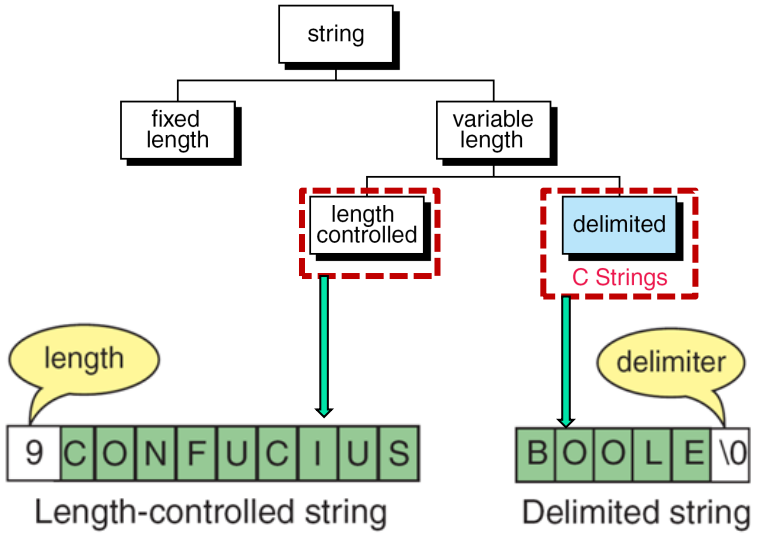
C언어 (CSE2035)
(Chap10. Strings) (1-1)

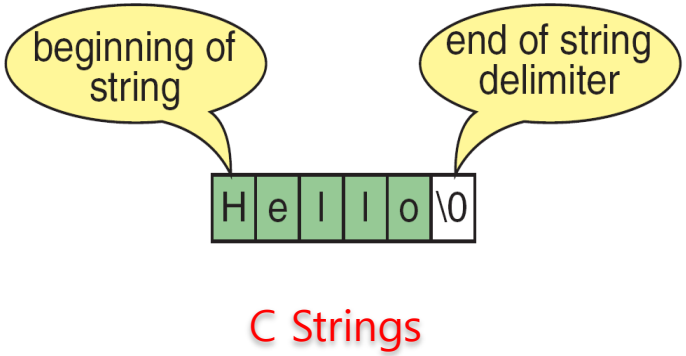
Sungwon Jung, Ph.D.

Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea
Tel: +82-2-705-8930
Email : jungsung@sogang.ac.kr

String concepts

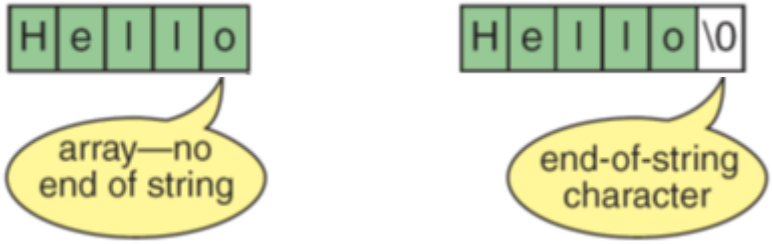
- 문자열(string)은 문자(character)의 연속이다.
  - 즉, 문자열을 구성하는 문자들이 Char 배열의 형태로 메모리에 저장된다.
- 문자열은 길이가 가변적이기 때문에 구현 방법도 다양하다.
  - C언어는 문자열의 끝을 알리는 Null Character('0')를 사용한다.





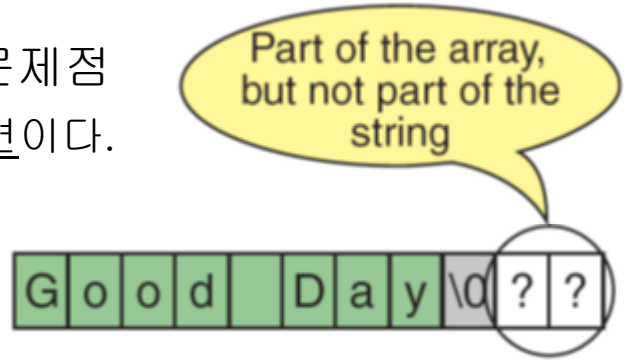
C strings

- **The String Delimiter(\0)**
  - 왜 문자상수 null을 문자열의 마지막에 포함시키는 것이 필요한가?
    - C언어에는 '문자열(String)'이라는 데이터 타입이 없다.
    - C언어에서는 문자열을 저장하기 위해 다른 데이터 타입(Char array)을 사용
    - → 즉, Char Array와 문자열(String)을 구별하는 구분 문자(Delimiter)가 필요하다



  - char타입의 배열을 대신 사용할 때 발생하는 문제점
    - 배열은 길이가 고정이지만, 문자열은 길이가 가변이다.
    - 문자열의 끝을 알리는 delimiter로 null 사용

```
char str[11] = "Good Day";
```

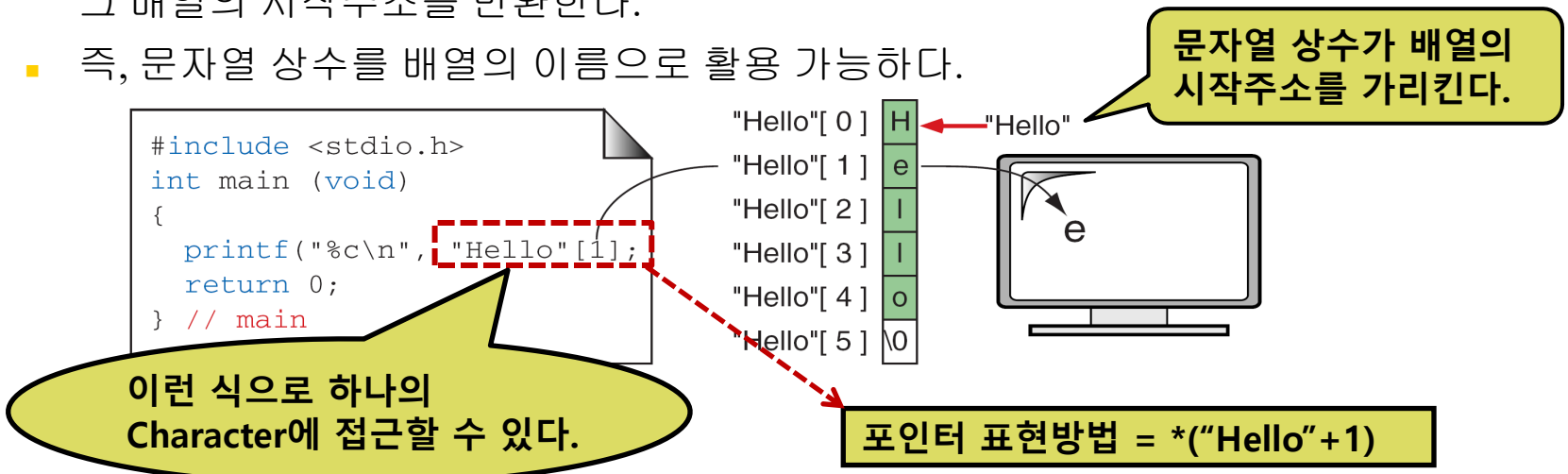


C strings

- 문자열 상수 (String Literals / String Constant)
  - 문자열 상수는 큰 따옴표로 enclose되는 character의 sequence이다.

"C is a high-level language"
"Hello"

  - 문자열 상수는 포인터이다.
    - 문자열 상수가 사용되면, C는 자동적으로 character 배열을 생성해서 이를 저장하고, 그 배열의 시작주소를 반환한다.
    - 즉, 문자열 상수를 배열의 이름으로 활용 가능하다.



C strings

- 문자 상수 (Character Constant)
  - 문자 상수는 작은 따옴표로 enclose되는 하나의 character이다.

```
'a', 'b', 'c', 'd', ..., '0', '1', '2'
'₩0'(null character),
'₩n'(new line), '₩t'(tab), ...
```

  - 문자 상수는 하나의 정수(0~255)이다.
    - ASCII code 체계에 따라 분류된다. Ex) 'A' = 65, '\0' = 0, '0' = 30 등 …
    - 정수이므로 문자 상수 + 정수 등의 정수 연산이 가능하다.

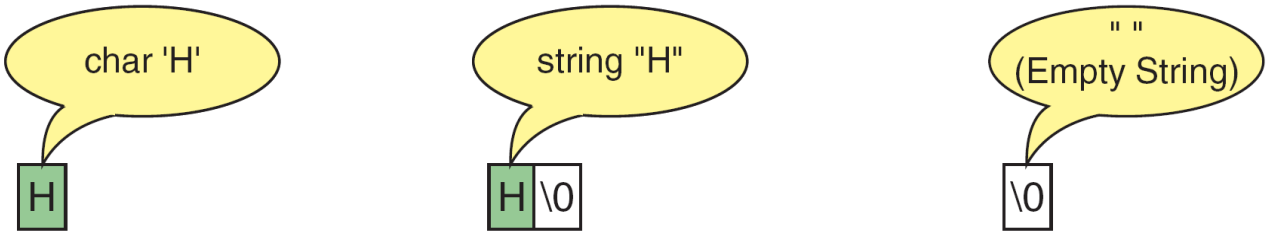
      Ex) 'A' + 1 = 65 + 1 = 66 = 'B'

    - 또한, 정수이므로 조건문, 반복문 등에서도 활용 가능하다.

```
scanf("%c", &c);
if( 'A' <= c && c <= 'Z' )     // 입력 받은 c가 대문자인가?
```

C strings

- **Storing Strings and Characters**
  - 다음 그림은 문자열과 문자형이 메모리에 저장될 때의 차이를 보여준다.
    - "H" 문자열은 'H'와 함께 문자열의 끝을 표시하는 null character를 저장해야 하므로 2byte에 저장된다.
    - null character는 비어있는 문자열임을 표시한다. (1byte에 저장된다.)

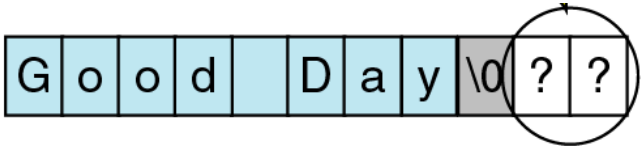


  - 문자열 상수를 저장할 때, 마지막에 null character를 위한 공간까지 고려해야 한다.
    - 즉, "Hello"라는 문자를 저장하기 위해서는 character 5개와 마지막 null character 까지 총 6개의 공간이 필요하다.
    - 따라서 만약 최대 80글자의 문장을 입력 받기 위해서는 80 + 1개의 공간에 입력을 받아야 한다.

C strings

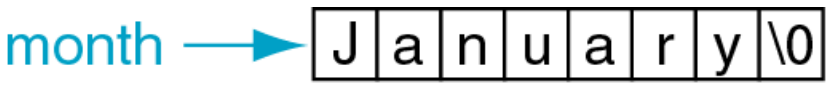
- 문자열 초기화의 4가지 방법
  - 고정된 길이의 배열을 선언

```
char str[11] = "Good Day";
```



  - 배열 크기를 비워둠
    - 문자열의 길이에 딱 맞는 배열이 생성된다.

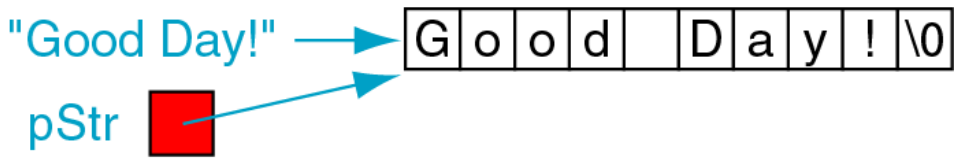
```
char month[] = "January";
```



C strings

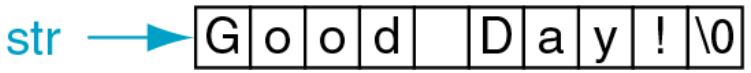
- 포인터를 이용
  - 문자열 상수가 임의의 배열에 저장되고, 이 배열의 시작 주소를 포인터 변수가 갖는다.

```
char *pStr = "Good Day!";
```



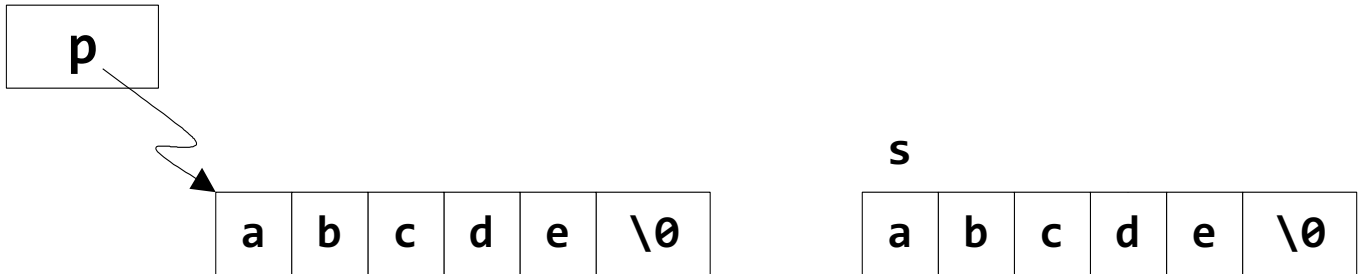
- 배열 선언 후 한 문자씩 입력하는 방법

```
char str[10] =
     {'G','o','o','d',' ','D','a','y','!','\0'};
```



C strings

- 문자열을 초기화 할 때, 포인터와 배열을 사용하는 것의 차이
  - char *p = “abcde”;
    char s[] = “abcde”; /* char s[ ] = {'a', 'b', 'c', 'd', 'e', '\0'}; 와 같음 */
  - 포인터 p는 하나의 변수로서 별도의 저장공간을 갖지만, s는 단지 배열의 시작주소를 나타내므로 하나의 변수가 아니다. 배열의 이름은 포인터처럼 활용할 수 있지만, 포인터와 다른 점은 다른 주소 값을 대입할 수 없다.



C strings

- 즉, 배열로 초기화된 문자열 간에는 대입연산자를 사용할 수 없다.
  - 컴파일 에러가 발생한다.
  - 다음은 str2를 초기화하기 위해 str2에 문자열 str1을 저장하고자 했다. 하지만 str2는 배열의 이름으로 다른 주소 값으로 변경할 수 없으므로 아래 예제는 세 번째 줄에서 compile error를 발생한다.

```
char str1[11] = "Hello";
char str2[11];
str2 = str1;                    /* Compile error */
```

C strings

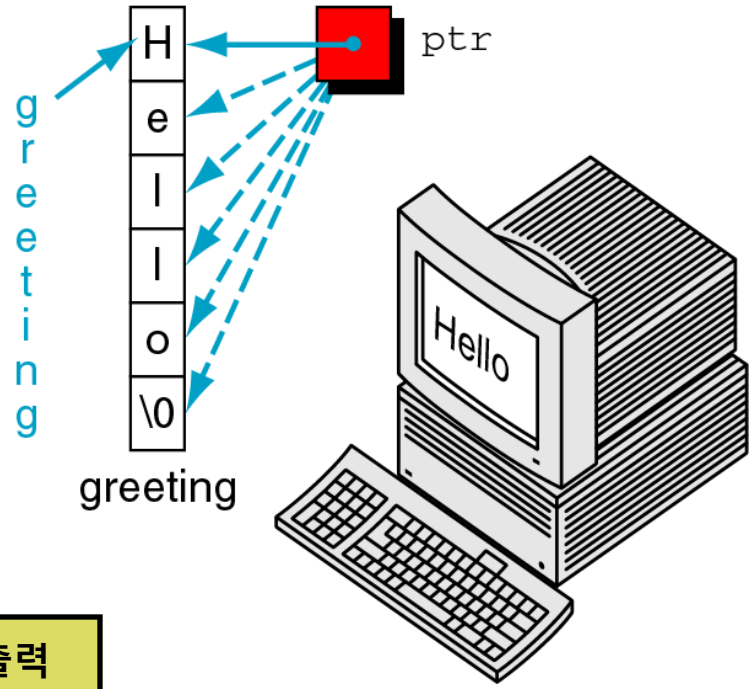
- **Strings and Pointers**
  - 문자열은 배열에 저장되고, 배열의 이름은 포인터이므로, 이를 이용해서 다음과 같이 문자열을 출력하는 것이 가능하다.

```
{ /* Printing Strings */
  char  greeting[] = "Hello";
  char *ptr;

  ptr = greeting;
  while (*ptr != '\0')
       {
     printf( "%c", *ptr);
      ptr++;
    } /* while */
  print("\n");
} /* Printing Strings */
```

null을 만날때까지 반복

한 문자씩 출력



C strings

- **Strings and Pointers**
  - %s는 주어진 포인터의 시작지점부터 Delimiter(\0)까지를 출력하므로, 이를 이용해서 다음과 같이 부분 문자열을 출력하는 것이 가능하다.

```
char s[] = {'a','b','c','d','e','\0'};

printf("Original String : %s\n", s);
printf("Substring from 2 : %s\n", s+2);
```

```
Original String : abcde
Substring from 2 : cde
```