7차시 - 문자열(2)

문자열 처리 함수

함수	설명
strlen	문자열의 길이를 반환한다.
strcpy	문자열의 원본과 동일한 복사본을 만든다. (a = b 와 같은 기능)
strcat	두 개의 문자열을 나란히 붙인다. (a = b + c 와 같은 기능)
strcmp	두 개의 문자열을 비교한다.(알파벳순) (a == b 와 같은 기능)

- #include <string.h> 를 추가해야 한다.

strlen

- int strlen(const char *str) : **문자열의 길이를 반환**
  - 입력 : 길이를 확인하고자 하는 문자열의 포인터
  - 반환값 : 문자열의 길이

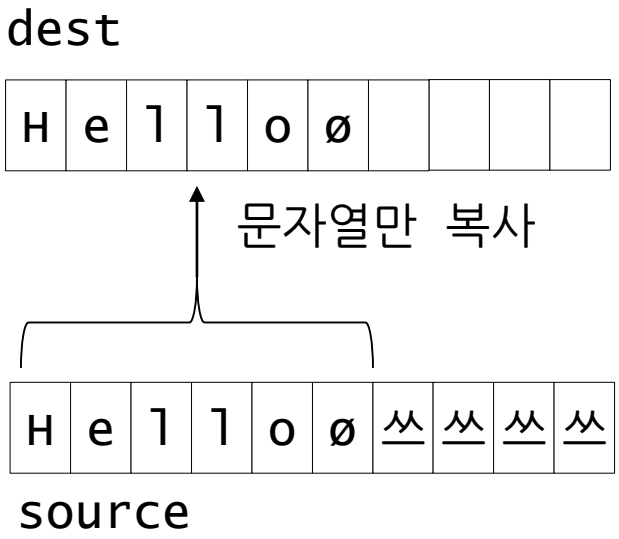
```
char str[20];

gets(str);
printf("입력한 문자열의 길이는 %d 입니다.\n", strlen(str) );
```

- **문자열의 길이란**
  - 앞에서부터 차례로 숫자를 세면서 ₩0이 나타나는 곳의 번호를 반환

strcpy, strncpy

- strcpy : **문자열 전체 복사**, strncpy : **문자열 n 바이트 복사**
- char *strcpy(char *dest, const char *src);
  - 입력 : src 원본 문자열 포인터, dest 목적 문자열 포인터
  - 반환값 : dest 목적 문자열 포인터 (잘 사용하지 않음)

```
char   source[10], dest[10];

// 초기화 방법
strcpy(source, “Hello”);
strcpy(dest, source);
```



strcpy가 필요한 이유

- 다음 중 정상적인 문장은?

(1)
```
char str1[10], str2[10]= “Hello”;
strcpy(str1, str2);
```

(2)
```
char str1[10], str2[10] = “Hello”;
str1 = str2;
```

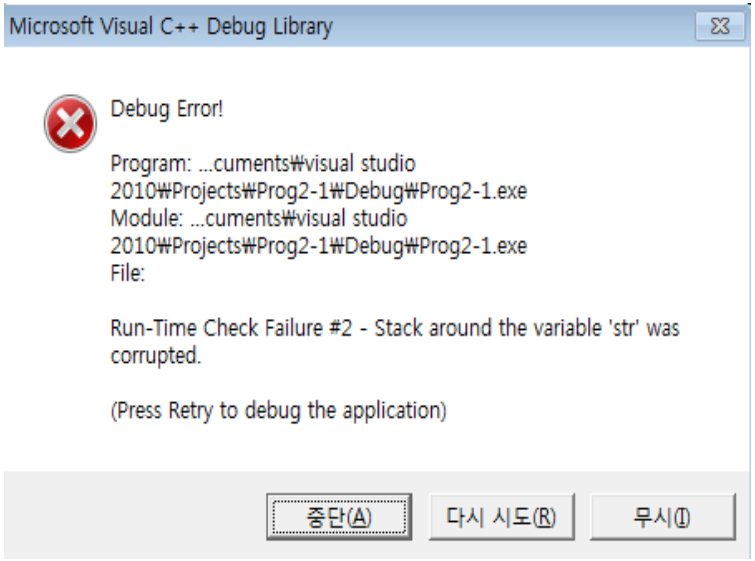
(3)
```
char *str1, *str2 = “Hello”;
str1 = str2;
```

(4)
```
char str1[5], str2[] = “Hello”;
strcpy(str1, str2);
```

(5)
```
char  *str1, str2[] = “Hello”;
str1 = str2;
```

strcpy 주의점

```
char str[5], str2[] = “Hello”;
strcpy(str, str2);
```



- “Run-Time Check Failure #2 - Stack around the variable ‘str’ was corrupted.”의 의미는?

문자열에서는 메모리 사용에 주의해야 한다.

strncpy

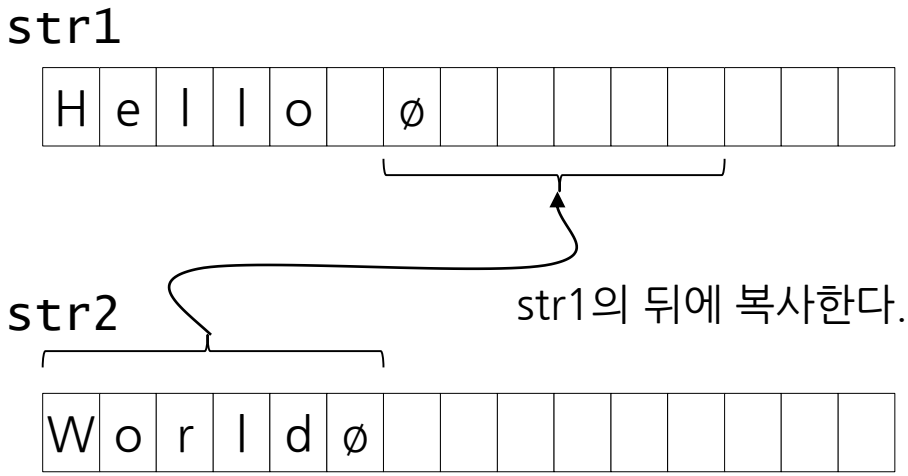
```
char  str1[] = “hello world”;
char  str2[20], str3[10];

strncpy(str2, str1, sizeof(str2));
strncpy(str3, str1, 5); str3[5] = 0;
```

- n 바이트 만큼만 복사
- 일부분만 복사하면 ₩0이 복사 안될 수도 있다.
- 메모리 문제를 피할 수 있는 방법이다. (strcpy_s도 있다)

strcat, strncat

- char *strcat(char *str1, const char *str2) **문자열 접합**
  - str1, str2 : 접합할 문자열.
  - str1 = str1 + str2;
  - 반환값 : 접합된 문자열이 포인터. str1과 같다.

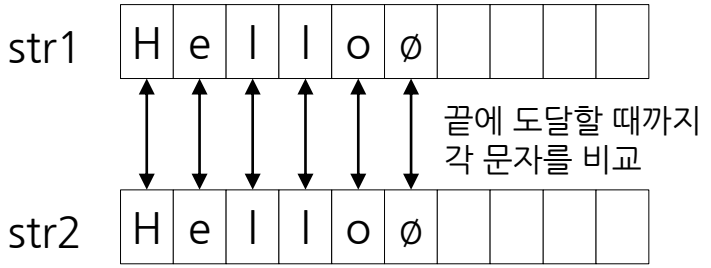
```
char  str1[20] = "Hello ";
char str2[10] = "world";

strcat(str1, str2);
```



메모리 초과 주의

strcmp, strncmp

- `int strcmp(const char *str1, const char *str2)` **비교**
  - str1, str2 : 비교할 문자열
  - 반환값 : 비교 결과 > 0 : str1 이 크다.
    == 0 : 같다.
    < 0 : str2 이 크다.



- **if문을 쓸 때 주의해야 한다.**

```
char   str1[ ] = "Hello";
char   str2[ ] = "World";

if ( ! strcmp(str1, str2) ) printf("equal");
```

왜 !를 붙였을까?

strcmp의 사용

(1)

```
char  str1[10] = “Hello”; // 문자열 부분만 비교하며
char  str2[20] = “Hello”; // 자료형의 크기는 무관하다.
if (! strcmp(str1, str2)) printf(“identical\n”);
```

(2)

```
char  str1[10] = “Hello”;
char  str2[20] = “Hello  ”; // 공백도 문자다(ASCII  32)
char  str3[30] = “hello”;   // 대소문자는 다르다( A;65, a:97)
if (! strcmp(str1, str2)) printf(“identical\n”);
if (! strcmp(str1, str3)) printf(“identical\n”);
```

strcmp의 사용

(3) 결과는?

```
char   str1[20] = “Hello”;
char   str2[20] = “Hello World”;


str2[5] = 0;
if (!strcmp(str1, str2) printf(“identical”);
```

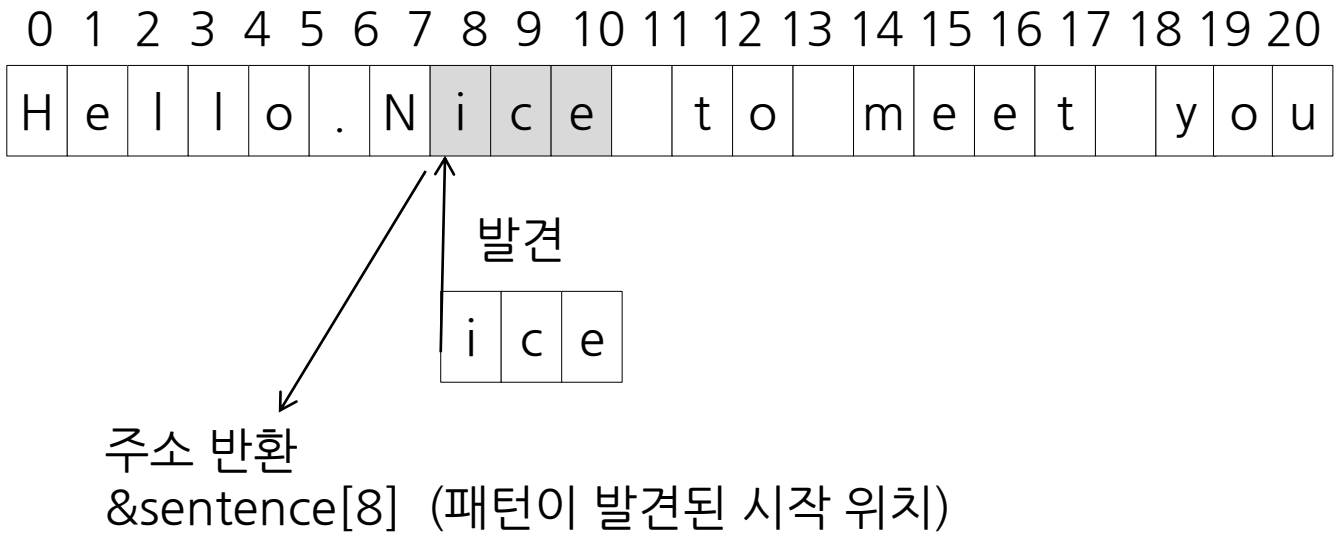
strchr, strstr

- `char *strchr(char *str, int cha)`
  - 문자열 str에서 하나의 문자 cha 가 있는 위치를 반환 (문자 검색)
  - 없으면 NULL, 여러 개면 직접 다음 위치부터 계속 찾아야 한다.

- `char *strstr(char *str1, char *str2)` (*)
  - 문자열 str1에서 문자열 str2가 있는 위치를 반환(문자열 검색)
  - 없으면 NULL, 여러 개면 직접 다음 위치부터 계속 찾아야 한다.

strstr

```
char sentence[ ] = “Hello.Nice to meet you”;
char key[ ] = “ice”;
char *ptr = sentence;

ptr = strstr(ptr, key);          // 첫 번째 검색
ptr = strstr(ptr+1, key);    // 다음 검색
```



strtok (*)중요

- 문자열에서 토큰 단위로 자를 때 매우 유용하다
- char *strtok(char *str, const char *delimeter)
  - str : 문자열
  - delimeter : 토큰을 구분하는 문자들의 집합이다.
  - 두 번째 호출부터는 str에 NULL을 준다.

```
tok = strtok (str, " ,.-?\'\""");   //한 단어씩 가져온다.
tok = strtok (NULL , " ,.-?\'\""); // 두 번째 이후 호출 방식
```

strtok의 사용

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

int main()
{
    char str[100];
    char *token;

    printf("Input a sentence :");
    gets(str);

    token = strtok(str, " ,.-");       // 첫 번째 토큰을 얻음
    while (token != NULL) {            // 더 이상 토큰이 없을 때까지 반복
       printf("%s\n", token);
       token = strtok(NULL, " ,.-");  // 두 번째 이후의 토큰을 얻음
    }
    return 0;
}
```

문자 검사 함수

함수 이름	의미
isalnum	문자가 알파벳 또는 숫자인지 판단
isalpha	문자가 알파벳(a-z, A-Z)인지 판단
isdigit	문자가 숫자(0-9)인지 판단
islower	문자가 소문자(a-z)인지 판단
isspace	문자가 공백 문자(탭 문자, 줄 바꿈 문자 포함)인지 판단
isupper	문자가 대문자(A-Z)인지 판단

- #include <ctype.h>

```
while (str[i]) {
        if ( isalpha(str[i]) )
                printf("%c is alphabet₩n", str[i]);
        else if ( isdigit(str[i]) )
                printf("%c is digit₩n", str[i]);
        i++;
}
```

대소문자 변환

- int toupper (int c);
- int tolower (int c);

```
while ( ( ch = getchar( ) ) != '₩n')
        putchar( toupper(ch) );
```

- toupper, tolower 함수의 정체

```
#define tolower(c)    ( (((c) >= 'A') && ((c) <= 'Z')) ? ((c) - 'A' + 'a') : (c) )
#define toupper(c)    ( (((c) >= 'a') && ((c) <= 'z')) ? ((c) - 'a' + 'A') : (c) )
```

문자열 변환 함수

- 다음의 두 값은 다르다
  - `int  i = 1234;`
  - `char s[ ] = “1234”;`
- 그러나 프로그램에서는 서로 바꾸어 쓸 때가 많다.
- 문자열 변환 함수는 숫자 1234와 문자열 “1234”를 바꿀 때 사용한다.
  - 문자열 → 숫자 : `atoi, atof, atol`
  - 숫자 → 문자열 : `itoa, ltoa(표준이 아니다), fcvt, sprintf`

atoi, atol, atof

```
int  i;
char s[ ] = “1234”;

i = atoi(s);
printf(“%d\n”, i);
```

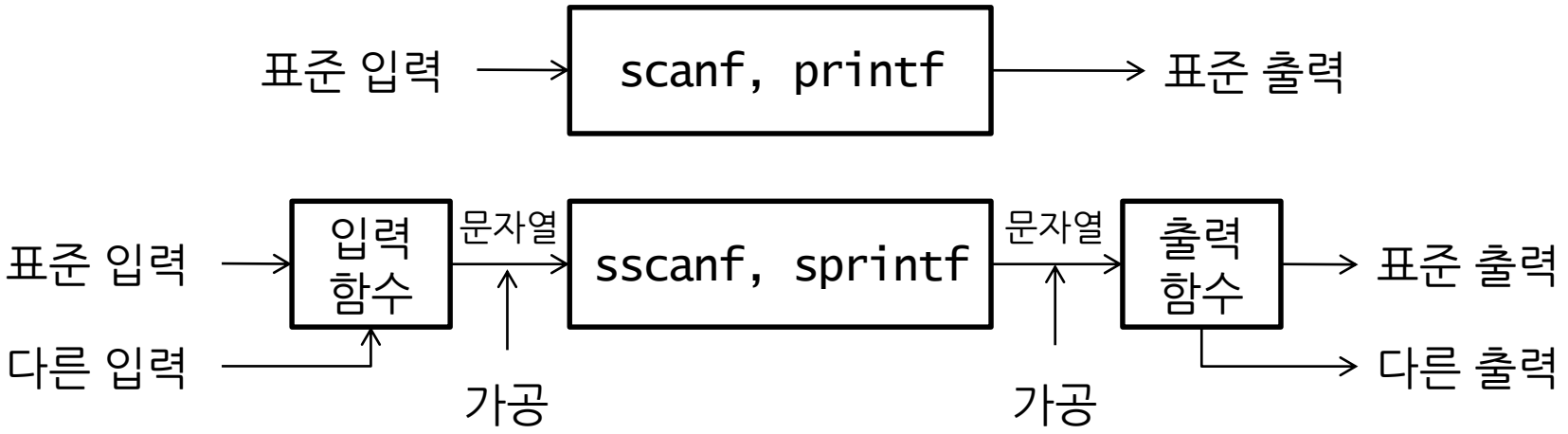
- #include <stdlib.h>
- **확장된 기능의 strtol, strtod, strtoul 도 있다.**
  - 진수 변환이 포함되어 있다. (2진수도 가능)

숫자를 문자열로

- **itoa, ltoa (정수형)**
  - 표준이 아니라서 컴파일러마다 지원 여부가 다르다.
- **fcvt, ecvt (실수형)**
  - 점차 사라지는 추세이다.
- **sprintf 로 대체할 수 있다. (다음 절 참고)**

sprintf, sscanf (*)중요

- **printf, scanf**
  - 장점 : 형식에 맞추기 좋다.
  - 단점 : 양식에 맞지 않은 입력이 있으면 동작하지 않는다.
- **sscanf, sprintf : 장점은 취하고 단점은 개선한다.**
  - gets로 일단 입력 받아 잘못된 입력 문제를 해결한다.
  - 형식에 맞추어 변수에 넣거나 변수의 값을 출력한다.



sscanf, sprintf

- `int sscanf (const char *str, const char *format, …)`
- `int sprintf (char *str, const char *format, …)`
  - `printf, scanf`의 기능을 그대로 이용할 수 있다.
  - 입출력에 사용하기 전에 문제가 될만한 것은 미리 개발자가 검사한다.

sprintf의 활용

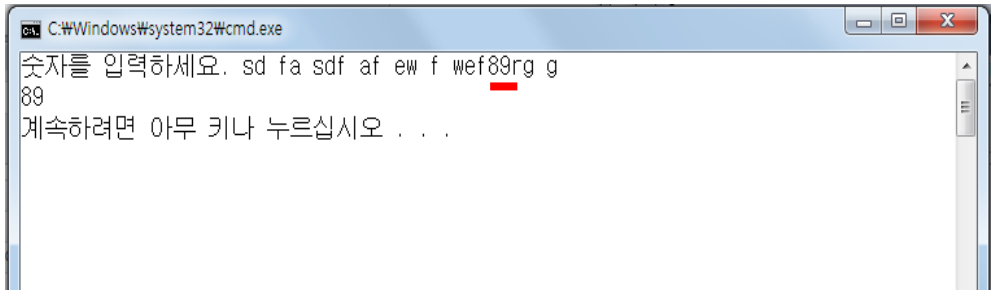
- **itoa, ftoa의 대체**
  - sprintf(numstr, “%d”, value);

- **출력 양식을 직접 만드는 방법**

```
printf("실수와 소수점 자릿수를 입력하세요 : ");
scanf("%lf %d", &d, &i);                    // 실수와 정수를 하나씩 입력 받는다.
sprintf(sformat, "formatted value is %%.%dlf₩n", i);   // 출력 양식을 직접 만든다.
                                  // “formatted value is %.4lf₩n” 을 만든다 ( i가 4일 때 ).
printf(sformat, d);                         // 만들어진 출력 양식으로 d 값을 출력한다.
```

사용자 입력의 예외 처리

```
char input[50], number[10];
char *ptr;
int  length = 0, isstart = 0;

printf("숫자를 입력하세요.");
gets(input);
ptr = input;

while (*ptr) {                  // 문장의 끝까지 검토
    if (isdigit(*ptr)) {        // 숫자라면
        isstart = 1;
        length++;               // 길이를 측정
    }
    else if (isstart)           // 숫자가 아니면서, 숫자 부분이 지나갔다면
        break;                  // 검사 중지
    ptr++;
}

strncpy(number, ptr - length, length);     // 숫자 부분만 추출
number[length] = 0;

printf("%s\n", number);                    // 화면에 출력
```



직접 만들어 보는 문자열 처리 함수

- strcpy

```
char strcpy(char *dest, const  char *source)
{
    char *dptr = dest;          // 작업용 포인터 선언

    do {
        *dptr++ = *source++;    // 한 문자씩 복사
    } while( *source);          // source에 널(NULL)문자가
                                // 올 때까지 반복
      return dest;
}
```

널(NULL) 문자에 신경 써야 한다.

직접 만들어 보는 문자열 함수

- strcat, strcmp, strtok
- trim : 앞 뒤의 공백문자 제거
  - ltrim
  - rtrim
- mid : 문자열의 가운데 부분 가져오기
  - 어떤 함수를 이용하면 될까?
- replace : 문자열 바꾸기
  - str 문자열에서 key 문자열을 찾아 replace 문자열로 바꾸기

문자열의 이해

```
int main( void )
{
      char  str1[] = “abcd”;

      str1[4] = ‘a’;

      printf(“%s\n”, str);
}
```

```
int main( void )
{
     char  str2[] = “Hello World”;

      str2[5] = 0;

      printf(“%s\n”, str);
}
```

- 널 문자의 관리

문자열의 이해

- `char str[10] = “Hello”;`
- **문자열 변수 str의 내용 가장 끝에 .을 추가하라. “Hello.”**

```
str [strlen(str)] = ‘.’ ;     // 가장 끝에 마침표를 추가하였다.
str [strlen(str)+1] = 0;    // 빠뜨리지 않고 널 문자를 추가하였다.
```

- **널 문자를 고려한 과정을 생각하라.**
  - 널 문자 부터 넣고,
  - .을 입력하라.
  - 문자열은 계속 바뀌고 있으므로, `strlen`도 계속 변화한다.