



C 프로그래밍 및 실습

C Programming

이지민



CHAP 09 문자열

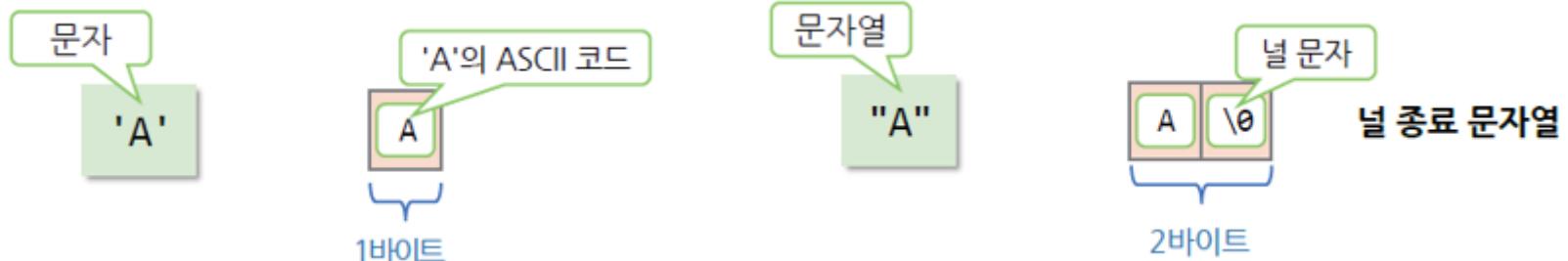
문자 배열_문자와 문자열

❖ 문자열(string) : 연속된 문자들의 모임

- 널 종료 문자열 : 끝을 나타내는 널 문자를 함께 저장
- 문자열 상수(문자열 리터럴) : "A", "hello world"
- 문자열 변수 : 문자 배열(char 배열)

❖ 문자 : 하나의 문자로 구성

- 문자 상수 : 'A', 'W012'
- 문자 변수 : char형 변수, ASCII 코드 저장



문자 배열_문자와 문자열

❖ 문자열 상수

- 값이 변경되지 않는 문자열
- "hello", "Wn"처럼 표시한다.
- 문자열 리터럴이라고도 한다.

❖ 문자열 변수

- 문자 배열에 저장한다.
- 실행 중에 사용자로부터 입력받은 문자열을 저장하거나, 내용이 변경되는 문자열을 저장하려면 문자 배열을 사용해야 한다.

문자 배열_문자 배열의 선언 및 초기화

❖ 배열의 크기는 ‘저장할 문자열의 길이+1’로 지정한다.

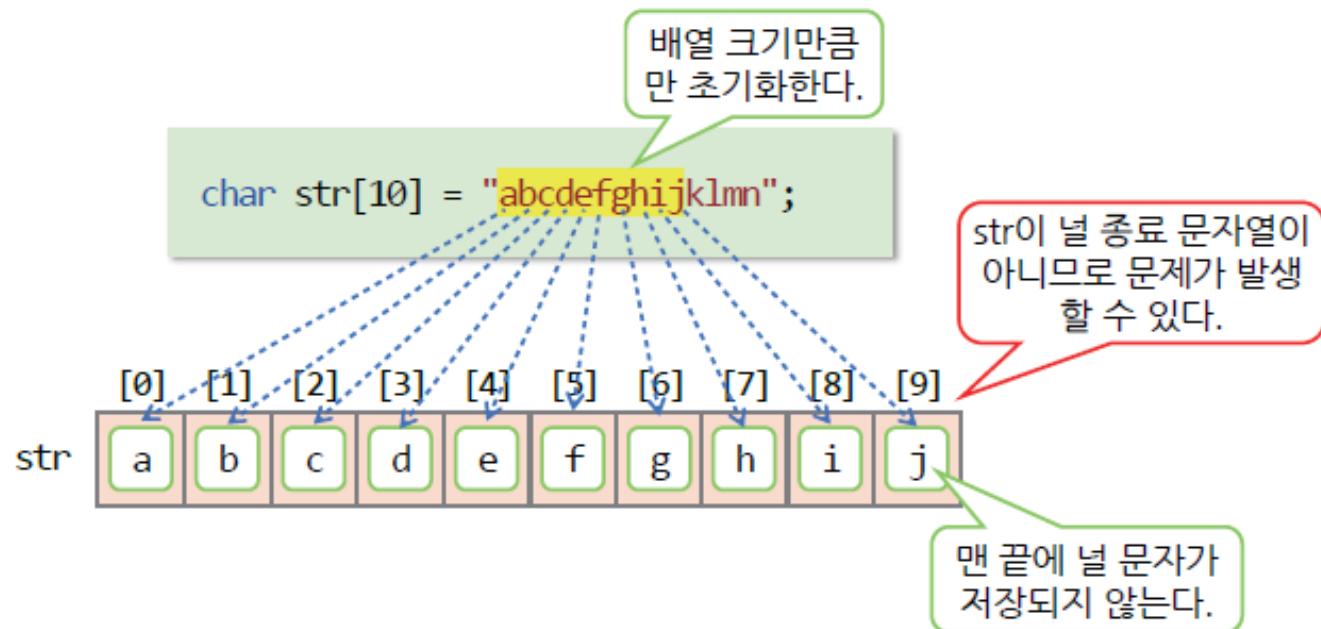
```
char str[10];
```

길이가 9인 문자열을 저장하기
위한 문자 배열

```
char str[10] = "abc";
```

❖ 문자 배열을 초기화할 때는 문자열 리터럴을 이용한다.

- 문자 배열의 크기보다 긴 문자열로 초기화하면 문제가 발생할 수 있다.



문자 배열_문자 배열의 선언 및 초기화

❖ 초기값을 지정하는 경우에는 문자 배열의 크기를 생략할 수 있다.

- ‘문자열의 길이+1’의 크기로 문자 배열을 할당한다.

```
char str[] = "abc";
```

크기가 4인 문자 배열

❖ 문자 배열 전체를 널 문자로 초기화하려면 널 문자열("")로 초기화한다.

- 문자열의 끝에 널 문자 저장
- 배열의 나머지 원소도 널 문자로 초기화

```
char str[10] = "abc";
```

```
char str[10] = "";
```

널 문자 하나로
이루어진 문자열

문자 배열_문자 배열의 사용

- ❖ 문자 배열의 인덱스를 이용하면, 문자열의 문자를 하나씩 읽거나 변경할 수 있다.

```
char str[10] = "abc";  
str[0] = 'A';
```

- ❖ 문자열 전체를 출력하려면 %s 서식 지정자를 사용한다.
 - 문자 배열을 직접 printf 함수의 첫 번째 인자로 전달할 수도 있다.

```
printf("str = %s\n", str);  
printf(str);
```

널 문자를 만날 때까지 str 배열의 원소를 모두 출력한다.

- ❖ 문자 배열에 문자열 리터럴을 대입하면 안 된다.



```
str = "XYZ";
```

배열 이름은 배열의 시작 주소이므로 변경할 수 없다.

문자 배열_문자 배열의 사용

❖ 문자 배열의 선언 및 초기화, 사용 예

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     char str1[10] = "abc";
06     char str2[10] = "very long string";
07     char str3[] = "abc";
08     char str4[10] = "";
09     int i;
```

문자 배열보다 긴 문자열로
초기화하는 경우

문자 배열_문자 배열의 사용

❖ 문자와 문자열

```
11     str1[0] = 'A';
12
13     printf("str1 = ");
14
15     for (i = 0; str1[i] != '\0'; i++)
16         printf("%c", str1[i]);
17
18     printf("\n");
19
20 }
```

널 문자열이므로 아무것도 출력되지 않는다.

```
printf("str1 = ");
for (i = 0; str1[i] != '\0'; i++)
    printf("%c", str1[i]);
printf("\n");
```

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

```
printf(str3);
```

```
printf("\nstr4 = %s\n", str4);
```

널 문자를 만날 때까지
str1[i]를 한 문자씩
출력한다.

문자열의 끝에 널 문자가 없
으므로 쓰레기값 출력

실행 결과

```
str1 = Abc
```

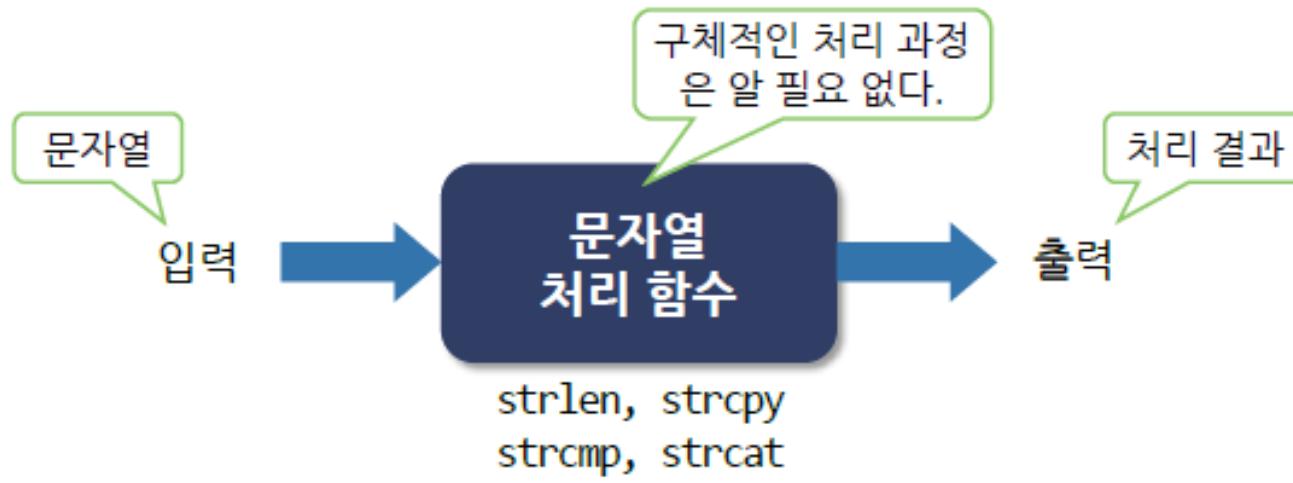
```
str2 = very long 啼啼啼啼啼Abc
```

```
abc
```

```
str4 = -
```

문자 배열_표준 C의 문자열 처리 함수

- ❖ 문자열 처리 함수를 이용하면 구체적인 처리 과정은 알 필요 없이 간단하게 문자열을 다룰 수 있다.



- ❖ 문자열 처리 함수를 사용하려면 `<string.h>`를 포함해야 한다.

문자 배열_표준 C의 문자열 처리 함수

❖ 표준 C 라이브러리의 문자열 처리 함수

함수	역할
strlen(<i>str</i>);	<i>str</i> 의 길이를 구한다. (널 문자 제외)
strcpy(<i>dest</i> , <i>src</i>);	<i>src</i> 를 <i>dest</i> 로 복사한다.
strcmp(<i>lhs</i> , <i>rhs</i>);	<i>lhs</i> 와 <i>rhs</i> 를 비교해서 같으면 0을, <i>lhs</i> > <i>rhs</i> 면 0보다 큰 값을, <i>lhs</i> < <i>rhs</i> 면 0보다 작은 값을 리턴한다.
strcat(<i>dest</i> , <i>src</i>);	<i>dest</i> 의 끝에 <i>src</i> 를 연결한다.
strchr(<i>str</i> , <i>ch</i>);	<i>str</i> 에서 <i>ch</i> 문자를 찾는다. (찾은 문자의 주소 리턴)
strstr(<i>str</i> , <i>substr</i>);	<i>str</i> 에서 <i>substr</i> 문자열을 찾는다. (찾은 문자열의 주소 리턴)
strtok(<i>str</i> , <i>delim</i>);	<i>str</i> 을 <i>delim</i> 을 이용해서 토큰으로 분리한다. (토큰 문자열 리턴)

문자 배열_표준 C의 문자열 처리 함수

❖ 표준 C 라이브러리의 문자 처리 함수

- 문자 처리 함수를 사용하려면 <ctype.h>를 포함해야 한다.

함수	설명	함수	설명
isalnum(ch);	알파벳이나 숫자인지 검사한다.	isalpha(ch);	알파벳인지 검사한다.
isdigit(ch);	숫자인지 검사한다.	islower(ch);	소문자인지 검사한다.
isupper(ch);	대문자인지 검사한다.	isspace(ch);	공백 문자인지 검사한다.
isxdigit(ch);	16진수 숫자인지 검사한다.	tolower(ch);	소문자로 변환한다.
toupper(ch);	대문자로 변환한다.		

문자 배열_표준 C의 문자열 처리 함수

❖ 표준 C 데이터 변환 함수

헤더 파일	함수 원형	설명
⟨stdlib.h⟩	int atoi(const char* str);	문자열을 정수로 변환한다.
	double atof(const char* str);	문자열을 실수로 변환한다.
	long atol(const char* str);	문자열을 long형 값으로 변환한다.
⟨stdio.h⟩	int sscanf(const char* buff, const char* format, ...);	문자열을 정수나 실수로 변환해서 읽어온다.
	int sprintf(char* buff, const char* format, ...);	형식 문자열을 이용해서 정수나 실수를 문자열로 변환한다.

문자 배열_표준 C 문자열 처리 함수

❖ 표준 C 문자열 입출력 함수

- 문자열 입출력 함수를 사용하려면 `<stdio.h>`를 포함해야 한다.

입출력 함수	설명
<code>scanf("%s", str);</code>	공백 문자까지 문자열을 입력받아서 <code>str</code> 에 저장한다.
<code>printf(str);</code> <code>printf("%s", str);</code>	<code>str</code> 을 출력한다.
<code>gets_s(str, count);</code>	한 줄의 문자열을 읽어서 <code>str</code> 에 저장한다. (줄바꿈 문자 포함 X)
<code>fgets(str, count, stdin);</code>	한 줄의 문자열을 읽어서 <code>str</code> 에 저장한다. (줄바꿈 문자 포함)
<code>puts(str);</code>	<code>str</code> 을 출력하고 줄을 바꾼다.

문자 배열_표준 C 문자열 처리 함수

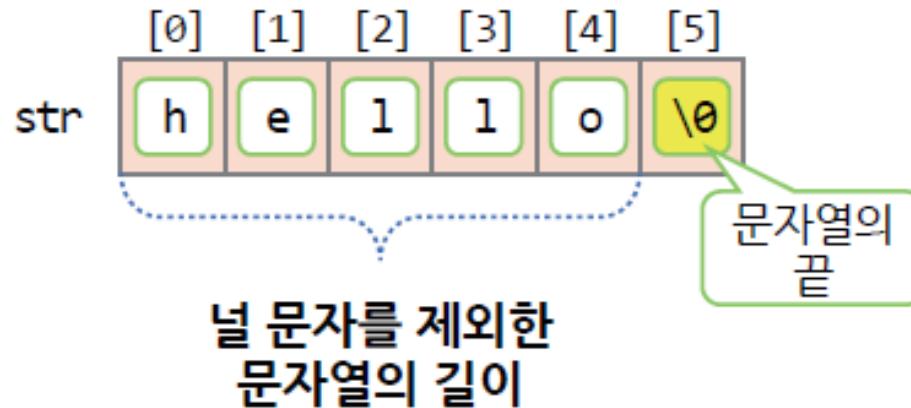
❖ **strlen** 함수 : 널 문자를 제외한 문자열의 길이를 구한다.

```
size_t strlen(const char* str);
```

문자 배열이나 문자열
리터럴을 전달

```
printf("%d\n", strlen(str));
```

```
printf("%d\n", strlen("good bye"));
```



문자 배열_표준 C 문자열 처리 함수

❖ 여러 가지 문자열의 길이 구하기 예

```
01 #include <stdio.h>
02 #include <string.h>
04 int main(void)
05 {
06     char str[10] = "hello";
07     int len = 0;
09     printf("str의 길이: %d\n", strlen(str));
10     printf("\\"good bye\"의 길이: %d\n", strlen("good bye"));
12     printf("str = %s\n", str);
13     len = strlen(str);
14     if (len > 0)
15         str[len - 1] = '\0';
16     printf("str = %s\n", str);
17 }
```

문자열 처리 함수 사용시
포함해야 한다.

실행 결과

```
str의 길이: 5
"good bye"의 길이: 8
str = hello
str = hell
```

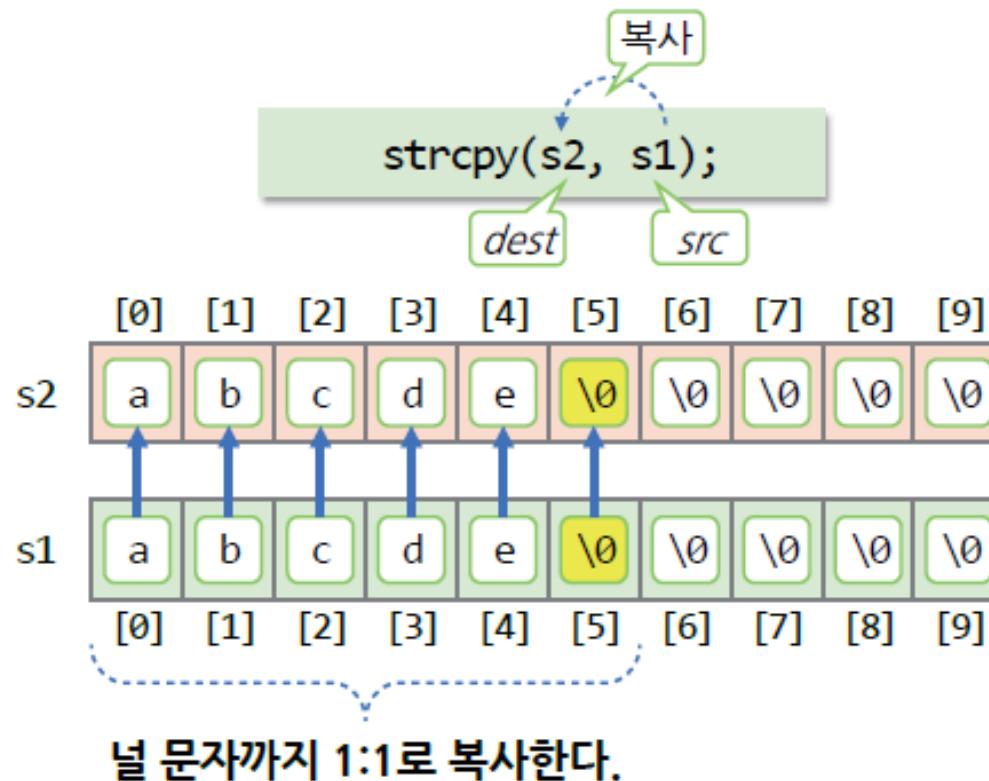
문자열 안에서 " 문자를 사용하
려면 \와 함께 지정한다.

널 문자열이 아니면 마지막
한 글자를 삭제한다.

문자 배열_표준 C 문자열 처리 함수

- ❖ **strcpy** 함수 : *src* 문자열을 *dest* 문자 배열로 복사한다.
 - *src* 문자열을 *dest* 문자 배열을 복사한 다음에 *dest*를 리턴

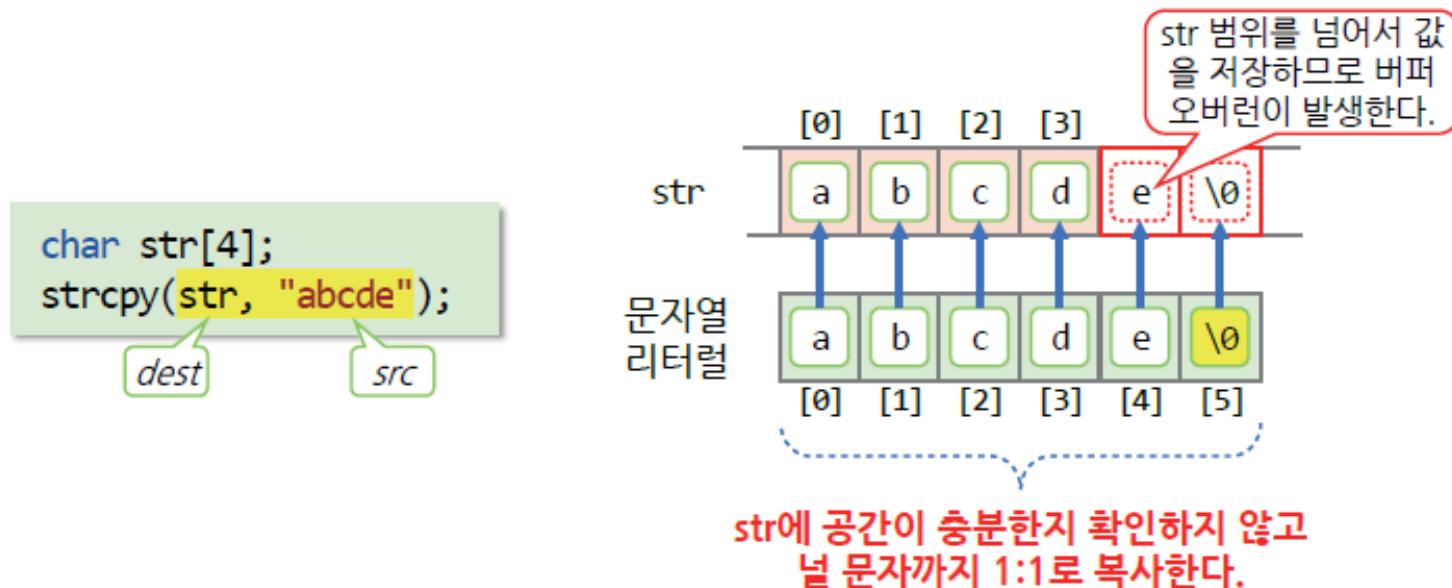
```
char* strcpy(char* dest, const char* src);
```



문자 배열_표준 C 문자열 처리 함수

❖ strcpy 함수의 안전성 문제(1/2)

- Visual Studio 2019에서 strcpy 함수를 사용하면, 컴파일 에러가 발생한다.
 - strcpy 함수는 *dest* 문자 배열의 크기에 관계없이 무조건 *src* 문자열을 복사하므로 버퍼 오버런의 위험이 있다.



문자 배열_표준 C 문자열 처리 함수

❖ strcpy 함수의 안전성 문제(2/2)

▪ 버퍼 오버런(buffer overrun)

- 할당 받은 메모리 범위를 넘어서 값을 저장할 때 메모리가 변조(corrupt)되는 상황
- C99 이상을 지원하는 컴파일러는 표준 C 라이브러리 함수 중 버퍼 오버런의 위험성이 있는 함수에 대하여 컴파일 경고/에러를 발생시킨다.

▪ 안전성 관련 컴파일 경고/에러를 없애는 방법

- 안전 문자열 처리 함수인 strcpy_s를 대신 사용한다.
- _CRT_SECURE_NO_WARNINGS 매크로를 라이브러리 헤더를 포함하는 문장 앞에 정의 한다.

```
#define _CRT_SECURE_NO_WARNINGS  
#include <stdio.h>  
#include <string.h>
```

라이브러리 헤더 보다 앞에
매크로를 정의해야 한다.

문자 배열_표준 C 문자열 처리 함수

❖ 문자열 맞바꾸기 예

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 #include <string.h>
05 int main(void)
06 {
07     char str1[10] = "";
08     char str2[10] = "";
09     char temp[10];
10
11     printf("2개의 문자열? ");
12     scanf("%s %s", str1, str2);
13     printf("str1 = %s, str2 = %s\n", str1, str2);
```

Visual Studio 2019에서
scanf, strcpy 사용 시 필요

널 문자열로 초기화

빈칸으로 구분해서 2개의
문자열 입력

문자 배열_표준 C 문자열 처리 함수

```
16     strcpy(temp, str1);
17     strcpy(str1, str2);
18     strcpy(str2, temp);
19     printf("str1 = %s, str2 = %s\n", str1, str2);
20 }
```

두 문자 배열을
swap한다.

실행 결과

빈칸으로 구분
해서 입력한다.

2개의 문자열? mango blueberry
str1 = mango, str2 = blueberry
str1 = blueberry, str2 = mango

문자 배열_표준 C 문자열 처리 함수

❖ **strcmp** 함수 : *lhs* 문자열과 *rhs* 문자열을 알파벳순으로 비교한다.

- *lhs*와 *rhs*가 같으면 0을, *lhs*가 *rhs*보다 알파벳순으로 앞쪽이면 음수를, 뒤쪽이면 양수를 리턴한다.

```
int strcmp(const char* lhs, const char* rhs);
```

```
char s1[10] = "apple";
char s2[10] = "apple";
if (strcmp(s1, s2) == 0)
    printf("same string\n");
```

s1과 s2의 내용이 같은지 비교한다.



- 관계 연산자를 이용하면 문자열의 주소를 비교한다.

```
if (s1 == s2)
    printf("s1과 s2의 같습니다.\n");
```

문자 배열_표준 C 문자열 처리 함수

❖ 문자열의 비교 예

```
01 #include <stdio.h>
02 #include <string.h>
04 int main(void)
05 {
06     char s1[10] = "apple";
07     char s2[10] = "apple";
09     if (s1 == s2)
10         printf("s1과 s2의 주소가 같습니다.\n");
11     else
12         printf("s1과 s2의 주소가 다릅니다.\n");
14     if (strcmp(s1, s2) == 0)
15         printf("s1과 s2의 내용이 같습니다.\n");
16     else
17         printf("s1과 s2의 내용이 다릅니다.\n");
18 }
```

실행 결과

s1과 s2의 주소가 다릅니다.
s1과 s2의 내용이 같습니다.

s1과 s2의 주소가
같은지 비교한다.

s1과 s2의 내용이
같은지 비교한다.

문자 배열_표준 C 문자열 처리 함수

❖ **strcat** 함수 : *dest* 문자 배열에 저장된 문자열의 끝에 *src* 문자열을 복사해서 연결

- *dest* 문자 배열에 *src* 문자열을 연결할 만큼 메모리가 충분한지를 확인하지 않으므로 버퍼 오버런의 위험이 있다.

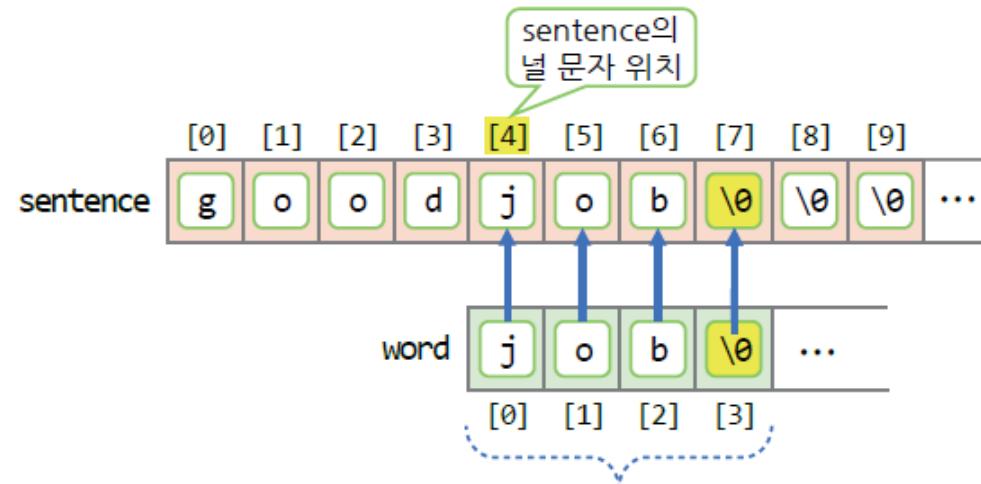
```
char* strcat(char* dest, const char* src);
```

```
char sentence[100] = "";  
char word[20];  
scanf("%s", word);  
strcat(sentence, word);
```

입력받은 word를
sentence의 끝에
붙인다.

dest

src



널 문자까지 1:1로 복사한다.

문자 배열_표준 C 문자열 처리 함수

❖ 문자열의 연결 예

```
01 #define _CRT_SECURE_NO_WARNINGS  
02 #include <stdio.h>  
03 #include <string.h>  
05 int main(void)  
06 {  
07     char sentence[100] = "";  
08     char word[20];  
10     do {  
11         printf("단어? ");  
12         scanf("%s", word);  
13         strcat(sentence, word);  
14         strcat(sentence, " ");  
15     } while (strcmp(word, ".") != 0);  
17     printf("%s\n", sentence);  
18 }
```

실행 결과

단어? this
단어? program
단어? tests
단어? strcat
단어? .
this program tests strcat .

입력받은 단어를
sentence의 끝에 붙인다.

단어를 구분할 수 있도록
빈칸을 붙인다.

."\r"이 입력될 때까지 반복한다.

문자 배열_표준 C 문자열 처리 함수

❖ **strchr** 함수와 **strstr** 함수 : **strchr** 함수는 *str*에서 *ch* 문자를 찾고, **strstr** 함수는 *str*에서 *substr* 문자열을 찾는다.

- **strchr**와 **strstr** 함수는 문자나 문자열을 찾으면 찾은 위치에 있는 문자의 주소(char*형)를 리턴한다.
- 찾을 수 없으면 **NULL**을 리턴한다.

```
char* strchr(const char* str, int ch);
```

```
char* strstr(const char* str, const char* substr);
```

문자 배열_표준 C 문자열 처리 함수

❖ 문자열의 검색 예

01 #include <stdio.h>	실행 결과	
02 #include <string.h>		
04 int main(void)		
05 {		
06 char filename[] = "readme.txt";		filename에서 '.'를 찾아 그 주소를 받아온다.
07 char *p = NULL;		
09 p = strchr(filename, '.');		
10 if (p != NULL)		'.' 다음 위치의 문자열 주소
11 printf("file extension: %s\n", p + 1);		
12		
13 p = strstr(filename, ".txt");		filename에서 ".txt"를 찾아 그 주소를 받아온다.
14 if (p != NULL)		
15 printf("file type: TEXT file\n");		
16 }		

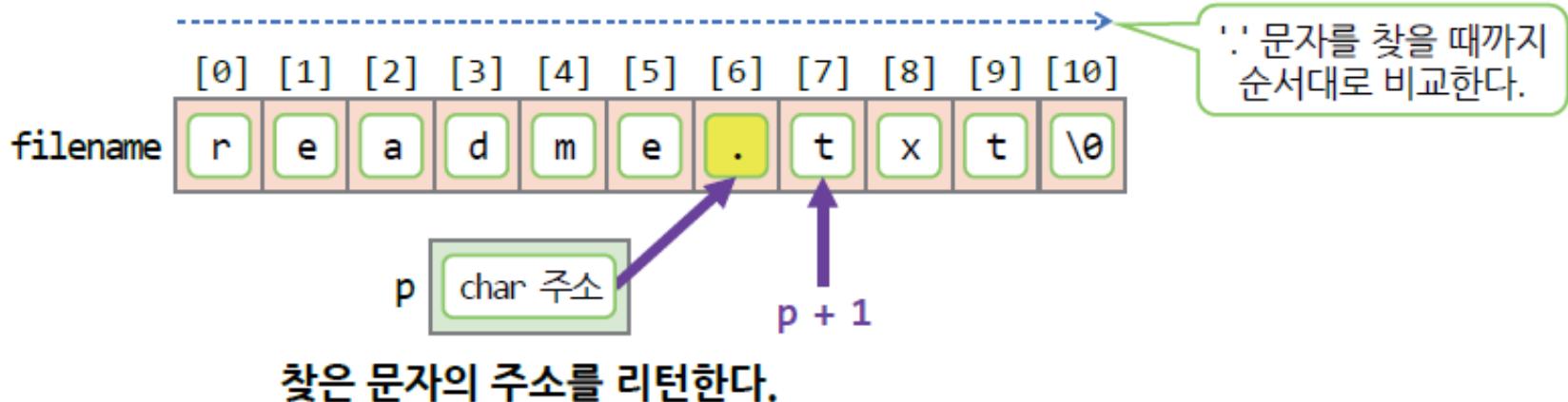
문자 배열_표준 C 문자열 처리 함수

❖ 문자열의 검색 예시 프로그램 실행과정

```
char filename[] = "readme.txt";
char *p = NULL;
p = strchr(filename, '.');
if (p != NULL)
    printf("file extension: %s\n", p + 1);
```

‘‘ 문자가 있는 위치를 찾는다.’’

‘‘ 다음 위치에 있는 문자열을 출력한다.’’



문자 배열_표준 C 문자열 처리 함수

❖ **strtok** 함수 : *str* 문자열을 *delim* 문자열에 있는 구분 문자들을 이용해서 분리한다.

- 토큰 : 문장에서 더 이상 나눌 수 없는 최소 단위
- 토큰으로 쪼개고 토큰의 주소를 리턴한다. 토큰이 없으면 NULL을 리턴한다.
- strtok 함수 호출 후에 첫 번째 매개변수인 str 이 변경된다.
- strtok 함수의 첫 번째 인자로 NULL을 지정하면 계속해서 다음 토큰을 얻을 수 있다.

```
char* strtok(char* str, const char* delim);
```

출력 매개변수

문자 배열_표준 C 문자열 처리 함수

❖ 문자열의 토큰 나누기 예

```
01 #define _CRT_SECURE_NO_WARNINGS  
02 #include <stdio.h>  
03 #include <string.h>  
05 int main(void)  
06 {  
07     char phone[] = "010-123-4567";  
08     char *p = NULL;  
10     p = strtok(phone, "-");  
11     printf("mobile : %s\n", p);  
12     p = strtok(NULL, "-");  
13     printf("prefix : %s\n", p);  
14     p = strtok(NULL, "-");  
15     printf("line no.: %s\n", p);  
16 }
```

실행 결과

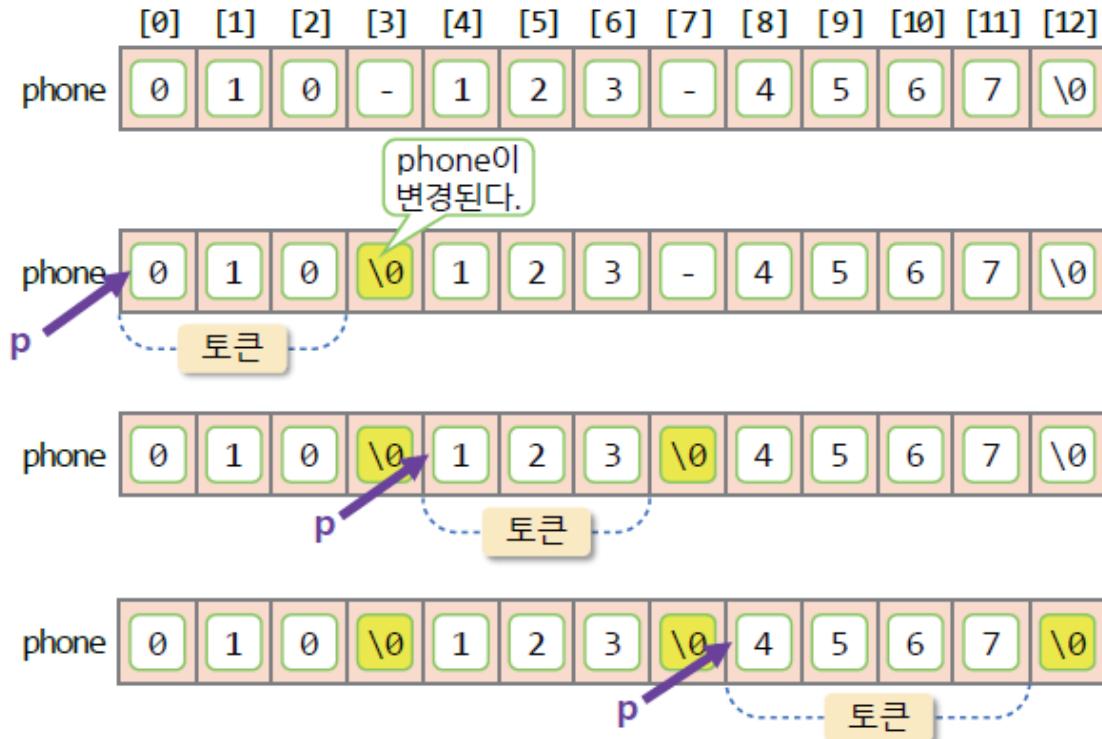
```
mobile : 010  
prefix : 123  
line no.: 4567
```

'-'을 이용해서 토큰을
구분한다.

문자 배열_표준 C 문자열 처리 함수

❖ 문자열의 토큰 나누기 예시 프로그램 실행과정

```
char phone[] =  
    "010-123-4567";  
char *p = NULL;  
  
p = strtok(phone, "-");  
    첫 번째 토큰 리턴  
  
p = strtok(NULL, "-");  
    두 번째 토큰 리턴  
  
p = strtok(NULL, "-");  
    세 번째 토큰 리턴
```



문자 배열_표준 C 문자열 처리 함수

❖ 문자열의 입출력(1/2)

- **scanf** 함수의 서식 지정자로 **%s**을 사용한다.
 - 입력 버퍼에서 공백 문자를 만날 때까지 문자열을 읽어온다.
- **빈칸을 포함한 문자열을 입력받으려면 gets_s 함수나 fgets 함수를 이용한다.**

str 배열의 크기

```
char* fgets(char* str, int count, FILE* stream);  
char* gets_s(char* str, size_t n);
```

str 배열의 크기

줄바꿈 문자까지
str로 읽어온다.

```
char str[128];  
fgets(str, sizeof(str), stdin);  
gets_s(str, sizeof(str));
```

str에 줄바꿈 문
자가 포함된다.

str에 줄바꿈 문자가
포함 되지 않는다.

문자 배열_표준 C 문자열 처리 함수

❖ 문자열의 입출력(2/2)

- 한 줄의 문자열을 출력하려면 puts 함수를 사용한다.
 - 출력 후 자동으로 줄이 바뀐다.

```
int puts(const char* str);
```

- gets_s 함수나 puts 함수는 주로 sscanf, sprintf 함수와 함께 사용된다.

```
int sscanf(const char* buffer, const char* format, ...);  
int sprintf(char* buffer, const char* format, ...);
```

```
gets_s(str, sizeof(str));
```

```
sscanf(str, "%d", &n);
```

str에서 정수를 읽어서
n에 저장한다.

```
sprintf(str, "n = %d", n);
```

```
puts(n);
```

출력할 문자열을 만들어
str에 저장한다.

문자 배열_표준 C 문자열 처리 함수

❖ 문자열의 입출력 예

```
01 #define _CRT_SECURE_NO_WARNINGS  
02 #include <stdio.h>  
03 #include <string.h>  
05 int main(void)  
06 {  
07     char str_in[128];  
08     char str_out[128];  
09     int year, month, day;  
11     printf("날짜(yyyymmdd)? ");  
12     gets_s(str_in, sizeof(str_in));  
15     sscanf(str_in, "%4d%2d%2d", &year, &month, &day);  
18     sprintf(str_out, "Due Date: %04d-%02d-%02d", year, month, day);  
19     puts(str_out);  
20 }
```

실행 결과

날짜(yyyymmdd)? 20190501
Due Date: 2019-05-01

한 줄의 문자열 입력

출력할 문자열을
str_out에 생성한다.

문자 배열_문자열 포인터

❖ char*형의 포인터

- char형의 변수 또는 char형 배열의 원소를 가리킬 수 있다.
- 문자열을 가리키는 용도로 사용된다.

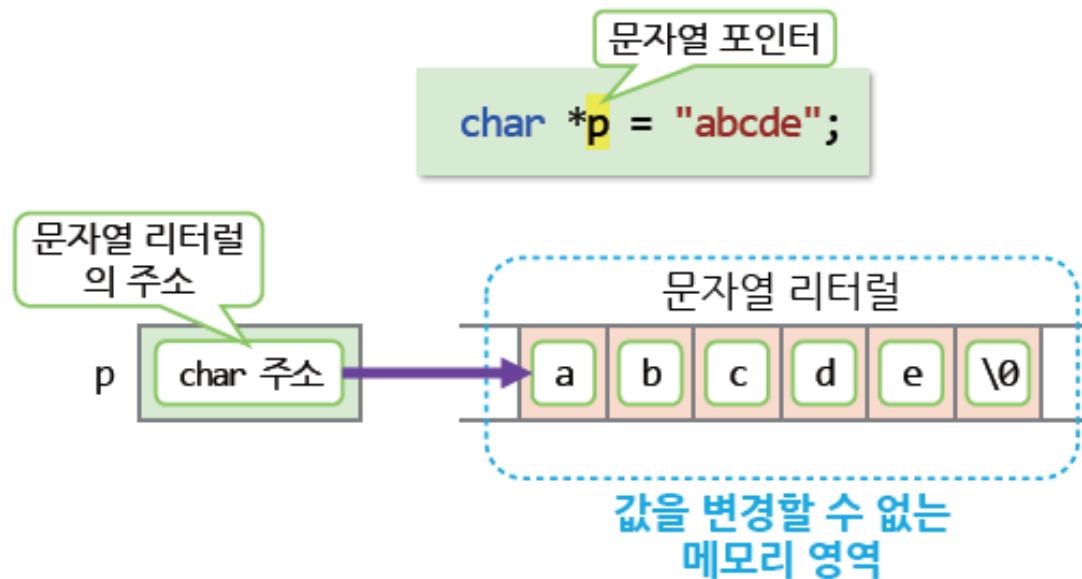
```
char* p = "abcde";
```

문자열 포인터

문자 배열_문자열 포인터

❖ 문자열 리터럴(1/3)

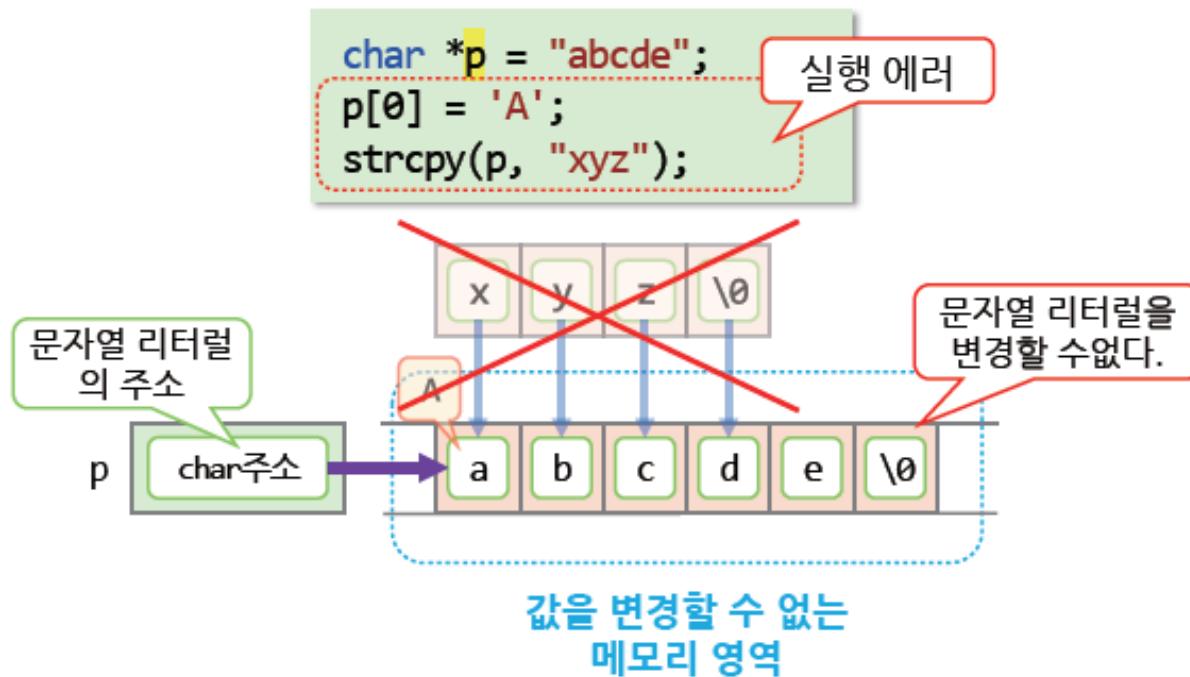
- 문자열 리터럴은 문자열 리터럴의 주소를 의미
 - 상수는 CPU레지스터에 값을 저장하고 사용하는 임시값
 - 문자열 리터럴은 예외적으로 메모리에 할당



문자 배열_문자열 포인터

❖ 문자열 리터럴(2/3)

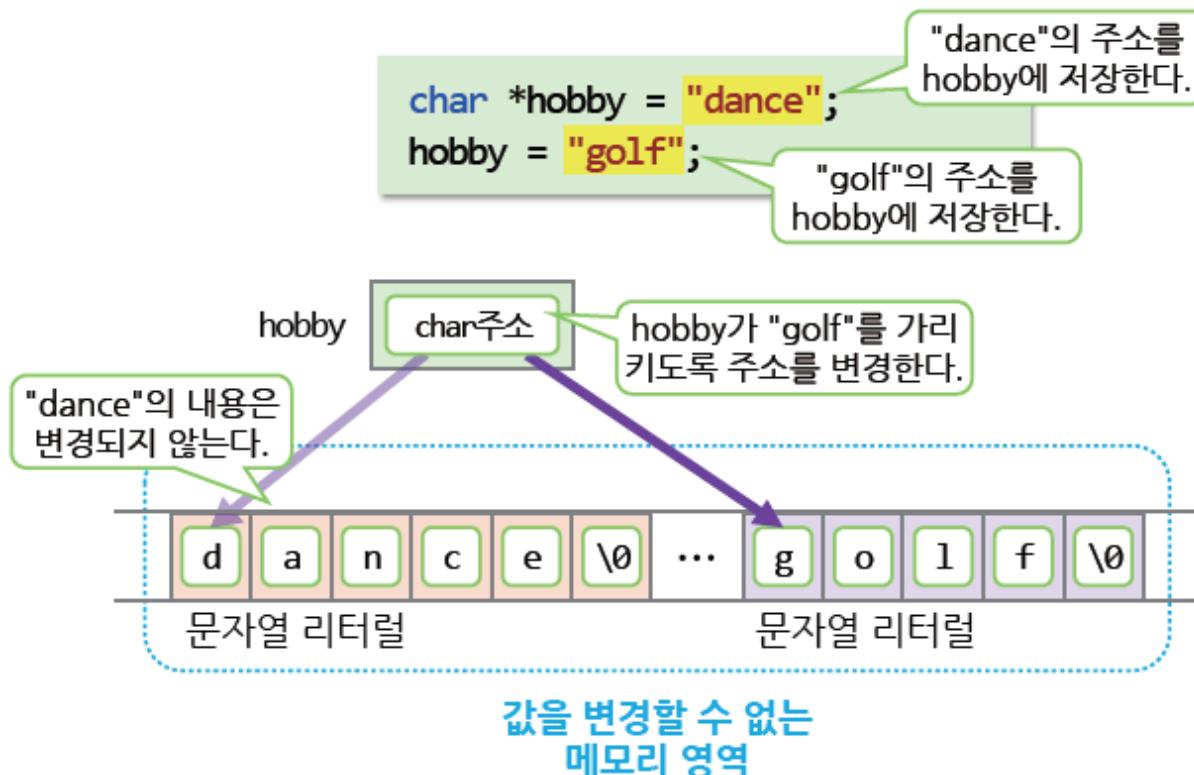
- 문자열 리터럴은 변경할 수 없다.
 - 문자열 리터럴은 변경할 수 없는 메모리에 할당되므로 변경하려고 하면 실행 에러가 발생



문자 배열_문자열 포인터

❖ 문자열 리터럴(3/3)

- 문자열 포인터가 다른 문자열 리터럴을 가리킬 수는 있다.
 - 문자열 포인터에 다른 문자열 리터럴을 대입하면, 포인터가 가리키는 대상만 달라진다.



문자 배열_문자열 포인터

❖ char*형 포인터의 용도

- char*형의 문자열 포인터는 주로 문자 배열을 가리키는 데 사용

```
char str[64] = "";  
char* p = str;
```

p는 str 배열을
가리킨다.

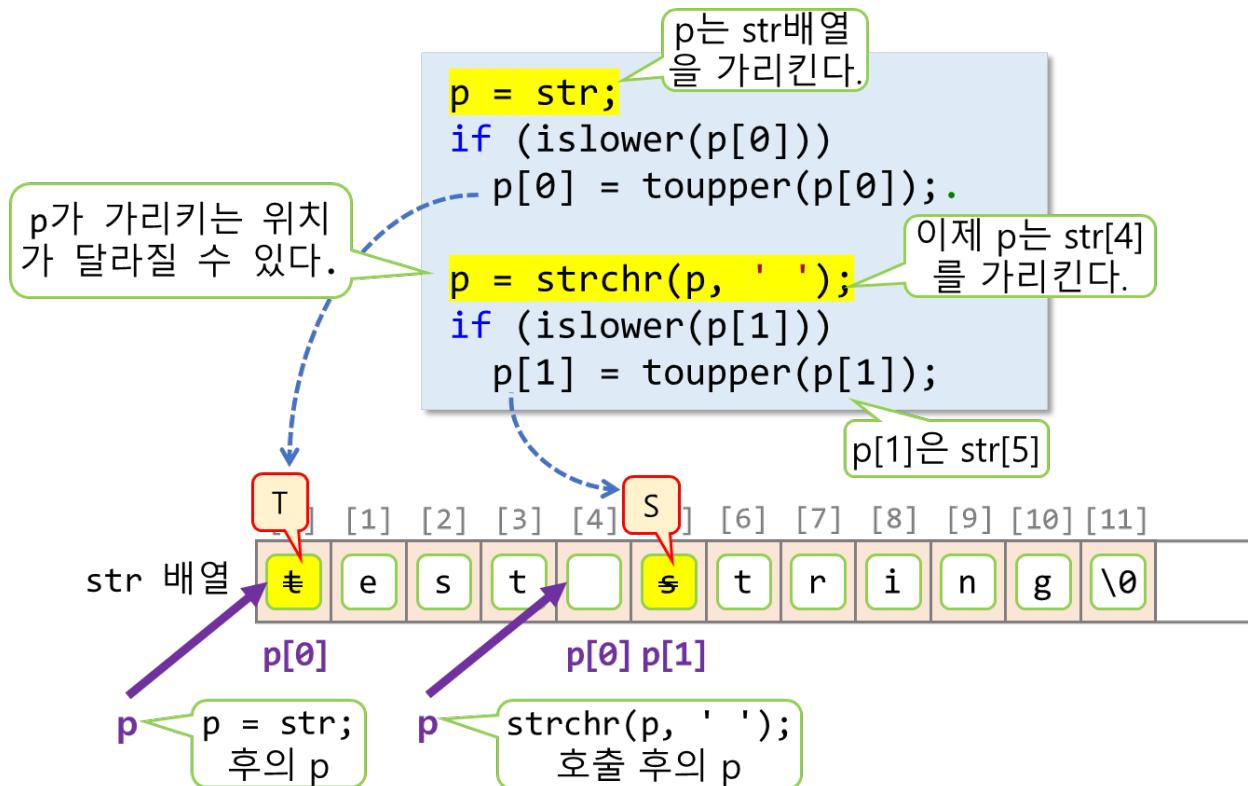
- 문자열 포인터가 가리키는 문자 배열을 변경할 수 있다.

```
strcpy(p, "test string");
```

p가 가리키는 str을
변경한다.

문자 배열_문자열 포인터

- 문자열 포인터를 이용하면 문자 배열의 특정 위치를 가리키도록 문자열 포인터를 변경해 가면서 문자열에 대한 처리를 할 수 있다.



문자 배열_문자열 포인터

❖ char*형 포인터의 용도

```
05 int main(void)
06 {
07     char str[64] = "this is test string for char pointer.";
08     char* p = str;
09
10    while (1) {
11        if (islower(p[0]))
12            p[0] = toupper(p[0]);
13        p = strchr(p, ' ');
14        if (p == NULL)
15            break;
16        p++;
17    }
18    puts(str);
19 }
```

p[0]이 소문자면
대문자로 변경한다.

str중 ''문자를 찾아 그
주소를 p에 저장한다.

p가 '' 문자 다음 문자
를 가리키게 한다.

실행 결과

This Is Test String For Char Pointer.

문자 배열_문자열 포인터

❖ const char*형의 문자열 포인터(1/)

■ 읽기 전용의 문자열 포인터

- 주로 문자열 리터럴을 가리키는 용도로 사용된다.
- 문자 배열을 가리킬 때는 문자 배열의 내용을 읽어볼 수만 있고 변경할 수 없다.

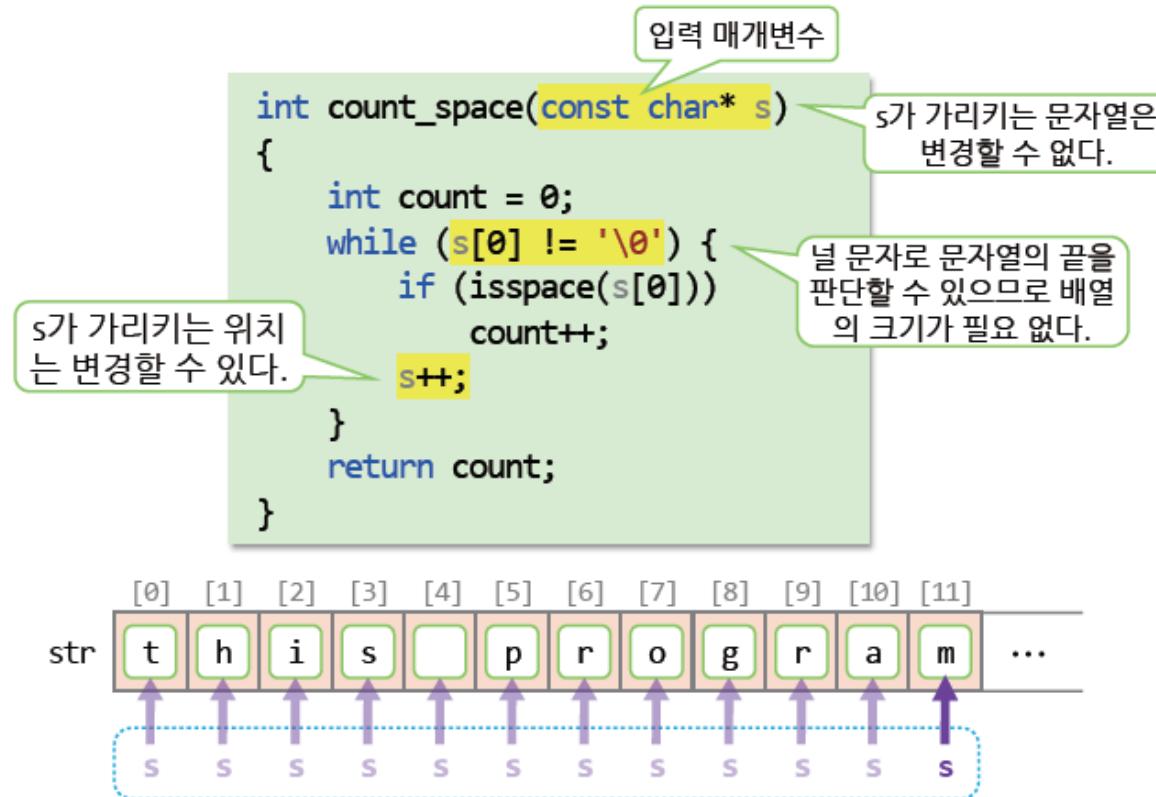
```
const char* p = "abcde";  
    ❌ p[0] = 'A';      → p가 가리키는 문자열 리터럴을  
    ❌ strcpy(p, "xyz");          변경할 수 없다.
```

```
char str[64] = "";  
const char* p = str;  
    ❌ p[0] = 'A';      → p가 가리키는 문자 배열을  
    ❌ strcpy(p, "xyz");          변경할 수 없다.
```

문자 배열_문자열 포인터

❖ const char*형의 문자열 포인터(2/)

- 문자열을 입력 매개변수로 전달할 때 유용하다.



문자 배열_문자열 포인터

❖ count_space 함수의 정의 및 호출 예(1/2)

```
01 #include <stdio.h>
02 #include <string.h>
03 #include <ctype.h>
04
05 int count_space(const char* s) {
06
07     int count = 0;
08
09     while (s[0] != '\0') {
10
11         if (isspace(s[0]))
12             count++;
13
14     }
15 }
```

s는 함수 안에서 변경되지 않는 입력 매개변수이다.

문자 배열_문자열 포인터

❖ count_space 함수의 정의 및 호출 예(2/2)

```
15  
16 int main(void)  
17 {  
18     char str[64] = "this program\ntests const pointer to string.\n";  
19  
20     puts(str);  
21     printf("공백 문자의 개수: %d\n", count_space(str));  
22 }
```

실행 결과

this program tests const pointer to string.

공백 문자의 개수: 7

문자 배열_문자열 포인터

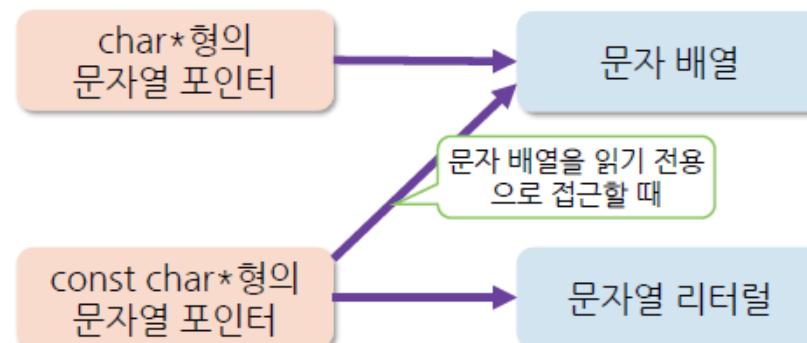
❖ 문자열 사용을 위한 가이드라인(1/3)

▪ 문자열을 어떤 형의 변수에 저장할지 선택하는 기준

- ① 사용자로부터 입력받거나 변경할 수 있는 문자열은 문자 배열에 저장
→ 문자열 변수
- ② 프로그램 실행 중에 변경되지 않는 문자열은 문자열 리터럴로 나타낸다.
→ 문자열 상수

▪ 어떤 형의 문자열 포인터를 사용할지 선택하는 기준

- ① `char*`형의 포인터는 문자 배열, 즉 변경할 수 있는 문자열을 가리킬 때만 사용
- ② `const char*`형의 포인터는 변경할 수 없는 문자열을 가리킬 때 사용
- ③ 문자열 리터럴을 가리킬 때는 `const char*`형의 포인터를 사용
- ④ 문자 배열을 읽기 전용으로 접근할 때는 `const char*`형의 포인터를 사용



문자 배열_문자열 포인터

❖ 문자열 사용을 위한 가이드라인(2/3)

- 문자열을 매개변수로 갖는 함수를 정의할 때의 주의 사항
 - ① 문자열이 출력 매개변수일 때는 `char*`형의 매개변수를 사용하고 문자 배열의 크기도 매개변수로 받아와야 한다. 함수 안에서 문자열을 변경할 때는 문자 배열의 크기를 넘어서지 않도록 주의해야 한다.
 - ② 문자열이 입력 매개변수일 때는 `const char*`형의 매개변수를 사용한다. 이때는 문자열의 끝을 널 문자로 확인할 수 있으므로 문자 배열의 크기를 매개변수로 받아올 필요가 없다.
 - ③ 문자열을 사용할 때는 문자 배열처럼 인덱스를 사용할 수 있다.

문자 배열_문자열 포인터

❖ 문자열 사용을 위한 가이드라인(3/3)

- 문자열을 매개변수로 갖는 함수를 호출할 때의 주의 사항
 - ① 매개변수의 데이터형이 **char***형일 때는 문자 배열과 char*형의 포인터만 인자로 전달할 수 있다. 함수 호출 후 인자로 전달된 문자열의 내용이 변경될 수 있다.
 - ② 매개변수의 데이터형이 **const char***형일 때는 문자 배열, 문자열 리터럴, char*형의 포인터, const char*형의 포인터를 모두 인자로 전달할 수 있다. 함수 호출 후에도 인자로 전달된 문자열의 내용은 달라지지 않는다.

문자 배열_문자열의 배열

❖ 2차원 문자 배열

- 입력된 문자열이나 실행 중에 변경되는 문자열을 여러 개 저장하려면 2차원 문자 배열을 사용한다.

❖ 문자열 포인터의 배열

- 변경되지 않는 문자열을 여러 개 저장하려면 문자열 포인터의 배열을 사용한다.
- 문자열 리터럴의 주소만 배열로 저장한다.

문자 배열_문자열의 배열

❖ 2차원 문자 배열의 선언 및 초기화

- 열 크기 : 널 문자를 포함한 문자열의 길이
- 행 크기 : 문자열의 개수

```
char books[5][30];
```

길이가 30인 문자열을
5개 저장한다.

- { } 안에 문자열을 나열해서 초기화한다.

```
char books[5][30] = {  
    "wonder",  
    "me before you",  
    "the hunger games",  
    "twilight",  
    "harry potter",  
};
```

문자열의
개수

문자열의
길이

행 인덱스만 사용
하면 문자열 하나
에 접근한다.

books[0]

books[1]

books[2]

books[3]

books[4]

"wonder"

"me before you"

"the hunger games"

"twilight"

"harry potter"

char[30]

char[30]

char[30]

char[30]

char[30]

문자 배열_문자열의 배열

❖ 2차원 문자 배열의 사용

- 2차원 문자 배열의 각 문자열에 접근하려면 행 인덱스만 사용한다.

```
for (i = 0; i < 5; i++)  
    printf("%s\n", books[i]);
```

i번째 문자열을 출력한다.

- i번째 문자열의 j번째 문자에 접근하려면 books[i][j]처럼 행 인덱스와 열 인덱스를 모두 사용한다.

```
for (i = 0; i < 5; i++) {  
    if (islower(books[i][0]))  
        books[i][0] = toupper(books[i][0]);  
}
```

i번째 문자열의 0번째 문자가 소문자인지 검사한다.

문자 배열_문자열의 배열

❖ 2차원 문자 배열의 사용 예

```
01 #include <stdio.h>
02 #include <string.h>
03 #include <ctype.h>
04
05 int main(void)
06 {
07     char books[5][30] = {
08         "wonder", "me before you", "the hunger games", "twilight", "harry potter"
09     };
10     int i = 0;
11
12     for (i = 0; i < 5; i++)
13         printf("%s\n", books[i]);
```

i번째 문자열을 출력한다.

문자 배열_문자열의 배열

```
15     for (i = 0; i < 5; i++) {  
16         if (islower(books[i][0]))  
17             books[i][0] = toupper(books[i][0]);  
18     }  
19  
20     puts("« 변경 후 »");  
21     for (i = 0; i < 5; i++)  
22         printf("%s\n", books[i]);  
23 }
```

i번째 문자열의 0번째 문자가 소문자면
대문자로 변경한다.

실행 결과

```
wonder  
me before you  
the hunger games  
twilight  
harry potter  
« 변경 후 »  
Wonder  
Me before you  
The hunger games  
Twilight  
Harry potter
```

문자 배열_문자열의 배열

❖ 문자열 포인터 배열 : 문자열 리터럴을 모아두면 문자열 리터럴을 관리하기가 쉬워진다.

- **const char*형의 포인터 배열**
- 문자열 리터럴의 주소만 저장하는 배열
- 각각의 문자열을 읽기 전용으로 접근
- **str_menu[i]** : i번째 문자열 리터럴



문자 배열_문자열의 배열

❖ 문자열 포인터 배열의 사용

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 #include <string.h>
04
05 int main(void)
06 {
07     const char* str_menu[] = {
08         "끝내기", "새로 만들기", "파일 열기", "파일 저장", "인쇄"
09     };
10     int sz_menu = sizeof(str_menu) / sizeof(str_menu[0]);
11     int menu;
```

실행 결과

- 0. 끝내기
- 1. 새로 만들기
- 2. 파일 열기
- 3. 파일 저장
- 4. 인쇄

메뉴 선택? 1

새로 만들기 메뉴를 선택했습니다.

문자 배열_문자열의 배열

```
13     while (1) {  
14         int i;  
15         for (i = 0; i < sz_menu; i++)  
16             printf("%d.%s\n", i, str_menu[i]);  
17         printf("메뉴 선택? ");  
18         scanf("%d", &menu);  
19         if (menu == 0)  
20             break;  
21         else if (menu > 0 && menu < sz_menu)  
22             printf("%s 메뉴를 선택했습니다.\n\n", str_menu[menu]);  
23         else  
24             printf("잘못 선택했습니다.\n\n");  
25     }  
26 }
```

str_menu[i]가 가리키는 문자열 출력