작성자 : 김기희 (cupid072@naver.com)

# 문자열 처리 함수

- 학습목표
  - 문자열의 길이를 구할 수 있다.
  - 문자열을 연결할 수 있다.
  - 문자열을 복사할 수 있다.
  - 문자열을 비교할 수 있다.
  - 문자열을 숫자로 변환할 수 있다.
  - 문자열 관련 함수를 사용하여 요구 사항에 맞는 프로그램을 작성할 수 있다.

# 문자열 관련 주의사항 #1

- 문자열은 다음과 같은 두 가지 방식의 문자열이 있다.

- 배열 기반의 문자열

  배열 기반의 문자열은 문자열이 STACK 세그먼트 영역에 저장되어 있으므로 문자열 내의 부분적인 편집이 가능하다.

  char name[] = “KIHEE”;

- 포인터 기반의 문자열

  포인터 기반의 문자열은 문자열이 DATA(CONST)영역에 저장되어 있으며 포인터를 통해 문자열을 참조하게 된다. 그러므로 문자열의 부분 편집이 되어서는 안 된다.

  char* name = “KIHEE”;

- 포인터 기반 문자열의 부분 편집을 방지하기 위해 포인터 기반 문자열은 다음과 같이 const 지정자를 사용하도록 한다.

  const char* name = “KIHEE”;

# 문자열 관련 주의사항 #2

- 포인터 기반 문자열은 부분적으로 변경하여서는 안 된다.
- 부분적이라는 의미는 문자열이 저장된 메모리 공간의 값을 변경하여서는 안된다는 것을 의미한다.

- 배열 기반 문자열에 대한 작업

```
char name[] = “KOREA”;
name[0] = ‘C’;                  // OK
strcpy(name, “HELLO”);          // OK
name = “HELLO”;                 // ERROR : name 은 (주소)상수이다.
```

- 포인터 기반 문자열에 대한 작업

```
char* name = “KOREA”;
name[0] = ‘C’;                  // NG : 포인터 기반 문자열의 부분적인 변경
strcpy(name, “HELLO”);          // NG : 포인터 기반 문자열의 부분적인 변경
name = “HELLO“;                 // OK
```

# strlen()

- 문자열의 길이를 반환한다.

```
#include <string.h>
size_t strlen (const char *s)

char* name = "KIHEE KIM";
size_t len = strlen(name);
printf("문자열의 길이 : %I64u\n", len);
```

- UTF-8 인코딩 문자열은 하나의 하나의 문자가 3Byte이다.
- 그러므로 한글 한 문자를 저장하기 위한 바이트 수는 3Byte이다.
- 반면 문자열 상수의 한글 하나의 크기는 2Byte이다.
- Windows 시스템에서의 한글 문자 하나의 크기는 2Byte이다.

# strcat()

- to 배열에 from 문자열을 연결한다.

  #include <string.h>

  char * strcat (char *restrict to, const char *restrict from)

  ```
  char first[100] = "KIHEE";
  char last[] = "KIM";
  char* name = strcat(first, last);
  printf("%s\n", name);                    // KIHEEKIM
  ```

- to의 여분의 공간이 from 문자열을 추가하기 부족한 경우 배열의 경계범위를 넘어서는 overflow가 발생하므로 주의해야 한다.

- to의 여분의 공간에 from 문자열을 추가하는 함수이므로 to 전달인자는 char형 배열의 시작주소 이어야 한다.

# strncat()

- to 배열에 from 문자열을 연결한다. 이때 from 문자열의 길이가 size 보다 크면 size 만큼 연결한다.

```
#include <string.h>
char * strncat (char *restrict to, const char *restrict from, size_t size)

char* first = "Hello ";
char* last = "KIHEE";
char name[100] = "";
strncat(name, first, sizeof(name) - 1);                    // 최대 name 배열의 여분 만큼 만 연결
strncat(name, last, sizeof(name) - strlen(name) - 1); // 최대 name 배열의 여분 만큼 만 연결
printf("%s\n", name);
```

# restrict 키워드

- C99에서 추가된 키워드로써 포인터에 사용된다.
- restrict 키워드가 부여된 포인터는 포인터가 가리키는 메모리 공간을 참조할 수 있는 유일한 포인터임을 나타낸다.
- restrict 키워드는 * 뒤에 기술되어야 한다.

  int* restrict ptr
- 아래 memcpy() 함수의 첫 번째 인수와 두 번째 인수에는 restrict 키워드가 지정 되어있다. 이것은 memcpy() 함수 내에서 to와 from은 to와 from이 가리키는 메모리 주소를 참조할 수 있는 유일한 포인터 임을 의미한다.
- 쉽게 이야기 한다면 to와 from은 동일한 메모리 번지 이거나 to와 from이 참조하는 메모리 범위가 중첩되어서는 안된다는 것을 의미한다.

  void * memcpy (void *restrict to, const void *restrict from, size_t size)
- restrict 키워드는 특정 메모리 영역에 접근할 수 있는 유일의 포인터를 지정함으로써 컴파일러가 더 나은 최적화를 하도록 도와준다.

# strcpy(), strncpy()

- to 배열에 from 문자열을 복사한다.

  #include <string.h>

  char * strcpy (char *restrict to, const char *restrict from)

- 두 번째 파라미터의 문자열이 첫 번째 파라미터의 배열 공간보다 클 경우 배열의 경계범위를 벗어나는 overflow 가 발생한다.

- overflow 방지를 위해 strncpy() 함수를 사용할 것을 권장한다.
- strncpy() 함수는 to 배열에 from 문자열을 복사한다. 이때 from 문자열이 size 보다 크면 size 만큼 만 복사한다.

  #include <string.h>

  char * strncpy (char *restrict to, const char *restrict from, size_t size)

# strcmp()

- 두 문자열 s1과 s2를 비교한다.

  #include <string.h>

  int strcmp (const char *s1, const char *s2)

- 반환값

  양수 또는 1 : s1이 더 큰 경우

  음수 또는 -1 : s1이 더 작은 경우

  0 : s1과 s2과 같은 경우

- strcmp() 함수는 사전순으로 비교한다.
- 두 문자열의 첫 번째 문자끼리 ASCII 코드 값을 비교 한 후 값의 차이에 따라 결과를 반환한다.
- 만약 첫 번째 문자가 동일한 문자라면 두 번째 문자끼리 비교를 한 후 값의 차이에 따라 결과를 반환한다.
- 반환 값은 컴파일러에 따라 양수, 음수, 0 또는 1, -1, 0의 값을 반환한다.

# strncmp()

- 두 문자열 s1과 s2를 size 길이 만큼 비교한다.

  #include <string.h>

  int strcnmp (const char *s1, const char *s2, size_t size)

- 반환값

  양수 또는 1 : s1이 더 큰 경우

  음수 또는 -1 : s1이 더 작은 경우

  0 : s1과 s2과 같은 경우

# atoi(), atol(), atoll(), atof()

- 문자열을 숫자로 변환한다.

```
#include <stdlib.h>
int atoi(const char* string);                 // int 형으로 변환
long int atol(const char* string);            // long int 형으로 변환
long long int atoll(const char* string);      // long long int 형으로 변환
double atof(const char* string);              // double 형으로 변환
```

- 위의 변환 함수는 모든 문자열의 변환 성공 여부를 확인할 수 있는 기능을 제공하지 않는다.

```
char* string = "A123";
int number = atoi(string);
printf("%d\n", number);
```

- 문자열의 모든 문자가 '0' 인가? 첫 번째 문자부터 변환에 실패하였는가? 또는 빈 문자열인가? 알 수 없음

# strtol(), strtoul(), strtoll(), strtoull(), strtof(), strtod()

- 문자열 string을 숫자로 변환한다.

  #include <stdlib.h>

  long int strtol (const char *restrict string, char **restrict tailptr, int base)

  unsigned long int strtoul (const char *retrict string, char **restrict tailptr, int base)

  long long int strtoll (const char *restrict string, char **restrict tailptr, int base)

  unsigned long long int strtoull (const char *restrict string, char **restrict tailptr, int base)

  float strtof (const char *string, char **tailptr)

  double strtod (const char *restrict string, char **restrict tailptr)

- 파라미터

  string : 숫자로 변환할 문자열

  tailptr : 숫자로 변환 성공한 마지막 문자 다음의 주소를 저장하기 위한 포인터

  base : 숫자 변환에 사용할 진법

# strtol() 함수의 사용 예

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

int main() {
	char* string = "FFFF";
	char* tailptr = NULL;                          // 문자 변환 성공 다음 문자의 주소를 저장하기 위한 포인터
	long int number = strtol(string, &tailptr, 16);  // string을 16진수 문자열로 보고 long int 형으로 변환

	if (tailptr == (string + strlen(string))) {    // 마지막 문자까지 변환에 성공하였는가?
		printf("%I32u\n", number);
	}
	else {
		printf("변환에 실패하였습니다.");
	}
	return 0;
}
```

# toupper(), tolower()

- 문자를 대문자 또는 소문자로 변환한다.

  #include <ctype.h>

  int toupper(int c)

  int tolower(int c)

- 다음의 문자열을 모두 대문자로 변환하는 프로그램을 작성해 보세요.

  char name = "Hello C Programming";