1, ptr1+2);
    printf("%p %p \n", ptr2+1, ptr2+2);
    printf("%p %p \n", ptr1, ptr2);
    ptr1++;
    ptr2++;
    printf("%p %p \n", ptr1, ptr2);
    return 0;
}
```

적절치 않은 초기화

왼편과 같이 포인터 변수에 저장된 값을 대상으로 하는 증가 및 감소연산을 진행할 수 있다(곱셈, 나눗셈 등은 불가).
그리고 이것도 포인터 연산의 일종이다.

실행결과

```
00000014 00000018
00000018 00000020
00000010 00000010
00000014 00000018
```

예제의 실행결과를 통해서 다음 사실을 알 수 있다.

- int형 포인터 변수 대상의 증가 감소 연산 시 sizeof(int)의 크기만큼 값이 증가 및 감소
- double형 포인터 변수 대상의 증가 감소 연산 시 sizeof(double)의 크기만큼 값이 증가 및 감소

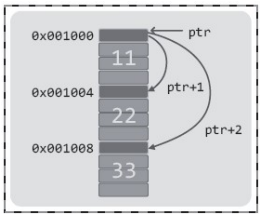
일반화

- type형 포인터 변수 대상의 증가 감소 연산 시 sizeof(type)의 크기만큼 값이 증가 및 감소

포인터를 대상으로 하는 증가 및 감소 연산 (2/2)

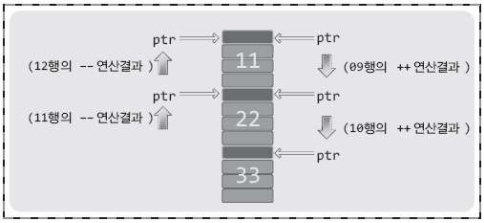
PointerBaseArrayAccess.c

```
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 함수호출 후, ptr++ 실행
    printf("%d ", *ptr); ptr++;
    printf("%d ", *ptr); ptr--;    // printf 함수호출 후, ptr-- 실행
    printf("%d ", *ptr); ptr--;
    printf("%d ", *ptr); printf("\n");
    return 0;
}
```




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

실행결과



int형 포인터 변수의 값은 4씩 증가 및 감소
int형 포인터 변수가 int형 배열을 가리키면,
int형 포인터 변수의 값을 증가 및 감소시켜서 배열 요소에 순차적으로 접근이 가능

중요한 결론! arr[i] == *(arr+i)

```
int main(void)
{
    int arr[3]={11, 22, 33};
    int * ptr=arr;
    printf("%d %d %d \n", *ptr, *(ptr+1), *(ptr+2));
    . . . .
}
```

배열이름도 포인터이니, 포인터 변수를 이용한 배열의 접근방식을 배열의 이름에도 사용할 수 있다. 그리고 배열의 이름을 이용한 접근방식도 포인터 변수를 대상으로 사용할 수 있다. 결론은 arr이 포인터 변수의 이름이건 배열의 이름이건 arr[i] == *(arr+i)

```
printf("%d %d %d \n", *(ptr+0), *(ptr+1), *(ptr+2));    // *(ptr+0)는 *ptr과 같다.
printf("%d %d %d \n", ptr[0], ptr[1], ptr[2]);
printf("%d %d %d \n", *(arr+0), *(arr+1), *(arr+2));    // *(arr+0)는 *arr과 같다.
printf("%d %d %d \n", arr[0], arr[1], arr[2]);
```

(배열 명)++ 가능?

- main 함수에서 다음과 같은 coding이 가능할까? exam_01.c

```
int main(void) {
        int i = 0, sum=0, aryLength;
        int point[ ] = {95, 88, 76, 54, 85, 33, 65, 78, 99, 82};
        aryLength = sizeof (point) / sizeof (int);

        for (i = 0; i < aryLength; i++) {
                sum += point[i];
                sum += *(point++);
        }
        printf("메인에서 구한 합은 %d\n", sum);
}
```

point라는 방은 없음

- sum += *(point++)은 맞나?
  - point++는 point += 1과 동일한 의미이므로 불가능
  - point는 변수가 아닌 상수이므로 (point라는 방은 없음) 증가연산자의 피연산자로 이용이 불가능하기 때문

(포인터) ++ 가능!

- point를 하나의 다른 pointer 변수에 저장하고 이용하면 가능

```
int main(void) {
        int i = 0, sum=0, aryLength;
        int point[ ] = {95, 88, 76, 54, 85, 33, 65, 78, 99, 82};
        int *pi = point; // point는 변수가 아닌 상수, pi는 변수
        aryLength = sizeof (point) / sizeof (int);

        for (i = 0; i < aryLength; i++) {
                sum += *(pi++); // pi는 변수, sum += point[i];
        }
        printf("메인에서 구한 합은 %d₩n", sum);
}
```

12장 실습문제 (Lab1)

- 아래와 같은 예제를 작성해보라.
  - 정수형 변수 num1과 num2를 선언과 동시에 각각 10과 50으로 초기화
  - 정수형 포인터 변수 ptr1과 ptr2를 선언하고 각각 num1과 num2를 가리키게 함
  - 포인터 변수를 사용하여 num1 값을 10 증가시키고, num2 값을 10 감소시킴
  - 두 포인터 변수가 가리키는 대상을 서로 바꿈
  - 두 포인터 변수가 가리키는 변수에 저장된 값을 출력

```
int num1 = 10, num2 = 50;
int *ptr1, *ptr2;

// ptr1, ptr2가 num1, num2를 가리키게 함
// ptr1, ptr2를 사용하여 num1, num2 값을 10씩 변경 (증가/감소)
// ptr1, ptr2가 가리키는 대상을 서로 바꿈
// ptr1, ptr2가 가리키는 변수 값을 출력
```

실습문제 (Lab 2)

- 길이가 5인 int형 배열 arr을 선언하고 이를 6, 7, 8, 9, 10으로 초기화한다.
- 이 배열의 첫 번째 요소를 가리키는 포인터 변수 ptr을 선언한다.
- 포인터 변수 ptr에 저장된 값을 증가시키는 연산을 기반으로 모든 배열 요소의 값을 5씩 증가시킨다.
- 정상적으로 증가가 이루어졌는지 확인할 수 있도록 결과를 출력한다.

```
int arr[5] = {6, 7, 8, 9, 10};
int *ptr = arr;

// 이후는 ptr 변수를 사용하여 연산
```

실습문제 (Lab 3)

- 길이가 6인 int형 배열 arr을 선언하고 이를 1, 2, 3, 4, 5, 6으로 초기화.
- 배열에 저장된 값의 순서가 6, 5, 4, 3, 2, 1이 되도록 변경함.
- 힌트: 배열의 앞과 뒤를 가리키는 포인터 변수 두 개를 선언해서 활용할 것.

```
int arr[6];
int i;
int arrLength;

arrLength = sizeof(arr)/sizeof(int);

for(i = 0; i < arrLength; ++i)
   arr[i] = i+1;

// 나머지 처리
```

```
arr 배열 값 (변경 전): 1 2 3 4 5 6
arr 배열 값 (변경 후): 6 5 4 3 2 1
```

실습 시간 (2019년 9월 17일)

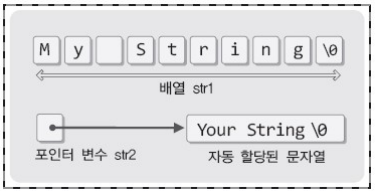
- 예제1 (12장): 1개
  - PointerOperation.c
- 예제2 (13장): 5개
  - ArrayNameType.c, ArrayNamePointerOperation.c, ArrayNamePointer.c, PointerOperationResult.c, PointerBaseArrayAccess.c
- Lab 문제: Lab1.c, Lab2.c, Lab3.c

Chapter 13-3. 상수 형태의 문자열을 가리키는 포인터

두 가지 형태의 문자열 표현

```
char str1[ ] = "My String";
char * str2 = "Your String";
```

문자열의 저장방식




```
int main(void)
{
    char * str = "Your team";
    str = "Our team";    // 의미 있는 문장
    . . . .
}
```

```
int main(void)
{
    char  str[ ] = "Your team";
    str = "Our team";    // 의미 없는 문장
    . . . .
}
```

str1은 문자열이 저장된 배열이다. 즉, 문자 배열이다. 따라서 변수성향의 문자열이다.
str2는 문자열의 주소 값을 저장한다. 즉, 자동 할당된 문자열의 주소 값을 저장한다.
따라서 상수성향의 문자열이다.

두 가지 형태의 문자열 표현의 예

TwoStringType.c

```
int main(void)
{
    char str1[]="My String";     // 변수 형태의 문자열
    char * str2="Your String";   // 상수 형태의 문자열
    printf("%s %s \n", str1, str2);

    str2="Our String";   // 가리키는 대상 변경
    printf("%s %s \n", str1, str2);

    str1[0]='X';   // 문자열 변경 성공!
    str2[0]='X';   // 문자열 변경 실패!
    printf("%s %s \n", str1, str2);
    return 0;
}
```

변수 성향의 str1에 저장된 문자열은 변경이 가능!
반면 상수 성향의 str2에 저장된 문자열은 변경이 불가능!

간혹 상수 성향의 문자열 변경도 허용하는 컴파일러가 있으나, 이러한 형태의 변경은 바람직하지 못하다!

어디서든 선언할 수 있는 상수 형태의 문자열

```
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.

실습문제 (Lab 4)

- 변수 형태의 문자열의 이름을 str1으로 정의하고 키보드로부터 문자열을 입력 받은 후 해당 문자열이 제대로 입력되었는지 출력해 본다.
- 상수 형태의 문자열의 이름을 str2로 정의하고 program 상에서 문자열 하나를 입력 (예, Communications)하고 해당 문자열이 제대로 들어있는지 출력해 본다.
- str2 문자열을 str1에 복사하고 제대로 복사되었는지 출력해본다.

```
문자열을 입력하세요 : Information
입력한 문자열 str1은 Information입니다.
str2 문자열에는 Communications가 있습니다.
str2를 str1에 복사한 후 str1 문자열을 출력하면 Communications입니다.
```

Chapter 13-4. 포인터 변수로 이뤄진 배열: 포인터 배열

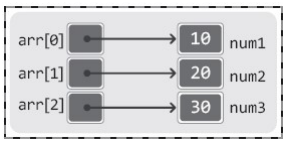
포인터 배열의 이해

```
int * arr1[20];    // 길이가 20인 int형 포인터 배열 arr1
double * arr2[30];     // 길이가 30인 double형 포인터 배열 arr2
```

```
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]);
    return 0;
}
```

실행결과

```
10
20
30
```



PointerArray.c

포인터 배열이라 해서 일반 배열의 선언과 차이가 나지는 않는다.
변수의 자료형을 표시하는 위치에 int나 double을 대신해서 int * 나 double * 가 올 뿐이다.

문자열을 저장하는 포인터 배열

StringArray.c

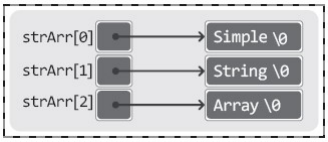
```
int main(void)
{
    char * strArr[3]={"Simple", "String", "Array"};
    printf("%s \n", strArr[0]);
    printf("%s \n", strArr[1]);
    printf("%s \n", strArr[2]);
    return 0;
}
```

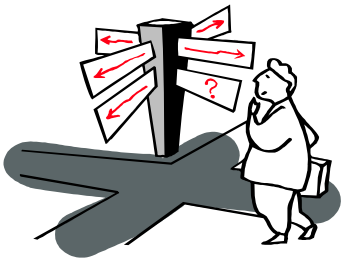
실행결과

```
Simple
String
Array
```

```
char * strArr[3]={"Simple", "String", "Array"};
```

```
char * strArr[3]={0x1004, 0x1048, 0x2012};   // 반환된 주소 값은 임의로 결정하였다.
```





Chapter 13이 끝났습니다. 질문 있으신지요?