문자열

목차

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', '\012'
  - 문자 변수 : char형 변수, ASCII 코드 저장

문자 배열의 선언 및 초기화 (1/3)

- 문자 배열의 크기
  - 저장할 문자열의 길이 + 1

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

길이가 9인 문자열 저장

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

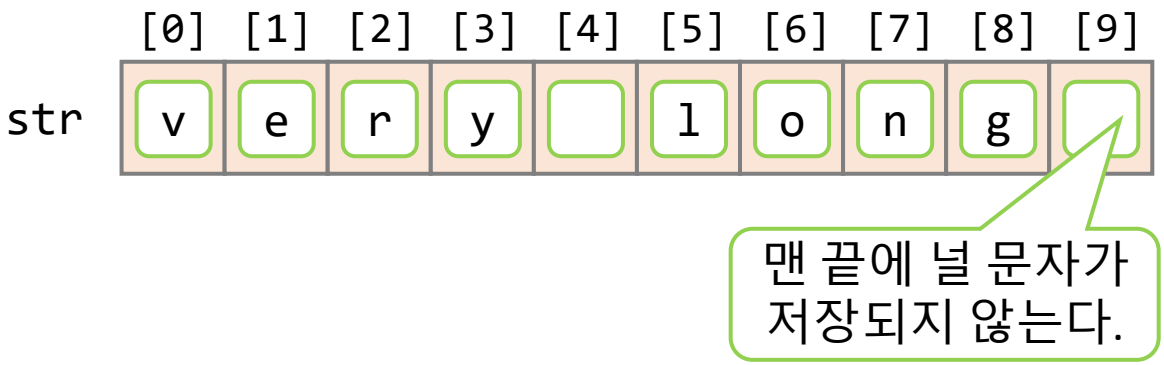
```
char str[10] = "abc";
```

문자 배열의 선언 및 초기화 (2/3)

- 문자열 리터럴의 길이가 '문자 배열의 크기-1' 보다 크면 컴파일 경고가 발생

배열 크기만큼만 초기화한다.

```
char str[10] = "very long string";
```



문자 배열의 선언 및 초기화 (3/3)

- 초기값을 지정할 때는 문자 배열의 크기를 생략할 수 있다.

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

크기가 6인 배열

- 문자 배열 전체를 널 문자로 초기화하려면 널 문자열로 초기화한다.

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

널 문자열

문자 배열의 사용 (1/2)

- 인덱스를 이용해서 배열의 원소에 접근할 수 있다.

```
for (i = 0; i < size; i++)
    printf("%c", str[i]);
```

한 문자씩 출력

```
printf("%s", str);
printf(str);
```

문자열 전체 출력

```
str[0] ='A';
```

한 문자씩 변경

예제 9-1 : 문자 배열의 초기화 및 출력 (1/2)

```
03  int main(void)
04  {
05      char str1[10] = { 'a', 'b', 'c' };
06      char str2[10] = "abc";
07      char str3[] = "abc";       // str3은 크기가 4인 배열
08      char str4[10] = "very long string";// 컴파일 경고
09      int size = sizeof(str1) / sizeof(str1[0]);
10      int i;
11
12      printf("str1 = ");
13      for (i = 0; i < size; i++)
14          printf("%c", str1[i]);          // 배열처럼 for문으로 출력할 수 있다.
15      printf("\n");
16
17      printf("str2 = %s\n", str2);        // 문자열을 %s로 출력한다.
```

예제 9-1 : 문자 배열의 초기화 및 출력 (2/2)

```
19      printf("str3 = ");
20      printf(str3);    // 문자 배열을 직접 printf 함수의 인자로 전달할 수 있다.
21      printf("\n");
22
23      printf("str4 = %s\n", str4);
24
25      return 0;
26  }
```

실행결과

```
str1 = abc
str2 = abc
str3 = abc
str4 = very long 傲傲傲傲傲
```

str4의 맨 끝에 널문자가 없으므로 문자열의 끝을 감지할 수 없다.

문자 배열의 사용 (2/2)

- 문자 배열에 직접 다른 문자열을 대입해서는 안 된다.

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

배열의 시작 주소는 변경할 수 없으므로 컴파일 에러

- 인덱스의 유효 범위를 넘어서지 않아야 한다.

```
str[20] ='A';
```

유효 범위가 아닌 인덱스를 사용하면 실행 에러

표준 C 문자열 처리 함수

문자열 처리 함수	설명
strlen(str);	str의 길이를 구한다. (널 문자 제외)
strcmp(lhs, rhs);	lhs와 rhs를 비교해서 같으면 0을, lhs > rhs면 0보다 큰 값을, lhs < rhs면 0보다 작은 값을 리턴한다.
strncmp(lhs, rhs, cnt);	lhs와 rhs를 cnt개만큼 비교한다. 리턴 값은 strcmp와 같다.
strcpy(dest, src);	src를 dest로 복사한다.
strncpy(dest, src, cnt);	src를 dest로 cnt개만큼 복사한다.
strcat(dest, src);	dest의 끝에 src를 연결한다.
strncat(dest, src, cnt);	dest의 끝에 src를 cnt개 연결한다.
strchr(str, ch);	str에서 ch 문자를 찾는다.
strstr(str, substr);	str에서 substr 문자열을 찾는다.
strtok(str, delim);	str을 delim을 이용해서 토큰으로 분리한다.

표준 C 문자 처리 함수

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

표준 C 데이터 변환 함수

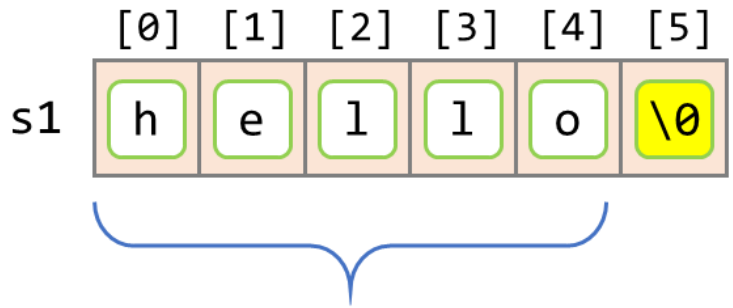
헤더 파일	함수 원형	설명
<stdlib.h>	int atoi(const char *str);	문자열을 정수로 변환한다.
	double atof(const char *str);	문자열을 실수로 변환한다.
	long atol(const char *str);	문자열을 long형 값으로 변환한다.
<stdio.h>	int sscanf(const char *buff, const char *format, ...);	문자열로부터 정수나 실수를 읽어 온다.
	int sprintf(char *buff, const char* format, ...);	정수나 실수를 형식 문자열을 이용해서 문자열로 만든다.

문자열의 길이 구하기

- 널 문자를 제외한 문자열의 길이

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

```
char s1[] = "hello";
printf("s1의 길이: %d\n", strlen(s1));
```




```
len = strlen(s1);
if (len > 0)
    s1[len - 1] = '\0';
```

마지막 한 글자를 삭제한다.

예제 9-2 : strlen 함수 사용 예 (1/2)

```c
#include <stdio.h>
#include <string.h>   // 문자열 처리 함수 사용 시 포함

int main(void)
{
    char s1[] = "hello";
    char s2[] = ""; // 널 문자열
    int len = 0;

    printf("s1의 길이: %d\n", strlen(s1));          // 널 문자를 제외한 문자열의 길이
    printf("s2의 길이: %d\n", strlen(s2));          // 널 문자열의 길이
    printf("s2의 길이: %d\n", strlen("bye bye"));   // 문자열 리터럴의 길이

    printf("s1의 크기 : %d\n", sizeof(s1));       // 널 문자를 포함한 배열의 크기

```

예제 9-2 : strlen 함수 사용 예 (2/2)

```
16      len = strlen(s1);
17      if (len > 0)
18          s1[len - 1] = '\0';    // 마지막 한 글자를 삭제한다.
19      printf("s1 = %s\n", s1);
20
21      return 0;
22  }
```

실행결과

```
s1의 길이: 5
s2의 길이: 0
s2의 길이: 7
s1의 크기 : 6
s1 = hell
```

strlen 함수를 이용해서
문자열의 마지막 1글자를 삭제한다.

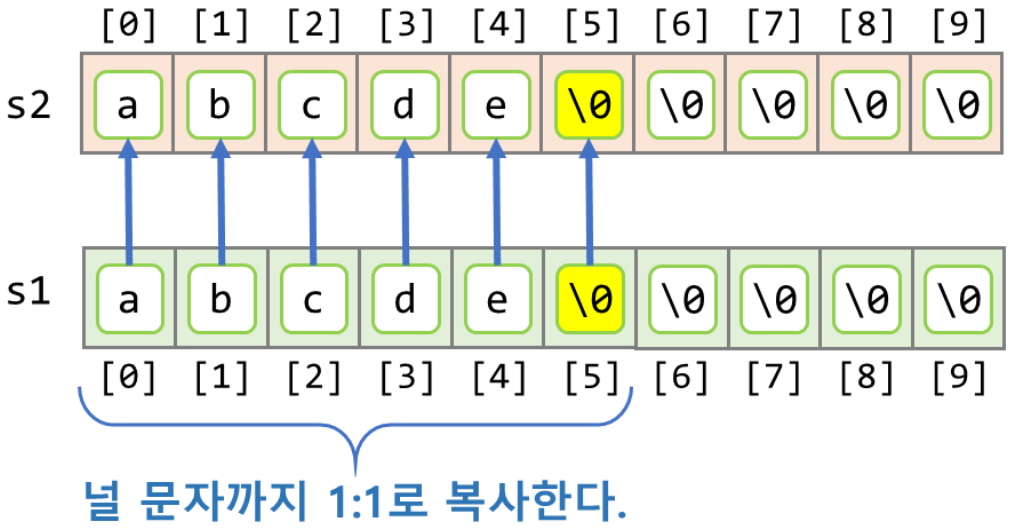
문자열의 복사 (1/2)

- src 문자열을 dest 문자 배열로 복사한다.

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

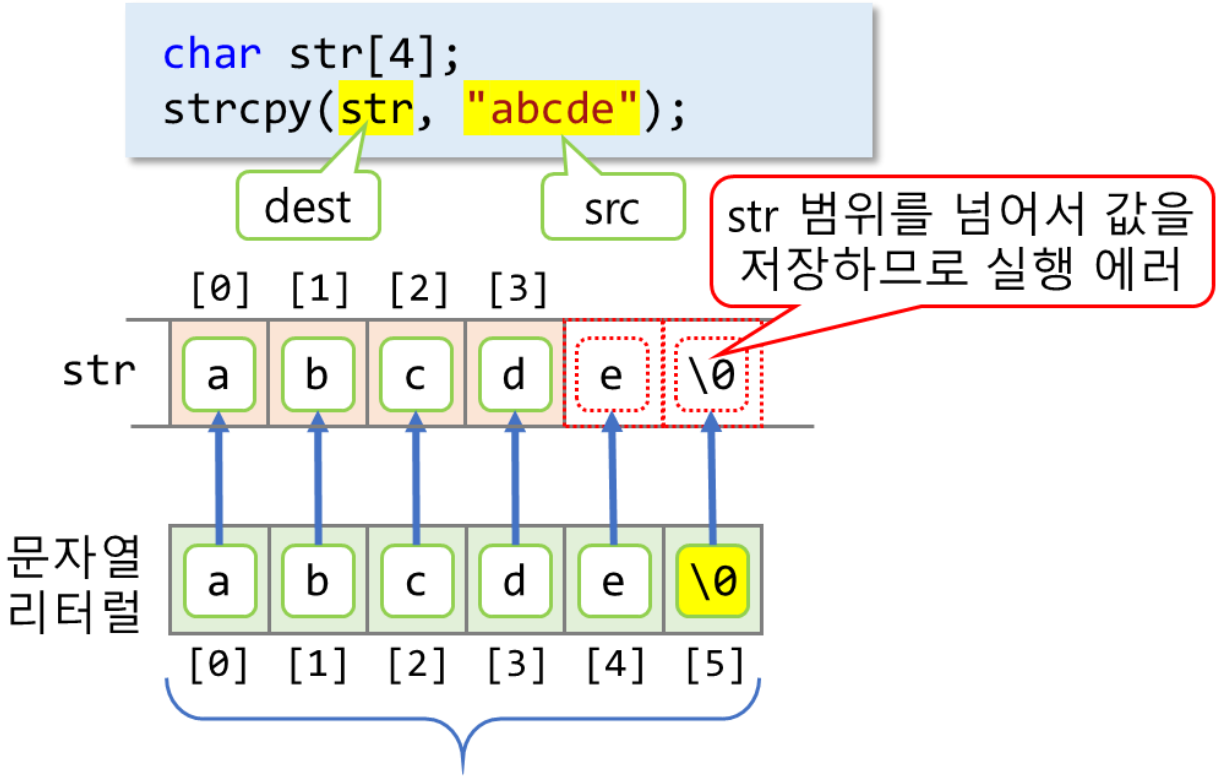
```
char s1[10] = "abcde";
char s2[10] = "";
strcpy(s2, s1);
```

dest src



문자열의 복사 (2/2)

- dest에 src를 복사할만큼 메모리가 충분한지 검사하지 않는다.



예제 9-3 : 문자열의 교환 (1/2)

```
01  #define _CRT_SECURE_NO_WARNINGS        // 라이브러리 헤더 앞에 정의한다.
02  #include <stdio.h>
03  #include <string.h>                    // 문자열 처리 함수 사용 시 포함
04  #define SIZE 32
05
06  int main(void)
07  {
08      char str1[SIZE] = "";      // 널 문자열로 초기화한다.
09      char str2[SIZE] = "";      // 널 문자열로 초기화한다.
10      char temp[SIZE];
11
12      printf("2개의 문자열? ");
13      scanf("%s %s", str1, str2);          // 빈칸으로 구분해서 문자열 입력
14      printf("str1 = %s, str2 = %s\n", str1, str2);
15
```

예제 9-3 : 문자열의 교환 (2/2)

```
16      // 두 문자 배열을 swap한다.
17      strcpy(temp, str1);         // str1을 temp로 복사한다.
18      strcpy(str1, str2);         // str2을 str1로 복사한다.
19      strcpy(str2, temp);         // temp을 str2로 복사한다.
20      printf("str1 = %s, str2 = %s\n", str1, str2);
21      return 0;
22  }
```

실행결과

```
2개의 문자열? apple orange
str1 = apple, str2 = orange
str1 = orange, str2 = apple
```

문자열의 비교 (1/2)

- lhs 문자열과 rhs 문자열을 알파벳 순으로 비교한다.
  - lhs와 rhs가 같으면 0을, lhs가 rhs보다 알파벳 순으로 앞쪽이면 음수를, 뒤쪽이면 양수를 리턴한다.

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

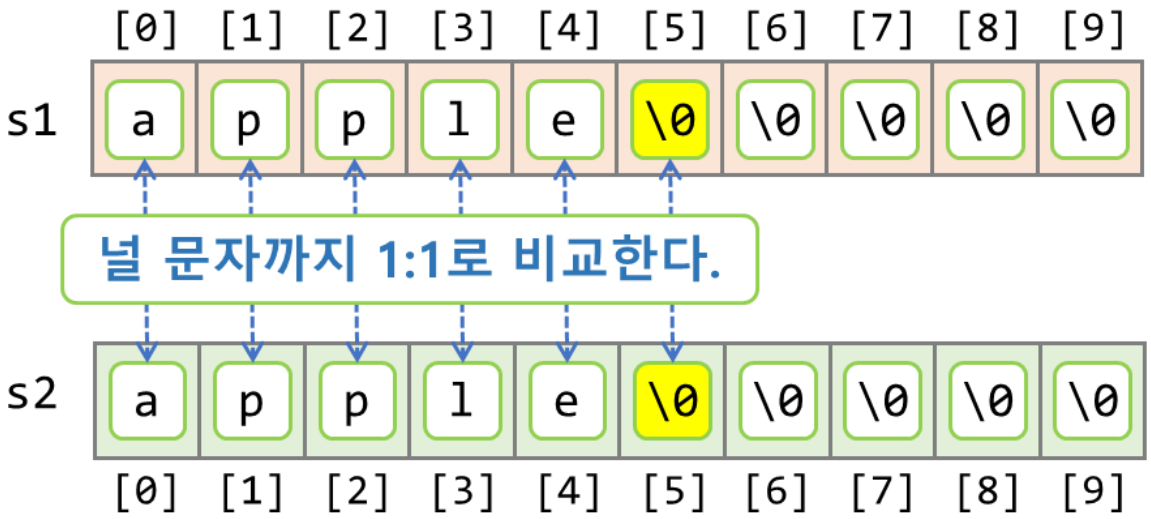
```
char s1[SIZE] = "apple";
char s2[SIZE] = "apple";

if (s1 == s2)
    printf("same address\n");
```

s1과 s2의 주소를 비교한다.

문자열의 비교 (2/2)

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

s1과 s2의 내용을 비교한다.



예제 9-4 : 문자열의 비교 (1/2)

```c
#define _CRT_SECURE_NO_WARNINGS        // 라이브러리 헤더 앞에 정의한다.
#include <stdio.h>
#include <string.h>                    // 문자열 처리 함수 사용 시 포함
#define SIZE 10

int main(void)
{
    char s1[SIZE] = "apple";
    char s2[SIZE] = "apple";
    char password[SIZE];

    if (s1 == s2)                      // s1의 주소와 s2의 주소를 비교하면 안된다.
        printf("same address\n");

    if (strcmp(s1, s2) == 0)           // s1과 s2의 내용을 비교한다.
        printf("same string\n");
```

예제 9-4 : 문자열의 비교 (2/2)

```
18      printf("패스워드? ");
19      scanf("%s", password);                  // 패스워드를 입력받는다.
20      if (strcmp(password, "abcd1234") == 0)  // 등록된 패스워드와 비교한다.
21          printf("login succeeded\n");
22      else
23          printf("login failed\n");
24
25      return 0;
26  }
```

실행결과

```
same string
패스워드? abcd1234
login succeeded
```

s1과 s2는 주소는 다르지만 문자열의 내용은 같다.

입력받은 패스워드 문자열을 등록된 문자열과 비교한다.

문자열의 연결 (1/2)

- dest 문자열의 끝에 src 문자열을 복사해서 연결한다.

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

dest에 src를 연결할 만큼 배열의 크기가 충분한지 확인 후 호출해야 한다.

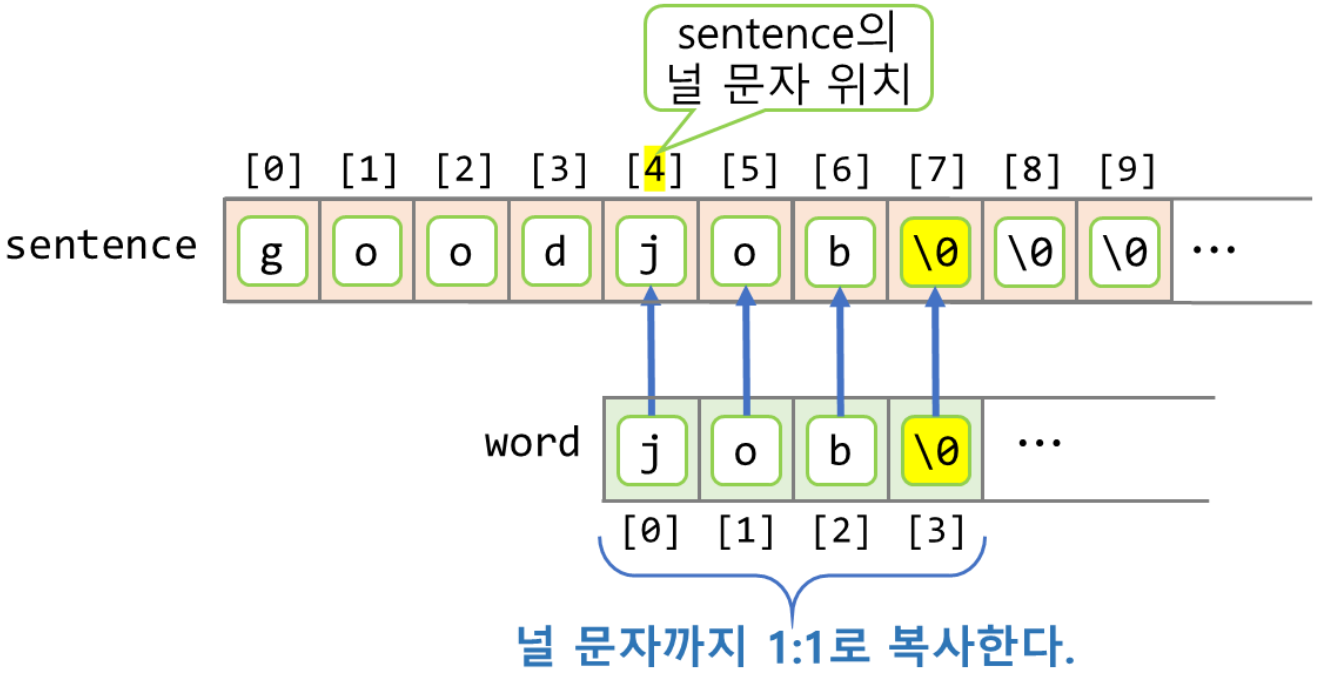
문자열의 연결 (2/2)

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

입력받은 단어를 문장의 끝에 붙인다.

dest

src



예제 9-5 : 문자열의 연결

```
05  int main(void)
06  {
07      char sentence[100] = "";
08      char word[20];
09
10      do {
11          printf("단어? ");
12          scanf("%s", word);
13          strcat(sentence, word);          // 입력받은 단어를 문장 끝에 붙인다.
14          strcat(sentence, " ");           // 단어를 구분할 수 있도록 " "을 붙인다.
15      } while (strcmp(word, ".") != 0);    // "."이 입력될 때까지 반복한다.
16
17      printf("%s\n", sentence);
18
19      return 0;
20  }
```

실행결과

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

문자열의 검색 (1/3)

- str에서 ch 문자가 있는지 찾는다.

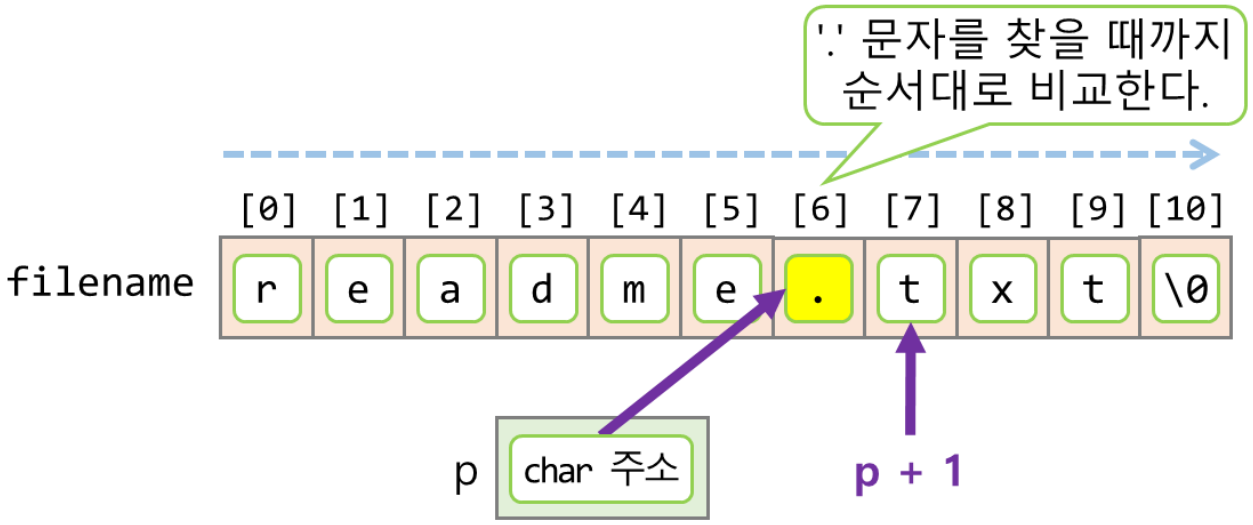
```
char *strchr(const char *str, int ch);
```

  - 찾은 위치에 있는 문자의 주소 리턴
  - str에서 ch를 찾을 수 없으면 NULL 리턴

문자열의 검색 (2/3)

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

'.' 다음에 있는 파일 확장자를 출력한다.



찾은 문자의 주소를 리턴한다.

문자열의 검색 (3/3)

- str에서 substr 문자열이 있는지 찾는다.

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

  - 찾은 위치에 있는 문자의 주소 리턴
  - str에서 ch를 찾을 수 없으면 NULL 리턴

```
char filename[] = "readme.txt";
char *p = NULL;
p = strstr(filename, ".txt");
if (p != NULL)
  printf("file type: TEXT file\n");
```

예제 9-6 : 문자열의 검색

```
05  int main(void)
06  {
07      char filename[] = "readme.txt";
08      char *p = NULL;
09
10      p = strchr(filename, '.');
11      if (p != NULL)
12          printf("file extension: %s\n", p + 1);
13
14      p = strstr(filename, ".txt");
15      if (p != NULL)
16          printf("file type: TEXT file\n");
17
18      return 0;
19  }
```

실행결과

```
file extension: txt
file type: TEXT file
```

문자열의 토큰 나누기 (1/2)

- 토큰 : 어떤 문장에서 더 이상 나눌 수 없는 최소 단위
- str을 delim에 있는 문자들을 이용해서 토큰으로 쪼갠다.

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

  - 토큰의 주소를 리턴
  - 더 이상 토큰이 없으면 NULL 리턴
  - 함수 호출 후 첫 번째 매개변수인 str이 변경
- 첫 번째 strtok 함수 호출 후에 이전 문자열에서 다음 토큰을 구하려면 strtok 함수의 첫 번째 인자로 NULL을 지정한다.

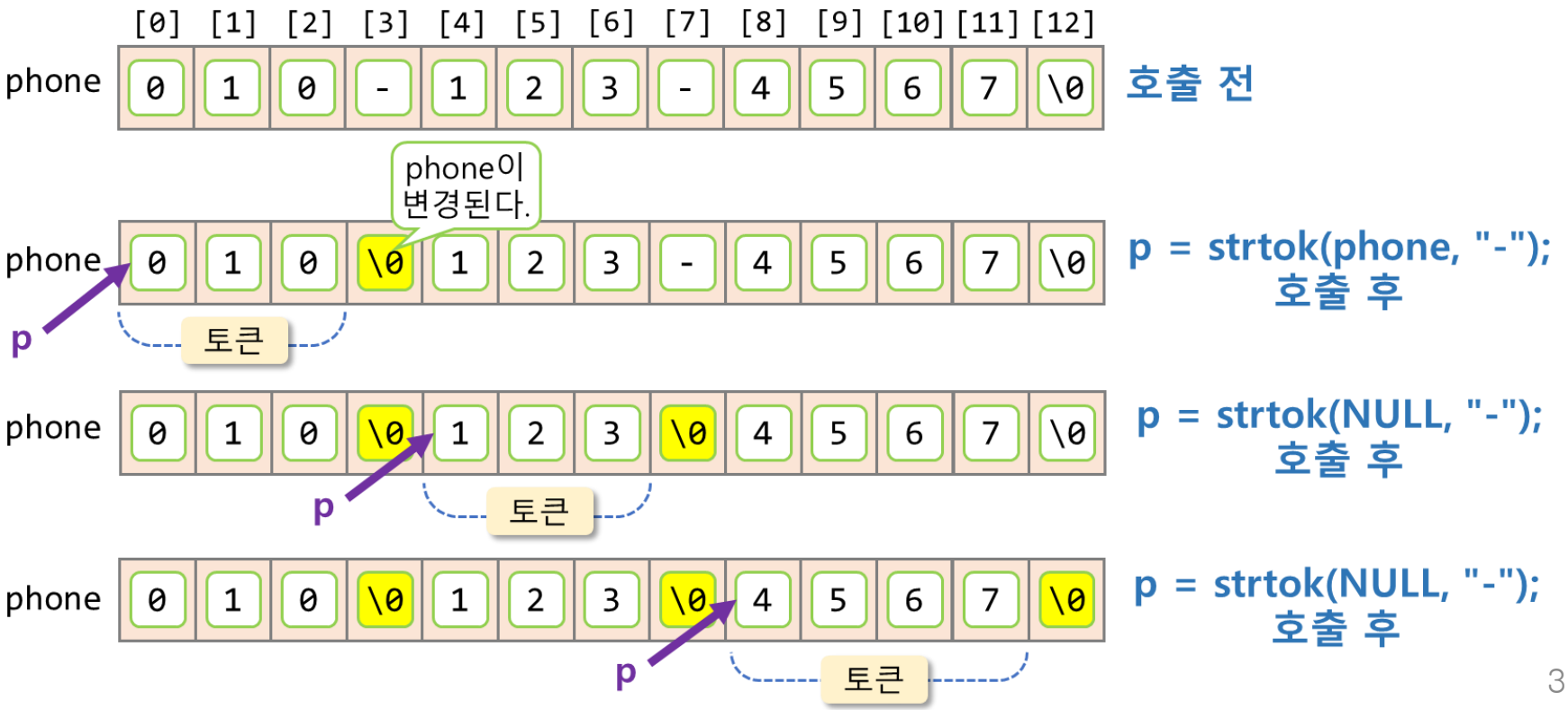
문자열의 토큰 나누기 (2/2)

```
char phone[] = "02-123-4567";
char *p = NULL;
p = strtok(phone, "-");
p = strtok(NULL, "-");
p = strtok(NULL, "-");
```

첫 번째 토큰 리턴

두 번째 토큰 리턴

세 번째 토큰 리턴



예제 9-7 : 문자열의 토큰 나누기

```
05  int main(void)
06  {
07      char phone[] = "02-123-4567";
08      char *p = NULL;
09
10      p = strtok(phone, "-");
11      printf("area code: %s\n", p);
12      p = strtok(NULL, "-");
13      printf("prefix: %s\n",p);
14      p = strtok(NULL, "-");
15      printf("line number: %s\n", p);
16
17      return 0;
18  }
```

실행결과

```
area code: 02
prefix: 123
line number: 4567
```

표준 C 문자열 입출력 함수

문자열 처리 함수	설명
scanf("%s", str);	공백 문자까지 문자열을 입력받아서 str에 저장한다.
printf(str); printf("%s", str);	str을 출력한다.
gets_s(str, count);	한 줄의 문자열을 읽어 줄바꿈 문자를 빼고 str에 저장한다.
fgets(str, count, stdin);	줄바꿈 문자를 포함한 한 줄의 문자열을 읽어서 str에 저장한다.
puts(str);	str과 줄바꿈 문자를 출력한다.
sscanf(str, "형식문자열", …);	str에서 형식 문자열에 지정된 대로 값을 읽어 온다.
sprintf(str, "형식문자열", …);	str을 형식 문자열에 지정된 대로 만든다.

문자열의 입력

- 빈칸을 포함한 문자열 입력

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

```
char str[128];
fgets(str, sizeof(str), stdin);          // 줄바꿈 문자까지를 str로 읽어 온다.
printf(str);          // str에 줄바꿈 문자가 포함되어 있으므로 출력 후 줄이 바뀐다.
```

```
char str[128];
gets_s(str, sizeof(str));       // 줄바꿈 문자까지 읽어 줄바꿈 문자를 빼고 str로 읽어 온다.
printf(str);          // str에 줄바꿈 문자가 포함되어 있지 않으므로 출력 후 줄이 바뀌지 않는다.
```

문자열의 출력

- 한 줄의 문자열 출력
  - 문자열 출력 시 줄바꿈 문자를 함께 출력

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

```
puts("hello there");          // "hello there\n"를 출력한다.
```

문자열 변환

- 형식 문자열을 이용해 문자열을 변환

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

```
char str[128];
int n;
gets_s(str, sizeof(str));       // 줄바꿈 문자까지를 str로 읽어 온다.
sscanf(str, "%d", &n);          // str에서 정수를 읽어서 n에 저장한다.
```

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

```
sprintf(out_str, "%02d:%02d:%02d", hour, min, sec);
puts(out_str);
```

예제 9-8 : 문자열의 입출력

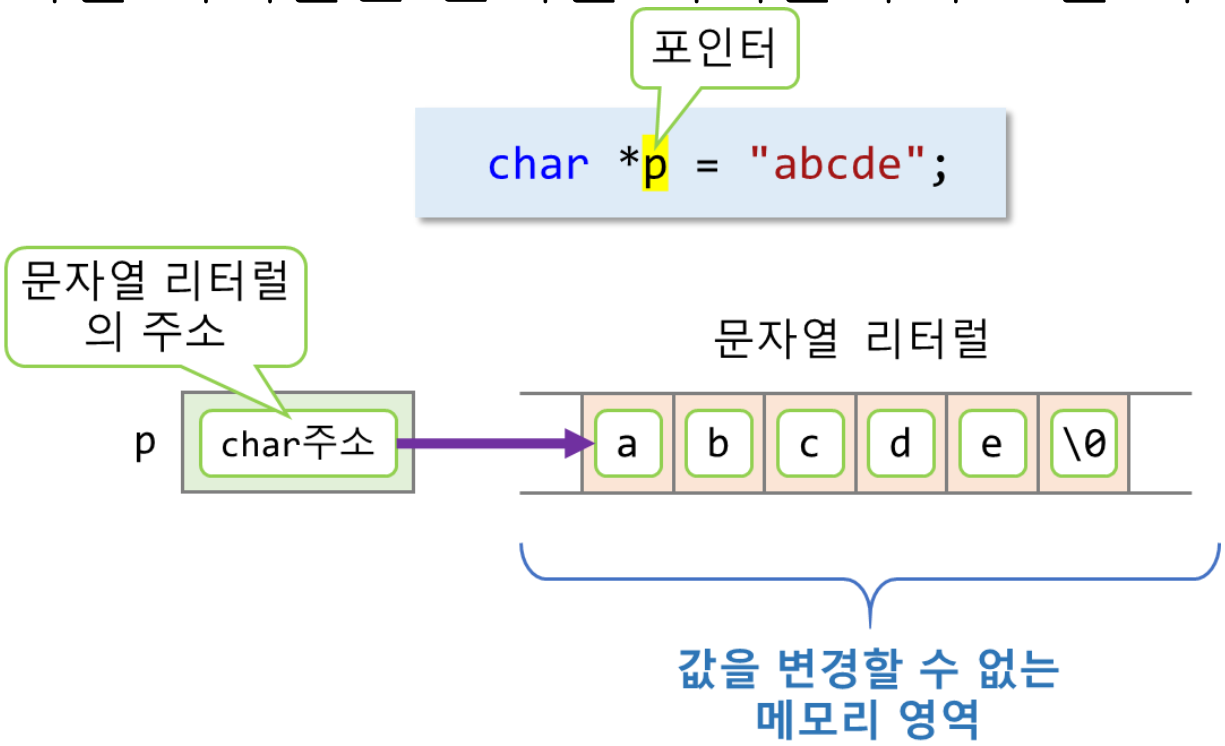
```
05  int main(void)
06  {
07      char in_str[128];
08      char out_str[128];
09      int hour = 12, min = 30, sec = 45;
10
11      printf("문자열? ");
12      gets_s(in_str, sizeof(in_str));      // 빈칸을 포함한 문자열 입력
13      puts(in_str);    // 문자열과 줄바꿈 문자를 함께 출력한다.
14      sprintf(out_str, "%02d:%02d:%02d", hour, min, sec);  // 문자열을 만든다.
15      puts(out_str);
16      return 0;
17  }
```

실행결과

```
문자열? gets_s can read string including spaces.
gets_s can read string including spaces.
12:30:45
```

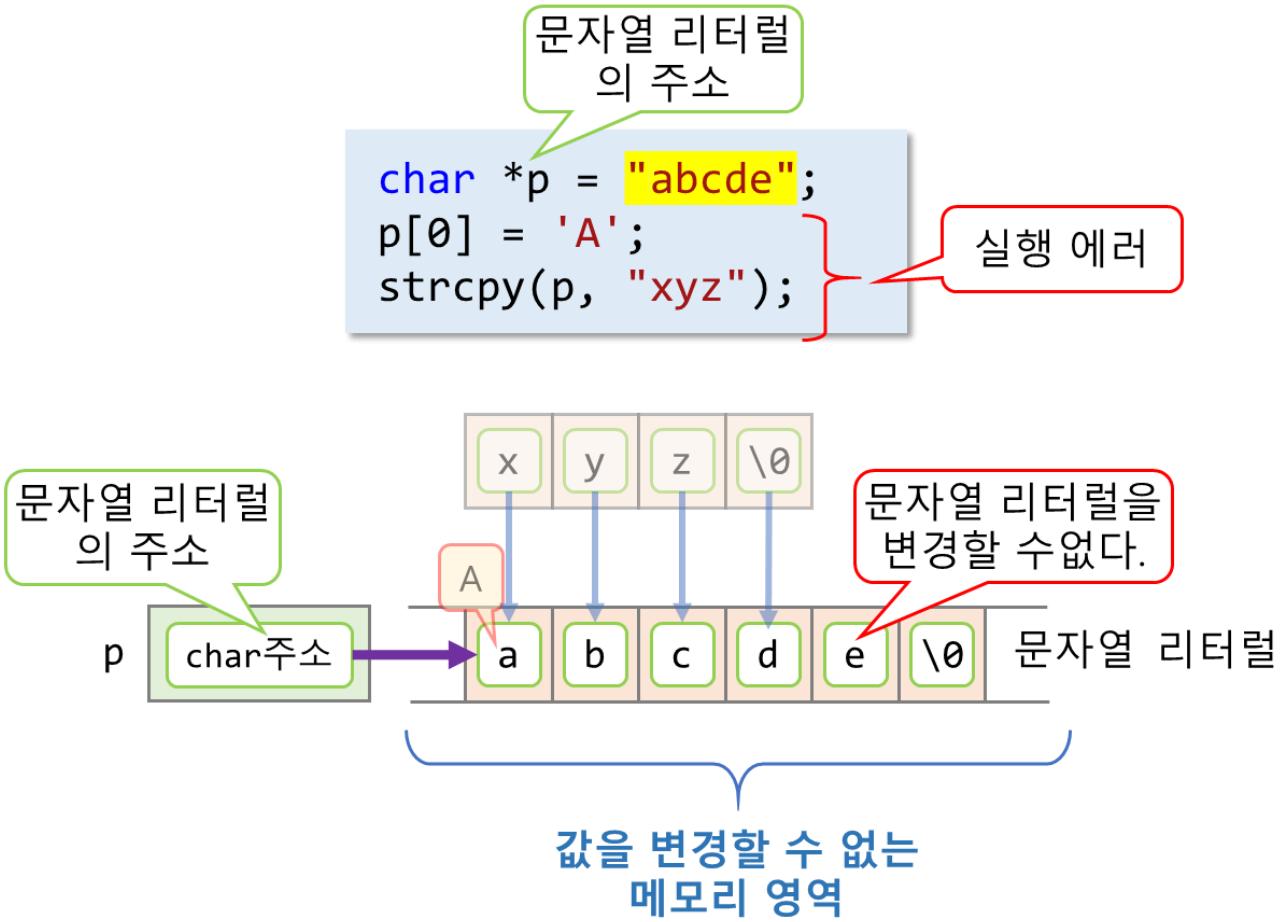
문자열 리터럴 (1/3)

- 문자열 리터럴은 예외적으로 메모리에 할당된다.
  - 텍스트 세그먼트라는 특별한 메모리 영역에 문자열 리터럴을 보관하고 그 주소를 대신 사용한다.
  - 문자열 리터럴은 문자열 리터럴의 주소를 의미한다.



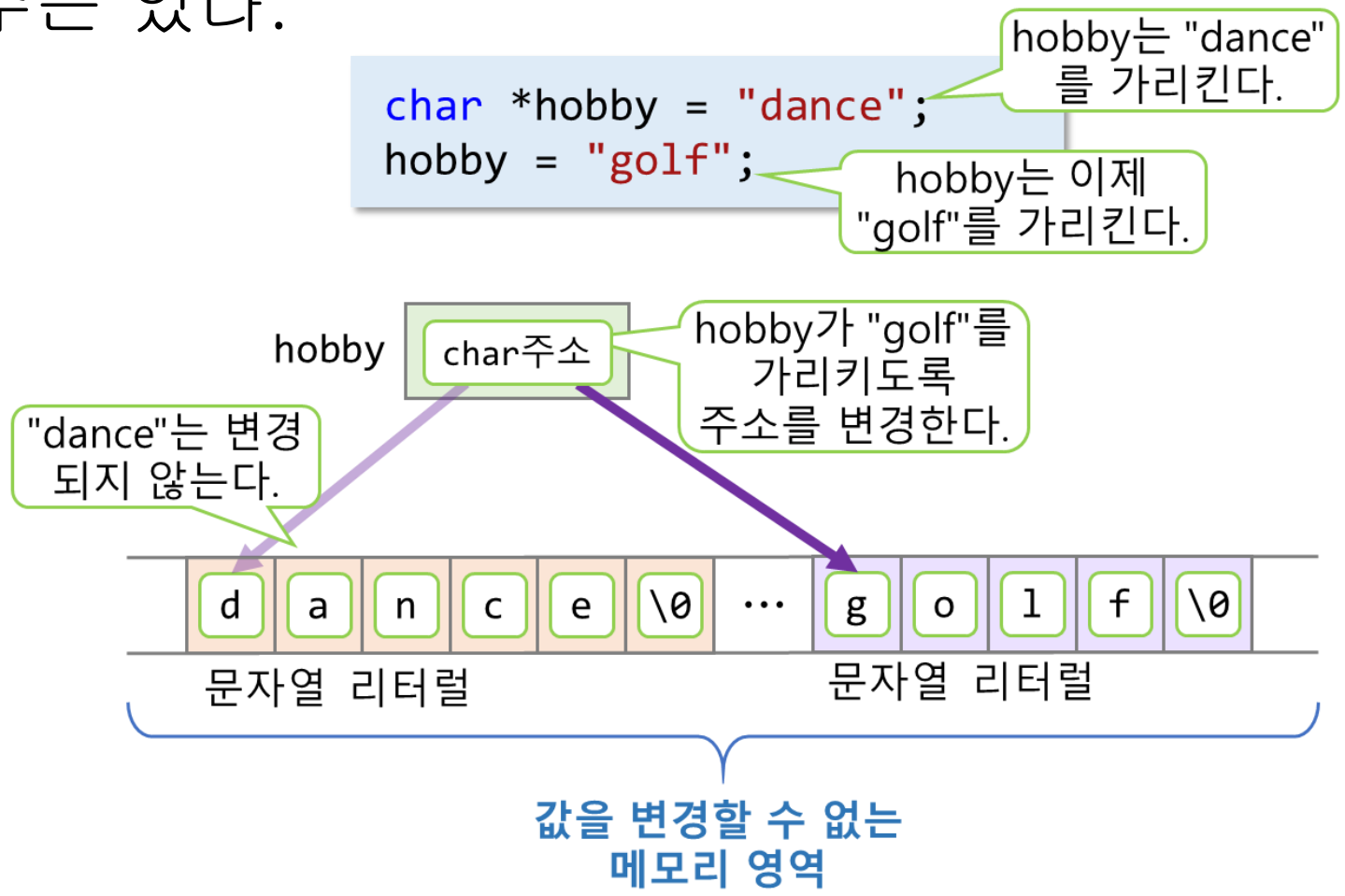
문자열 리터럴 (2/3)

- 문자열 리터럴은 문자 배열처럼 메모리에 저장되지만 값을 변경할 수가 없다.



문자열 리터럴 (3/3)

- 문자열 포인터가 다른 문자열 리터럴을 가리킬 수는 있다.



문자열 포인터

- char*형의 포인터는 문자 배열을 가리킬 수 있다.

```
char str[64] = "";
char *p = str;      // p는 str 배열을 가리킨다.
```

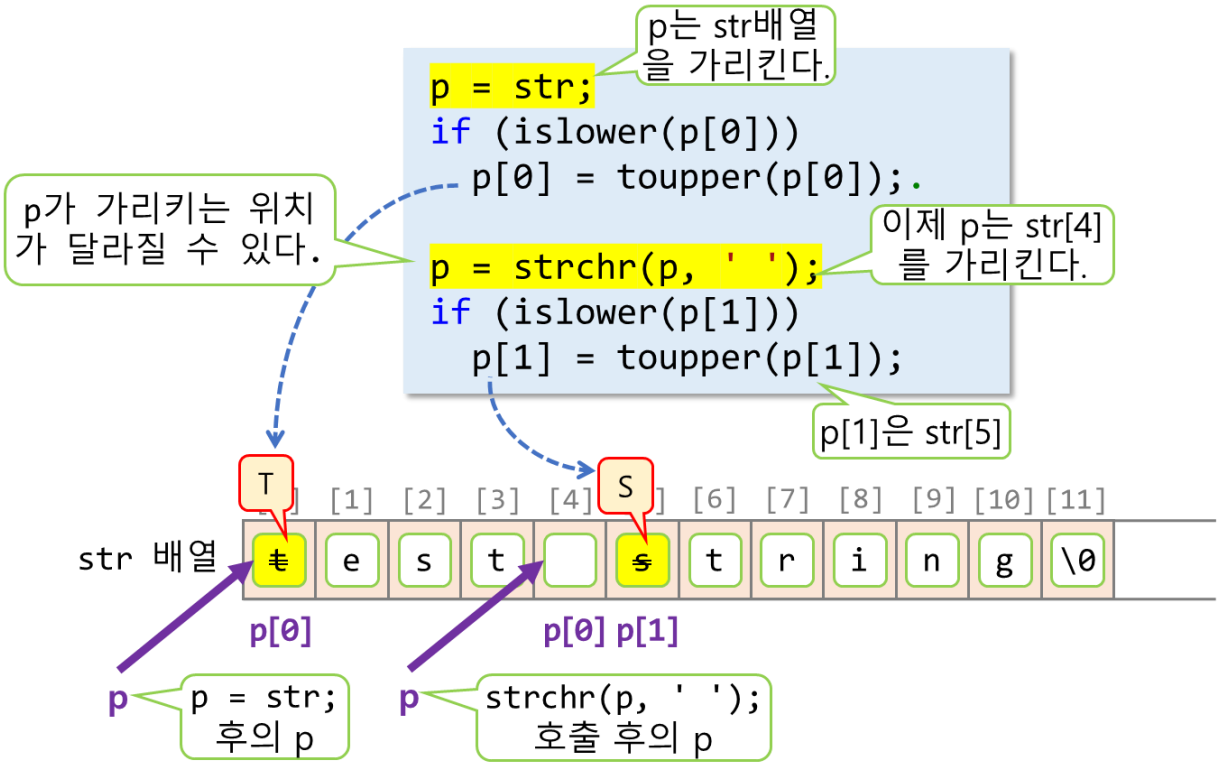
- 문자 배열을 가리키는 포인터를 이용해서 문자열을 변경할 수 있다.

```
p[0] = 'H';          // p가 str을 가리키므로 str[0]을 변경한다.
```

```
strcpy(p, "test string");      // p가 가리키는 str을 변경한다.
```

문자 배열을 가리키는 문자열 포인터의 용도

- 문자 배열을 직접 사용하지 않고 포인터를 사용하는 이유
  - 문자 배열의 특정 위치를 가리키도록 문자열 포인터를 변경해 가면서 문자열에 대한 처리를 할 수 있다.



예제 9-9 : 문자열 포인터가 문자 배열을 가리키는 경우 (1/2)

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
#include <ctype.h> // 문자 처리 라이브러리

int main(void)
{
    char str[64] = "";
    char *p = str; // p는 str 배열을 가리킨다.

    strcpy(p, "test string"); // p가 가리키는 문자 배열을 변경한다.

    if (islower(p[0]))          // p[0]이 소문자인지 검사한다.
        p[0] = toupper(p[0]); // p가 가리키는 str[0]을  대문자로 변경한다.
```

예제 9-9 : 문자열 포인터가 문자 배열을 가리키는 경우 (2/2)

```
16      p = strchr(p, ' ');         // str중 ' ' 문자의 주소를 포인터 p에 저장한다.
17      // ' ' 다음 문자를 대문자로 바꾼다.
18      if (islower(p[1]))
19          p[1] = toupper(p[1]);
20      puts(str);
21      return 0;
22  }
```

실행결과

```
Test String
```

const char*형의 문자열 포인터 (1/2)

- 읽기 전용의 문자열 포인터
  - 문자열의 내용을 읽어볼 수만 있고 변경할 수는 없다.

문자열 리터럴을 가리키는 경우

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

p[0] = 'A';          // 컴파일 에러
strcpy(p, "xyz");    // 컴파일 경고
```

문자 배열을 가리키는 경우

```
char str[64] = "";
const char *p = str;
p[0] = 'A';          // 컴파일 에러
strcpy(p, "xyz");    // 컴파일 경고
```

문자 배열을 읽기 전용으로 접근한다.

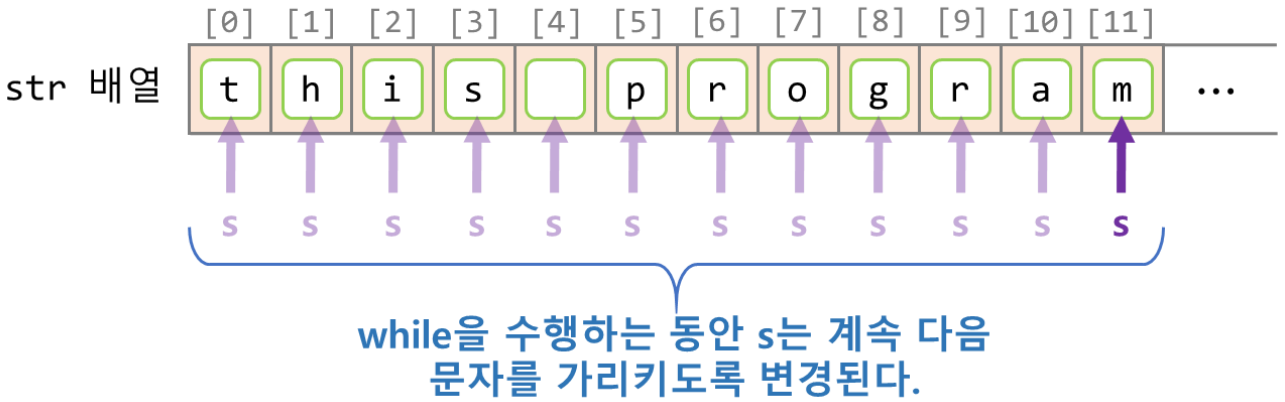
const char*형의 문자열 포인터 (2/2)

- 문자열을 입력 매개변수로 지정할 때 const char*형을 사용한다.

```
int count_space(const char* s)
{
  int count = 0;
  while (s[0] != '\0') {
    if (isspace(s[0]))
      count++;
    s++;
  }
  return count;
}
```

str 배열의 주소가 s로 전달된다.

s가 가리키는 위치가 달라질 수 있다.



예제 9-10 : count_space 함수의 정의 (1/2)

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
#include <ctype.h>  // 문자 처리 라이브러리
int count_space(const char* s);

int main(void)
{
    char str[64] = "this program\ttests const pointer to string\n";

    puts(str);
    printf("공백 문자의 개수: %d\n", count_space(str));
    return 0;
}

```

예제 9-10 : count_space 함수의 정의 (2/2)

```
16  int count_space(const char* s)  // s는 입력 매개변수
17  {
18      int count = 0;
19      while (s[0] != '\0') {   // while (*s != '\0') 과 같은 의미
20          if (isspace(s[0]))   // *s가 공백 문자인지 검사한다.
21              count++;
22          s++;                 // s는 다음 문자를 가리킨다.
23      }
24
25      //s[0] = 'A';       // s가 가리키는 문자열을 변경할 수 없으므로 컴파일 에러
26      //strcpy(s, "xyz"); // strcpy의 매개변수와 데이터형이 다르므로 컴파일 경고
27      return count;
28  }
```

실행결과

```
this program    tests const pointer to string

공백 문자의 개수: 7
```

문자열 사용을 위한 가이드라인 (1/5)

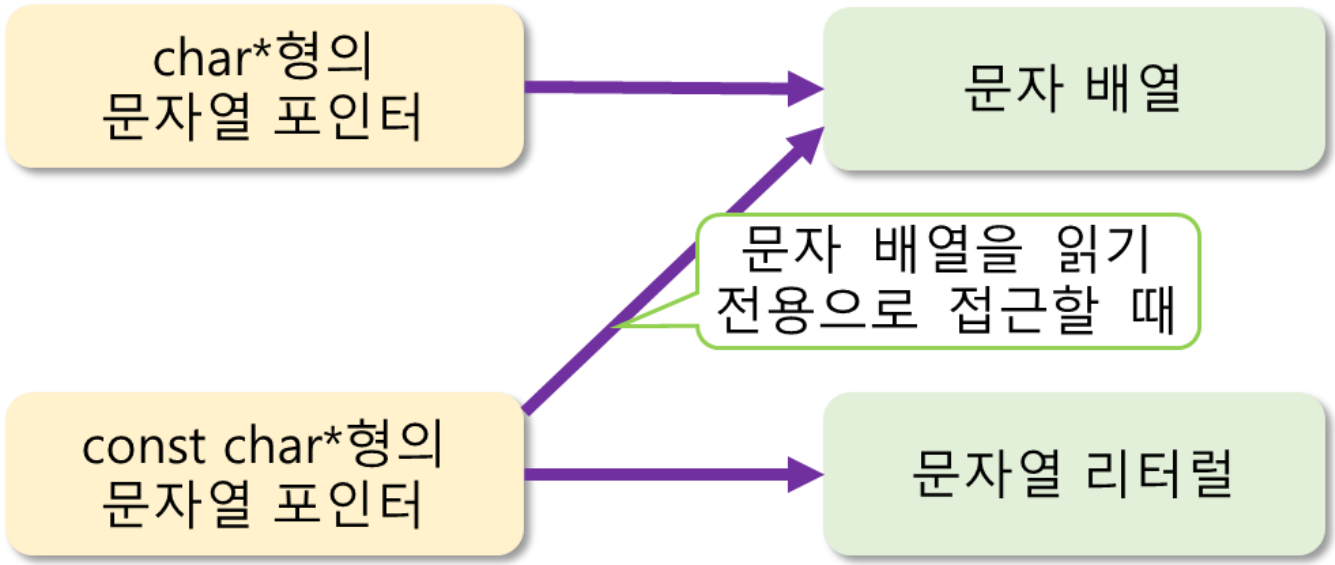
- 문자열의 데이터형을 선택하는 기준
  - ① 사용자로부터 입력받거나 변경할 수 있는 문자열, 즉 문자열 변수는 문자 배열에 저장한다.
  - ② 프로그램 실행 중에 변경되지 않는 문자열, 즉 문자열 상수는 문자열 리터럴로 나타낸다.

문자열 사용을 위한 가이드라인 (2/5)

- 문자열 포인터의 데이터형을 선택하는 기준
  ① char*형의 포인터는 문자 배열, 즉 변경할 수 있는 문자열을 가리킬 때만 사용한다.
  ② const char*형의 포인터는 변경할 수 없는 문자열을 가리킬 때 사용한다.
  ③ 문자열 리터럴을 가리킬 때는 const char*형의 포인터를 사용한다.
  ④ 문자 배열을 읽기 전용으로 접근할 때는 const char*형의 포인터를 사용한다.

문자열 사용을 위한 가이드라인 (3/5)

- 문자열 포인터의 데이터형을 선택하는 기준



문자열 사용을 위한 가이드라인 (4/5)

- 문자열을 매개변수로 전달하는 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.

문자열 사용을 위한 가이드라인 (5/5)

- 문자열을 매개변수로 전달하는 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.

swap_string 함수의 정의

입출력 매개변수

입출력 매개변수

배열의 크기

```
int swap_string(char* lhs, char* rhs, int size)
{
    int lhs_len = strlen(lhs);
    int rhs_len = strlen(rhs);
    char temp[SIZE] = "";

    if (lhs_len + 1 > size || rhs_len + 1 > size)
        return 0;   // swap_string 실패

    strcpy(temp, lhs);
    strcpy(lhs, rhs);
    strcpy(rhs, temp);
    return 1;       // swap_string 성공
}
```

lhs와 rhs는 함수 안에서 변경된다.

예제 9-11 : swap_string 함수의 정의 및 호출 (1/2)

```
04  #define SIZE 128
05  int swap_string(char* lhs, char* rhs, int size);
06
07  int main(void)
08  {
09      char str1[SIZE] = "";
10      char str2[SIZE] = "";
11
12      printf("문자열 2개? ");
13      scanf("%s %s", str1, str2);
14
15      printf("str1=%s, str2=%s\n", str1, str2);
16      swap_string(str1, str2, SIZE);
17      printf("str1=%s, str2=%s\n", str1, str2);
18      return 0;
19  }
```

예제 9-11 : swap_string 함수의 정의 및 호출 (2/2)

```
21  int swap_string(char* lhs, char* rhs, int size)
22  {
23      int lhs_len = strlen(lhs);
24      int rhs_len = strlen(rhs);
25      char temp[SIZE] = "";
26
27      if (lhs_len + 1 > size || rhs_len + 1 > size)
28          return 0;   // swap_string 실패
29
30      strcpy(temp, lhs);
31      strcpy(lhs, rhs);
32      strcpy(rhs, temp);
33      return 1;   // swap_string 성공
34  }
```

실행결과

```
문자열 2개? ski golf
str1=ski, str2=golf
str1=golf, str2=ski
```

문자열의 배열

- 변경할 수 있는 문자열을 여러 개 저장하려면 2차원 문자 배열을 사용한다.
  - 문자 배열이 여러 개 필요
- 변경되지 않는 문자열을 여러 개 저장하려면 문자열 포인터 배열을 사용한다.
  - 문자열 리터럴을 주소만 저장

2차원 문자 배열의 선언 및 초기화

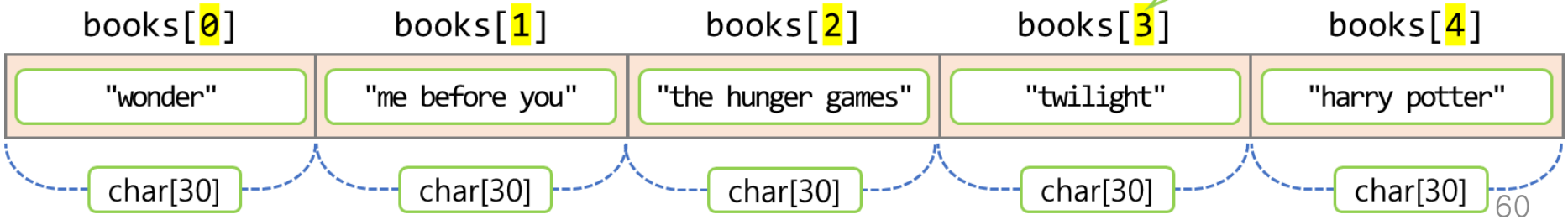
- 열 크기 : 널 문자를 포함한 문자열의 길이
- 행 크기 : 문자열의 개수

문자열의 개수

문자열의 길이

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

행 인덱스만 사용하면 문자열 하나에 접근한다.



2차원 문자 배열의 사용

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

```
for (i = 0; i < 5; i++)
    printf("책 제목: %s\n", books[i]);  // i번째 문자열에 접근한다.
```

- 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])) // 각 문자열의 0번 문자를 대문자로 만든다.
        books[i][0] = toupper(books[i][0]);
}
```

예제 9-12 : 2차원 문자 배열의 선언 및 초기화 (1/2)

```
02  #include <stdio.h>
03  #include <string.h>
04  #include <ctype.h>
05
06  int main(void)
07  {
08      char books[5][30] = {
09          "wonder",              // books[0]의 초기값
10          "me before you",       // books[1]의 초기값
11          "the hunger games",    // books[2]의 초기값
12          "twilight",            // books[3]의 초기값
13          "harry potter",        // books[4]의 초기값
14      };
15      int i = 0;
16
```

예제 9-12 : 2차원 문자 배열의 선언 및 초기화 (2/2)

```
17      for (i = 0; i < 5; i++)
18          printf("책 제목: %s\n", books[i]);        // i번째 문자열 사용
19
20      for (i = 0; i < 5; i++)
21      {
22          if (islower(books[i][0]))          // i번째 문자열의 0번째 문자 사용
23              books[i][0] = toupper(books[i][0]);
24      }
25
26      puts("<< 변경 후 >>");
27      for (i = 0; i < 5; i++)
28          printf("책 제목: %s\n", books[i]);
29
30      return 0;
31  }
```

실행결과

```
책 제목: wonder
책 제목: me before you
책 제목: the hunger games
책 제목: twilight
책 제목: harry potter
<< 변경 후 >>
책 제목: Wonder
책 제목: Me before you
책 제목: The hunger games
책 제목: Twilight
책 제목: Harry potter
```

예제 9-13 : 2차원 문자 배열의 정렬 (1/4)

```
05  int swap_string(char* lhs, char* rhs, int size);
06
07  #define MAX  5      // 2차원 배열의 행 크기
08  #define BUF_SZ 30   // 2차원 배열의 열 크기
09
10  int main(void)
11  {
12      char books[MAX][BUF_SZ] = {
13          "Wonder",
14          "Me before you",
15          "The hunger games",
16          "Twilight",
17          "Harry potter",
18      };
19      int i, j;
20      int index;
```

예제 9-13 : 2차원 문자 배열의 정렬 (2/4)

```
22      puts("<< 정렬 전 >>");
23      for (i = 0; i < MAX; i++)
24          puts(books[i]);
25
26      for (i = 0; i < MAX - 1; i++)
27      {
28          index = i;
29          for (j = i + 1; j < MAX; j++)
30          {
31              if (strcmp(books[index], books[j]) > 0)
32                  index = j;
33          }
34          if (i != index)
35          {
36              swap_string(books[index], books[i], BUF_SZ);
37          }
38      }
```

예제 9-13 : 2차원 문자 배열의 정렬 (3/4)

```
40      puts("<< 정렬 후 >>");
41      for (i = 0; i < MAX; i++)
42          puts(books[i]);
43
44      return 0;
45  }
46
47  int swap_string(char* lhs, char* rhs, int size)
48  {
49      int lhs_len = strlen(lhs);
50      int rhs_len = strlen(rhs);
51      char temp[BUF_SZ] = "";
52
53      if (lhs_len + 1 > size || rhs_len + 1 > size)
54          return 0;   // swap_string 실패
```

예제 9-13 : 2차원 문자 배열의 정렬 (4/4)

```
56          strcpy(temp, lhs);
57          strcpy(lhs, rhs);
58          strcpy(rhs, temp);
59          return 1;       // swap_string 성공
60      }
```

실행결과

```
<< 정렬 전 >>
Wonder
Me before you
The hunger games
Twilight
Harry potter
<< 정렬 후 >>
Harry potter
Me before you
The hunger games
Twilight
Wonder
```

문자열 포인터 배열 (1/2)

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

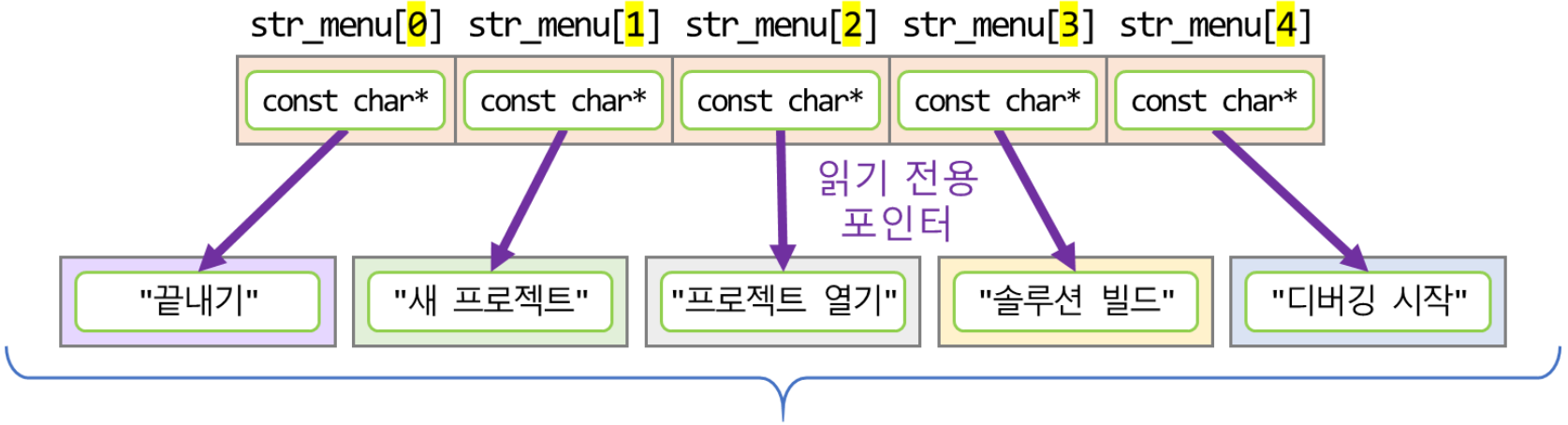
문자열 포인터 배열 (2/2)

문자열의 개수

5

```
const char *str_menu[] = {
    "끝내기",
    "새 프로젝트",
    "프로젝트 열기",
    "솔루션 빌드",
    "디버깅 시작"
};
```

문자열의 주소를 저장하는 배열



예제 9-14 : 문자열 포인터 배열의 사용 예 (1/2)

```
05  int main(void)
06  {
07      const char *str_menu[] = {          // str_menu는 원소가 5개인 포인터 배열
08          "끝내기",
09          "새 프로젝트",
10          "프로젝트 열기",
11          "솔루션 빌드",
12         "디버깅 시작"
13      };
14      int sz_menu = sizeof(str_menu) / sizeof(str_menu[0]);
15      int menu;
16
17      while (1)
18      {
19          int i;
20          for (i = 0; i < sz_menu; i++)
21              printf("%d.%s\n", i, str_menu[i]);
```

예제 9-14 : 문자열 포인터 배열의 사용 예 (2/2)

```
23          printf("메뉴 선택? ");
24          scanf("%d", &menu);
25          if (menu == 0)        // menu를 입력받은 다음 루프 탈출 조건을 검사한다.
26              break;
27          else if (menu > 0 && menu < sz_menu)
28              printf("%s 메뉴를 선택했습니다.\n\n", str_menu[menu]);
29          else
30              printf("잘못 선택했습니다.\n\n");
31      }
32
33      return 0;
34  }
```

실행결과

```
0.끝내기
1.새 프로젝트
2.프로젝트 열기
3.솔루션 빌드
4.디버깅 시작
메뉴 선택? 1
새 프로젝트 메뉴를 선택했습니다.

0.끝내기
1.새 프로젝트
2.프로젝트 열기
3.솔루션 빌드
4.디버깅 시작
메뉴 선택? 0
```