배열과 포인터의 기초

20200914

Array

배열(Array)의 기초

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

1차원 배열 선언의 예

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

생성되는 배열의 형태

배열의 필수 요소 3가지

1. 배열을 이루는 요소의 자료형
2. 배열의 이름
3. 배열의 길이

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

배열의 인덱스(index)

생성되는 배열의 형태

첫 번째 배열이니까

순서대로 1, 2, 3, 4 겠지??

배열의 인덱스(index)

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

배열의 위치 정보를 명시하는 인덱스 값은

1이 아닌, **0** 에서부터 시작한다!!

배열의 인덱스(index)

1. 배열의 인덱스는 0번째부터 시작!!

2. 배열의 마지막 인덱스는 항상 배열의 길이보다 -1이다!!

   즉, 배열의 길이와 똑같은 인덱스를 가진 배열은 없다!!

ex) int arr[3] -> arr[0] = 10 / arr[1] = 12 / arr[2] = 14 (배열의 개수 3개)

배열 선언 및 초기화

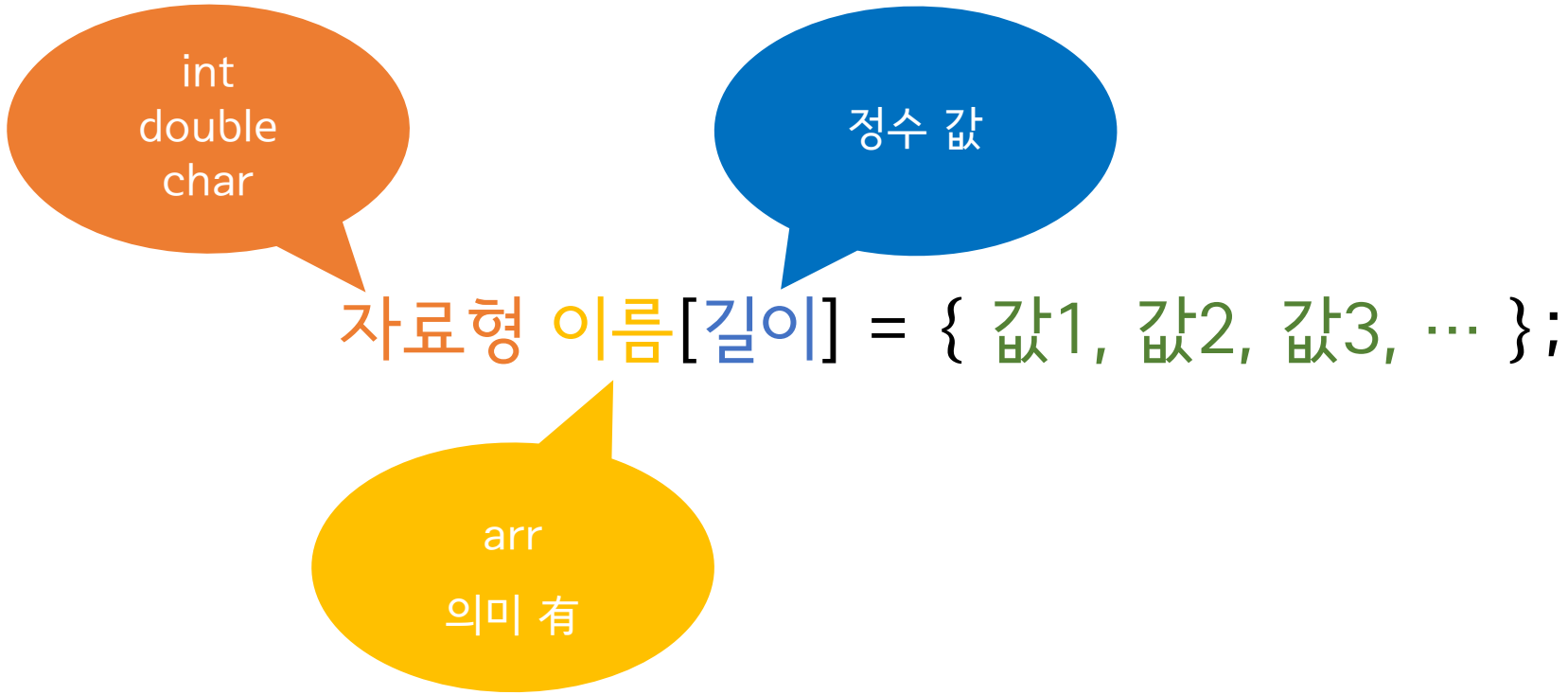
```
// 배열 선언 먼저
int arr[5];
// 그 다음 초기화
arr[0] = 10;
arr[1] = 20;
arr[2] = 22;
arr[3] = 26;
arr[4] = 37;

// 배열 선언 동시에 초기화
int arr[5] = {10, 20, 22, 26, 37};
```

```
// 몇 개인지 모르겠어요
int arr2[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

// 퀴즈
int arr3[5] = {1, 2};
```

배열 선언의 구조



배열의 크기와 길이

배열의 크기? (크기 != 길이)

int arr[5] = {2, 3, 5, 7, 11}; 의 배열의 크기는 얼마? 길이는 얼마?

“Integer형 변수 5개가 있는 array”

따라서, 배열의 길이는 **5**, 크기는 4(자료형) x 5(길이) = **20**

배열의 크기와 길이

```
int arr[5] = {10, 20, 22, 26, 37};
int size_of_Arr = sizeof(arr); // 20 (배열의 크기)
int length_of_Arr = sizeof(arr)/sizeof(int); // 5 (배열의 길이)
```

Practice 1 #배열의 기초 #배열의 인덱스 #배열의 크기

```
scanf("%d", &arr[2]);
```

길이가 5인 int형 배열을 선언해서 프로그램 사용자로부터 총 5개의 정수를 입력 받는다. 그리고 입력이 끝나면 입력된 정수의 총 합을 출력하도록 예제를 작성해보자.

```
출력 결과
배열의 0번째 인덱스: 2
배열의 1번째 인덱스: 3
배열의 2번째 인덱스: 5
배열의 3번째 인덱스: 7
배열의 4번째 인덱스: 9

입력된 전체 배열: 2 3 5 7 9
덧셈 결과 출력: 26
```

Summary

1. 배열의 인덱스는 0부터 시작 & 배열의 크기와 길이는 다름

문자열을
저장하고 싶어요

문자 vs 문자열

문자는 char형 <-> 문자열은 char형 배열

char ch = ‘a’;

Char str[14] = “Good morning!”;

문자는 null문자 X <-> 문자열은 무조건 문장 끝에 null문자 O

배열을 이용한 문자열 변수의 표현

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

배열에 문자열 저장

문자열의 끝에 널 문자라 불리는 ₩0 가 삽입되었음에 주목!
널 문자는 문자열의 끝을 의미함

문자열의 저장을 목적으로 char형 배열을 선언할 경우에는
특수문자 '\0'이 저장될 공간까지 고려해서 배열의 길이를 결정해야함

배열을 이용한 문자열 변수의 표현

```
// 작성
char str[] = "Good morning!";

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

0	1	2	3	4	5	6	7	8	9	10	11	12	13
G	o	o	d		m	o	r	n	i	n	g	!	₩0

문자열의 저장을 목적으로 char형 배열을 선언할 경우에는
특수문자 '\0'이 저장될 공간까지 고려해서 배열의 길이를 결정해야함

배열을 이용한 문자열 변수의 표현

```
**********입력**********
// 직접 입력 (Hard Coding)
char str[14] = "Good morning!";

// scanf통한 입력
char str[100];
scanf("%s", str);
**********************************
```

문자열을 입력받는 배열의 이름 앞에는

& 연산자를 붙이지 않는다!

%s의 의미 : “null 문자가 나올 때까지 출력하라!”

```
**********출력**********
// 한 번에 출력
printf("문자열 출력: %s \n", str);

// 각 인덱스 접근해서 출력
int index = 0;
while(str[index] != '\0') {
    printf("%c", str[index]);
    index++;
}
**********************************
```

문자열의 끝에 널문자가 필요한 이유

문자열의 시작은 판단할 수 있어도 문자열의 끝은 판단이 불가능함.
때문에 **문자열의 끝을 판단**할 수 있도록 널 문자가 삽입이 됨.

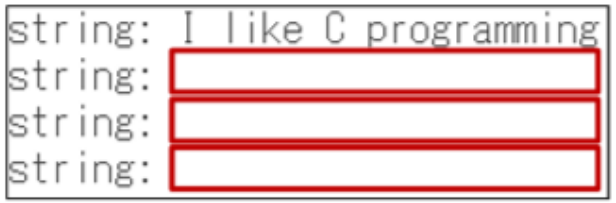
```
int main(void) {
   char str[50] = "I like C programming";
   printf("string: %s \n", str);

   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 시작함. 따라서 시작위치는 확인이 가능함. 하지만 **배열의 끝**이 **문자열의 끝**은 아니므로 널 문자가 삽입되지 않으면 문자열의 끝은 확인이 불가능함



실행결과

위 예제에서 보이듯이 printf 함수도 배열 str 의 시작위치를 기준으로 해서 널 문자를 만날 때까지 출력을 진행함. 따라서 널 문자가 없으면 printf 함수도 문자열의 끝을 알지 못함.

문자 (배열) vs 문자열

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

arr1 은 문자열이 아닌 **문자 배열**, 반면 arr2는 **문자열**!

널 문자의 존재여부는 문자열의 판단여부가 됨

Practice 2 #문자열 #null문자 #인덱스

프로그램 사용자로부터 하나의 영단어를 입력 받아서 입력 받은 영단어의 길이를 계산하여 출력하는 프로그램을 작성해보자.

```
출력 결과
영단어를 입력하세요: programming
영단어의 길이: 10
```

Summary

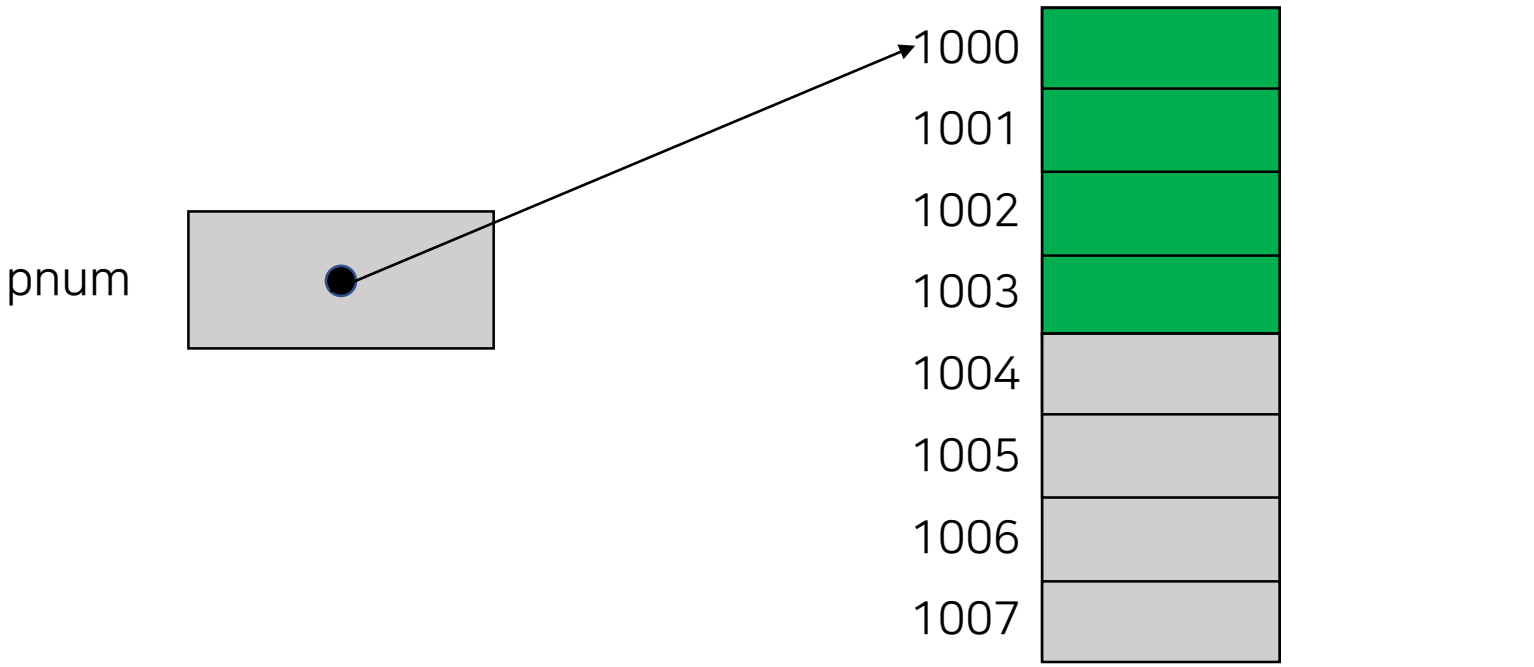
1. 배열의 인덱스는 0부터 시작 & 배열의 크기와 길이는 다름
2. 문자열의 끝에는 항상 null문자가 들어감

Pointer

포인터의 기초

https://www.youtube.com/watch?v=IjyO6IpDrbc&t=0s

```
int num = 7;
int* pnum; // 포인터 변수 pnum의 선언
pnum = &num; // num의 주소 값을 포인터 변수 pnum에 저장
```



포인터 변수의 선언

- **int *** pnum1;

  int * 는 int 형 변수를 가리키는 pnum1 의 선언을 의미함

- **double *** pnum2;

  double * 는 double 형 변수를 가리키는 pnum2 의 선언을 의미함

- **unsigned int *** pnum3;

  unsigned int * 는 unsigned int 형 변수를 가리키는 pnum3 의 선언을 의미함

일반화

- **type *** ptr;

  type 형 변수의 주소 값을 저장하는 포인터 변수 ptr 의 선언

포인터 변수의 선언

```
int * ptr;   // int 형 포인터 변수 ptr 의 선언
int* ptr;    // (상동)
int *ptr;    // (상동)
```

포인터 변수 선언에서 * 의 위치에 따른 차이는 없음
즉, 위의 세 문장은 모두 동일한 포인터 변수의 선언문임

포인터 변수의 선언

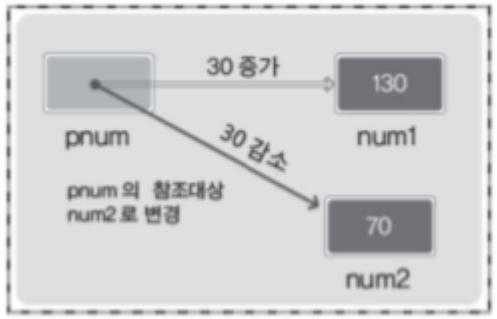
변수의 주소 값을 반환하는 & 연산자

& 연산자는 변수의 주소 값을 반환
int 형 변수 대상의 & 연산의 반환 값은
int 형 포인터 변수에 들어감

```
int num = 5;
int* pnum = &num; // pnum이 num을 가리킴
*pnum = 20; // pnum이 가리키는 공간에 20을 저장
printf("%d", *pnum); // pnum이 가리키는 공간에
                     // 저장된 값을 출력
```

포인터가 가리키는 메모리를 참조하는 * 연산자

- 연산자는 포인터가 가리키는 주소의 값을 반환
  *pnum은 num을 의미함



다양한 포인터 타입이 존재하는 이유

- 포인터 타입은 메모리 공간을 참조하는 방법의 힌트가 됨. 다양한 포인터 타입을 정의한 이유는 * 연산을 통한 메모리의 접근 기준을 마련하기 위함
  - int 형 포인터 변수로 * 연산을 통해 메모리(변수) 접근 시 4바이트 메모리 공간에 부호 있는 정수의 형태로 데이터를 읽고 씀
  - double 형 포인터 변수로 * 연산을 통해 메모리(변수) 접근 시 8바이트 메모리 공간에 부호 있는 실수의 형태로 데이터를 읽고 씀

```
int main(void){
    double num = 3.14;
    int * pnum = &num;
    printf("%d", *pnum);
    . . . .
}
```

형 불일치! 컴파일은 됨

pnum 이 가리키는 것은 double 형 변수인데,
pnum 이 int 형 포인터 변수이므로
int 형 데이터처럼 해석!

주소 값이 정수임에도 불구하고 int 형 변수에 저장하지 않는 이유는 int 형 변수에 저장하면 메모리 공간의 접근을 위한 * 연산이 불가능하기 때문임.

잘못된 포인터의 사용과 널포인터

```
int main(void){
    int * ptr;
    *ptr = 200;
    . . . .
}
```

위험한 코드

ptr 이 garbage 값으로 초기화됨. 따라서 200 이 저장되는 위치는 어디인지 알 수 없음! 매우 위험한 행동!

포인터 변수에 125를 저장했는데 이곳이 어디인가? 역시 매우 위험한 행동!

```
int main(void){
    int * ptr = 125;
    *ptr = 10;
    . . . .
}
```

위험한 코드

```
int main(void){
    int * ptr2 = 0;
    int * ptr2 = NULL;
    . . . .
}
```

안전한 코드

- 잘못된 포인터 연산을 막기 위해서 특정한 값으로 초기화하지 않는 경우에는 **널 포인터**로 초기화하는 것이 안전함
- **널 포인터 NULL 은 숫자 0을 의미함**. 그리고 **0은 0번지를 뜻하는 것이 아니라, 아무것도 가리키지 않는다**는 의미로 해석됨

Practice 3 #포인터의 기초 #포인터의 이해

아래의 예제 실행 시 변수와 포인터 변수의 관계를 그림을 그려서 설명해보자.

또한 출력의 결과도 예상해보자.

```c
int main(void) {
    int num = 10;
    int* ptr1 = &num1;
    int* ptr2 = ptr1;

    (*ptr1)++;
    (*ptr2)++;
    printf("%d \n", num);
    return 0;
}
```

Summary

1. 배열의 인덱스는 0부터 시작 & 배열의 크기와 길이는 다름
2. 문자열의 끝에는 항상 null문자가 들어감
3. 포인터... 무언가(변수의 주소) 를 가리킨다...?