
C 프로그래밍 및 실습

10. 문자열

p. 24 예시 추가

세종대학교

목차

- 1) 문자열 개요
- 2) 문자열 저장 및 기본 입출력
- 3) 문자열과 포인터
- 4) 문자열의 배열
- 5) 문자열 및 문자 처리 함수
- 6) 문자열 및 문자 입출력

1) 문자열이란

- 문자열(string): 연속적으로 나열된 문자들의 묶음
 - 문자열은 기본적으로 **문자 배열**을 사용하여 저장
 - ✓ 문자 단위로 초기화 하고 출력하는 코드(이미 학습)

```
char str[8] = {'H','e','l','l','o'}; // 문자로 초기화
int i;

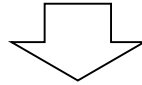
for (i=0 ; i<5 ; i++)
    printf("%c", str[i]); // 문자 출력
```

- 이름이나 주소처럼 하나의 묶음으로 처리되는 정보를 매번 문자 단위로 처리하는 것은 매우 불편
- ➔ C언어에서는 문자들을 **문자열 단위로** 처리할 수 있는 기능 제공

1) 문자열이란

- 맛보기: 앞의 예제 프로그램을 문자열 처리 방식으로 바꾸면?

```
char str[8] = {'H','e','l','l','o'}; // 문자 단위 초기화
int i;
for (i=0 ; i<5 ; i++)
    printf("%c", str[i]); // 문자 단위 출력
```



```
char str[8] = "Hello"; // 문자열로 초기화
printf("%s", str); // 문자열 출력
```

1) 문자열이란

- **문자열 표현**

- 큰 따옴표로 감싸서 나타냄
- 예) "Hello" , "abc" , "123"
- 비교) 문자는 작은 따옴표로 감싸서 표현 'a' , '1'

- **문자열 입출력**

- scanf, printf에서 문자열 단위 입출력 지원
- 문자열 입출력의 위한 서식 문자: %s
- 인자: 보통 문자열을 저장하는 문자 배열의 이름

```
char str[8] = "Hello"; // 문자열로 초기화
printf("%s", str);      // 문자열 출력
```

1) 문자열이란

- **[예제 10.1] 다음 프로그램을 작성하시오.**
 - 크기가 10인 문자 배열 str을 선언과 동시에 문자열 "Hello"로 초기화
 - 문자열 str을 화면에 출력
 - 사용자로부터 문자열 "World"를 입력 받아 str에 저장
 - 문자열 str을 화면에 출력

목차

- 1) 문자열 개요
- 2) **문자열 저장 및 기본 입출력**
- 3) 문자열과 포인터
- 4) 문자열의 배열
- 5) 문자열 및 문자 처리 함수
- 6) 문자열 및 문자 입출력

2) 문자열 저장 및 기본 입출력

- 문자열 표현: 큰 따옴표 사용
 - 문자열 예시: "Hello" , "A" , "123"
 - 공백 하나로 구성된 문자열: " " ⇒ 큰따옴표 사이에 공백
 - 큰따옴표 하나로 구성된 문자열: "₩" ⇒ 큰따옴표 문자 상수 사용 (2.5절 참조)
 - 길이가 0인 문자열: "" ⇒ 큰따옴표 사이에 아무것도 없음
- (비교) "A"와 'A':
 - "A" 는 문자열
 - 'A' 는 문자
 - 자세한 차이점은 잠시 뒤에 설명

2) 문자열 저장 및 기본 입출력

▪ 문자열 저장 및 초기화

- C언어에서는 **문자 배열** 에 문자열 저장
- 문자 배열 선언 및 초기화 예

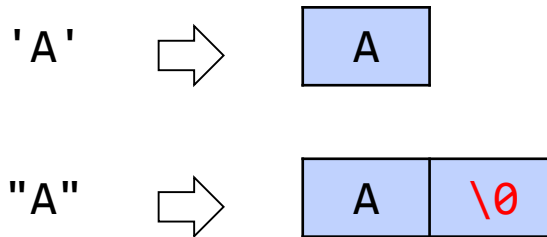
```
char str1[8] = "Hello" ;    ⇒ 배열 크기 지정  
char str2[ ] = "Hello" ;    ⇒ 배열 크기 미 지정  
                             (초기화 값에 의해 크기 결정)
```

- 문자열로 초기화하는 것은 선언 시에만 가능

```
char str[20];  
  
str[0] = 'a'; ⇒ 0번 원소에 문자 'a' 대입 (가능)  
str = "Hello"; ⇒ 컴파일 오류 : 배열 초기화는 선언 시에만 가능  
str = {'H','e','l','l','o' }; ⇒ 컴파일 오류 : 위와 동일
```

2) 문자열 저장 및 기본 입출력

- **널(null) 문자:**
 - 문자열의 끝을 의미하는 특수 문자로, '**\0**'으로 표현
 - 널문자의 아스키 코드 값은 정수 0, 즉 '**\0**' == 0
 - 문자열을 처리하는 기준이 되는 매우 중요한 요소
- **문자열은 항상 맨 마지막에 널 문자를 포함하고 있음** (명시하지 않지만)
 - ✓ 문자 '**A**' 와 문자열 "**A**"의 차이



2) 문자열 저장 및 기본 입출력

- 널(null) 문자
 - 예) 선언문의 초기화

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

||

```
char str[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

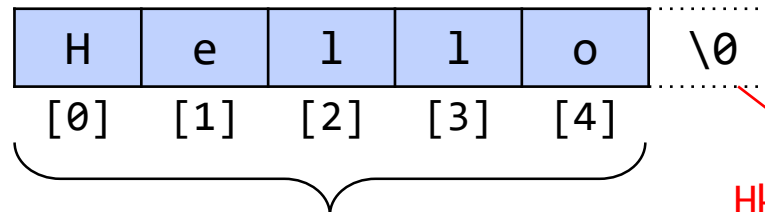
str	H	e	l	l	o	\0
	[0]	[1]	[2]	[3]	[4]	[5]

2) 문자열 저장 및 기본 입출력

■ 문자 배열의 크기

- 문자 배열에 문자열을 저장하기 위해서는
배열의 크기가 **문자열의 길이보다 하나 더 커야 함**

```
char str1[6] = "Hello";    // 정상 작동  
char str2[5] = "Hello";    // 런타임 오류 유발
```



str2

배열 영역 벗어남
런타임 오류의 원인

2) 문자열 저장 및 기본 입출력

- C언어에서 문자열의 기준
 - 널 문자까지의 문자들의 묶음을 지칭
 - 주의) 배열의 크기와 관계없음
 - ✓ 배열은 단순히 저장 공간으로서의 역할
 - 문자열의 끝은 배열의 크기가 아니라 널 문자에 의해 결정
 - ✓ 입출력을 비롯한 모든 문자열 처리의 기준

2) 문자열 저장 및 기본 입출력

- **printf() 함수를 이용한 문자열 출력**
 - 문자열을 하나의 단위로 취급
 - 서식 지정자: %s
 - 인자: 문자열의 시작 주소(보통 문자 배열의 이름)

```
char str[8] = "Hello";    // 문자 배열
printf("%c", str[2]);    // 문자 출력 (배열 원소 전달)
printf("%s", str);        // 문자열 출력 (배열 이름 전달)
```

2) 문자열 저장 및 기본 입출력

- 출력 시 널 문자의 역할
 - 느낌표의 출력 위치에 주목하자

```
char str[20] = "Hello World";
int i;
for( i=0 ; i < 20 ; ++i )
    printf("%c", str[i]);
printf("!!\n");
```

결과:

Hello World !!

```
char str[20] = "Hello World";

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

결과:

Hello World!!

배열의 크기는 20인데, 왜 11자만 출력할까?

배열에서 초기화가 명시되지 않은 원소는 0(즉, '\0')으로 초기화 됨
널 문자는 화면에 공백처럼 출력, but 공백 문자와는 다름

2) 문자열 저장 및 기본 입출력

출력 시 널 문자의 역할

- 인자로 전달된 주소의 문자부터 널 문자 전까지 출력
 - ✓ 배열 크기만큼 출력하는 것이 아님
 - ✓ printf 함수는 배열의 크기를 모름

```
char str[20] = "Hello World";
int i;
for( i=0 ; i < 20 ; ++i )
    printf("%c", str[i]);
printf("!!\n");
```

결과:

Hello World !!

```
char str[20] = "Hello World";

printf("%s!!\n", str);
printf("%s!!\n", str+5);
```

결과:

Hello World!!
World!!

2) 문자열 저장 및 기본 입출력

- `scanf()` 함수를 이용한 문자열 입력
 - 서식 지정자: `%s`
 - 인자: 문자열을 저장할 시작 주소(보통 배열의 이름)
 - 사용자로부터 입력 받은 문자열을 인자로 전달된 주소부터 차례로 저장

```
char str[20];  
  
scanf("%s", str);  
printf("%s!!\n", str);  
scanf("%s", str+5);  
printf("%s!!\n", str);
```

실행 예시

Hello	→ 입력
Hello!!	→ 출력
World	→ 입력
HelloWorld!!	→ 출력

2) 문자열 저장 및 기본 입출력

- **scanf의 %s 서식**

- 개행 문자, 공백 문자, 탭 문자 직전까지를 하나의 문자열로 인식
- 마지막에 널 문자를 자동으로 추가

```
char str[20];  
  
scanf("%s", str);  
printf("%s!!\n", str);
```

실행 예시

Hello World

→ 입력

Hello!!

→ 출력

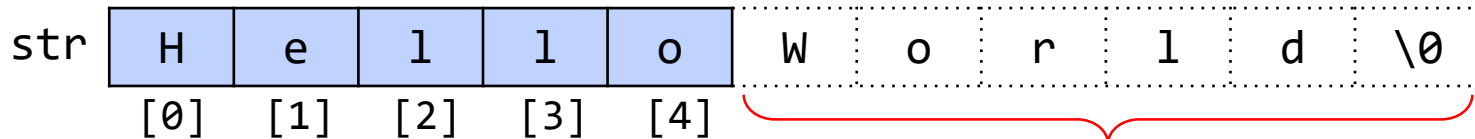
2) 문자열 저장 및 기본 입출력

- 주의 사항

- 문자열(**널 문자 포함**)을 저장할 충분한 공간이 확보되어 있어야 함 (모든 배열에 공통적인 사항)

```
char str[5]; // 크기 5인 배열  
scanf("%s",str);
```

- 만약, 사용자가 "HelloWorld"를 입력한다면?
 - ✓ 배열 범위를 벗어난 메모리 영역에 입력 받은 문자 저장
 - ✓ 위 문자열을 저장하기 위해서는 배열의 크기가 **11**이상 이어야 함 (왜 10 이상이 아니고, 11 이상일까?)



배열 범위 초과
(런타임 오류의 유발)

2) 문자열 저장 및 기본 입출력

- [예제 10.2] 다음 프로그램을 작성하여 실행해보자
 - 크기가 6인 문자 배열 str을 선언
 - 사용자로부터 문자열 "Hello"를 입력 받아 str에 저장
 - 문자열 str을 화면에 출력
 - str[5]에 물음표 문자 '?' 대입
 - 문자열 str을 화면에 출력

✓ 왜 이런 출력 결과가 나오는지 생각해보자.

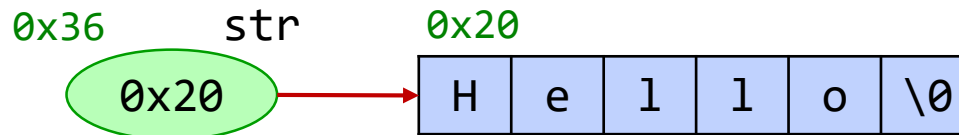
목차

- 1) 문자열 개요
- 2) 문자열 저장 및 기본 입출력
- 3) **문자열과 포인터**
- 4) 문자열의 배열
- 5) 문자열 및 문자 처리 함수
- 6) 문자열 및 문자 입출력

3) 문자열과 포인터

- **문자형 포인터를 활용한 문자열 처리문**
 - 문자형 포인터를 사용한 간단한 코드
 - ✓ 문자형 포인터 str을 선언하고, 문자열 (상수) "Hello"를 가리키도록 초기화
 - ✓ str에 주소가 저장되어 있으므로, printf의 %s 서식 이용해 출력

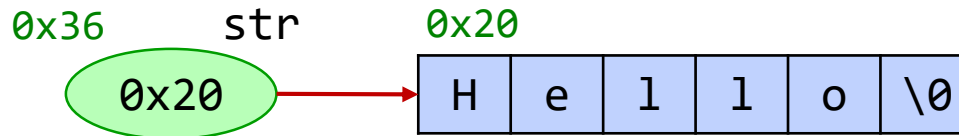
```
char *str = "Hello";           // 초기화  
  
printf("%s!!\n", str);         // 출력
```



3) 문자열과 포인터

- 문자형 포인터를 배열처럼 사용하기
 - 배열과 포인터의 관계 (9장에서 학습)를 이용

```
char *str = "Hello";  
  
for (i=0 ; i<5 ; i++)  
    printf("%c", str[i]); // 문자 출력
```



3) 문자열과 포인터

- [예제 10.3] 다음 프로그램을 작성하시오.
 - 문자 포인터 변수 pc를 선언하고, 다음 문자열로 초기화
 - ✓ "To be, or not to be : that is the question"
 - 반복문을 사용하여 영어 소문자 't' 가 몇 번 나오는 지 계산
 - ✓ 힌트: 널 문자 여부를 반복 종료 조건으로 사용
 - 다음과 같이 출력
 - ✓ 힌트: 큰 따옴표와 작은 따옴표를 출력하기 위해 `₩"` 과 `₩'` 사용

문자열 "To be, or not to be : that us the question"에 문자 't'가 6번 나타남

3) 문자열과 포인터

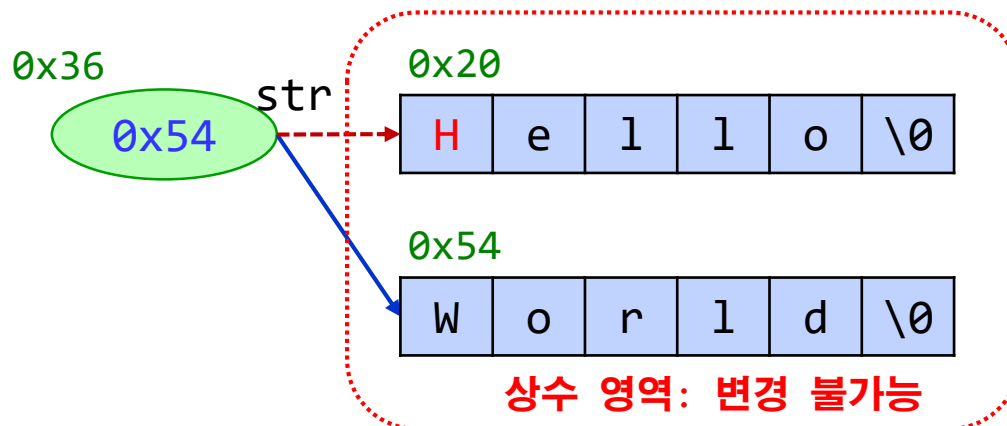
- 문자 배열과 문자열 상수 비교

- "Hello"는 문자열 상수로, 사용자 프로그램에서 **변경 불가능**
- 반면, str은 사용자 변수로 값을 변경할 수 있음

```
char *str = "Hello";
```

```
str[0] = 'h'; // 변경 불가능 (런타임 오류 발생)
```

```
str = "World"; // str에 저장된 값 변경 (가능)
```



3) 문자열과 포인터

- 문자 배열과 문자형 포인터 비교
 - 외우려고 하지 말고, 메모리 그림을 그려서 이해하자!!!

```
char str1[6] = "Hello";

printf("%c", str1[0]); // 0
printf("%s", str1);    // 0

str1[0] = 'h';          // 0
scanf("%s", str1);      // 0

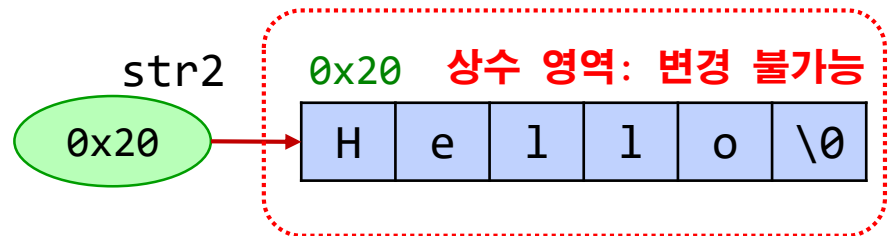
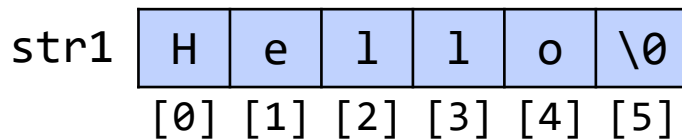
str1 = "World";         // X
```

```
char *str2 = "Hello";

printf("%c", str2[0]); // 0
printf("%s", str2);    // 0

str2[0] = 'h';          // X
scanf("%s", str2);      // X

str2 = "World";         // 0
```

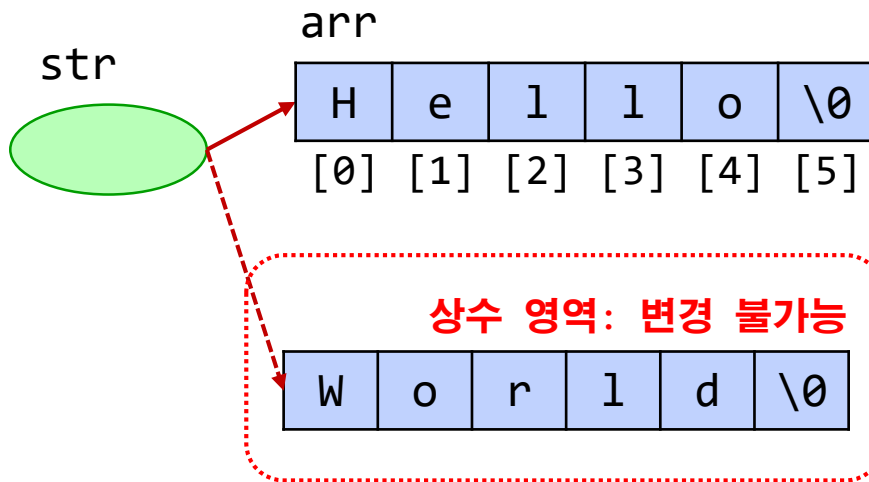


3) 문자열과 포인터

주의!!

- str이 포인터여서 문자 변경이 안 되는 것이 아님
- **str이 가리키는 영역의 성질에 따라 달라짐**

변수 영역: 변경 가능



```
char arr[6] = "Hello";  
char *str = arr;
```

```
str[0] = 'h';           // 0  
scanf("%s", str);      // 0
```

```
str = "World";          // 0
```

```
str[0] = 'w';           // X  
scanf("%s", str);      // X
```

목차

- 1) 문자열 개요
- 2) 문자열 저장 및 기본 입출력
- 3) 문자열과 포인터
- 4) 문자열의 배열**
- 5) 문자열 및 문자 처리 함수
- 6) 문자열 및 문자 입출력

4) 문자열의 배열

- 다수의 문자열 처리하기: 문자 배열을 여러 개 사용

```
char num0[5] = "zero";  
char num1[5] = "one";  
char num2[5] = "two";  
printf("%s\n", num0);  
printf("%s\n", num1);  
printf("%s\n", num2);
```

num0	z	e	r	o	\0
num1	o	n	e	\0	
num2	t	w	o	\0	
	[0]	[1]	[2]	[3]	[4]

- 문자열이 많아지면 불편

4) 문자열의 배열

- 다수의 문자열 처리하기: 문자열의 배열 (문자 배열을 배열로 묶기)
 - 2차원 문자 배열 이용
 - ✓ num[0], num[1], num[2]의 자료형은 char *

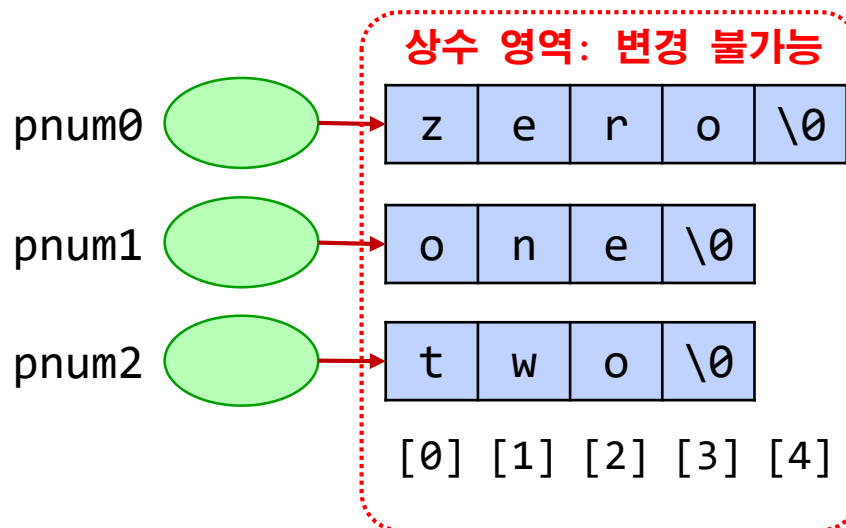
```
int i;  
char num[3][5] = {"zero", "one", "two"};  
for( i=0; i < 3; ++i )  
    printf("%s\n", num[i]);
```

num[0]	z	e	r	o	\0
num[1]	o	n	e	\0	
num[2]	t	w	o	\0	
	[0]	[1]	[2]	[3]	[4]

4) 문자열의 배열

- 다수의 문자열 처리하기: 문자형 포인터를 여러 개 사용

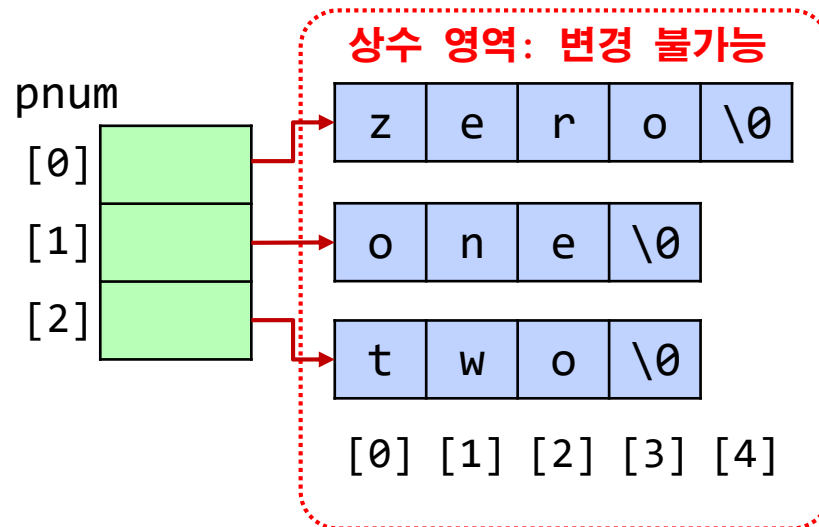
```
char *pnum0 = "zero";  
char *pnum1 = "one";  
char *pnum2 = "two";  
printf("%s\n", pnum0);  
printf("%s\n", pnum1);  
printf("%s\n", pnum2);
```



4) 문자열의 배열

- 다수의 문자열 처리하기: 문자 포인터 배열(포인터를 배열로 묶기)

```
int i;  
char *pnum[3] = {"zero", "one", "two"};  
for( i=0; i < 3; ++i )  
    printf("%s\n", pnum[i]);
```



4) 문자열의 배열

- **[예제 10.4] 다음 프로그램을 작성하시오.**
 - 3 X 20 크기의 2차원 문자배열을 선언하고, 다음 문자열로 초기화
 - ✓ "Time is gold"
 - ✓ "No pain no gain"
 - ✓ "No sweat no sweet"
 - 2중 반복문을 사용하여, 각 문자열에서 영어 소문자 'a' 가 몇 번 나오는 지 출력
- **[예제 10.5]**
 - 위 프로그램을 2차원 문자 배열대신 **문자 포인터 배열**을 사용하여 구현하시오.

목차

- 1) 문자열 개요
- 2) 문자열 저장 및 기본 입출력
- 3) 문자열과 포인터
- 4) 문자열의 배열
- 5) 문자열 및 문자 처리 함수**
- 6) 문자열 및 문자 입출력

5) 문자열 및 문자 처리 함수

- 문자열 처리 표준 함수

- C언어에서는 문자열 처리에 관련된 다양한 표준 함수 제공
- 대부분 <string.h> 헤더 파일에 함수의 원형 선언되어 있음
 - ✓ 이 헤더파일을 include 시켜야 함

#include <string.h>

- 대부분 문자열 처리 함수의 코드를 작성하는 것은 어렵지 않지만, 이미 구현되어 있는 표준 함수를 사용하는 것이 편리
- 다만, 정확한 사용법을 익혀야 함

5) 문자열 및 문자 처리 함수

- **문자열의 길이 구하기 1 (직접 구현)**
 - 널 문자와 반복문을 이용하여 구할 수 있음

```
char str[20] = "Hello World";  
int i = 0;  
  
while ( str[i] )      // 널문자가 아닌 동안  
    ++i;              // i 값 증가  
printf("length: %d\n", i);
```

결과:

length: 11

5) 문자열 및 문자 처리 함수

- 문자열의 길이 구하기 2 (표준 함수 strlen 이용)
 - 원형: unsigned int strlen(char *s)
 - 기능: 문자열 s의 길이 반환

```
#include<stdio.h>
#include<string.h> // strlen() 함수가 선언된 헤더 파일

int main(){
    char str[20] = "Hello World";
    printf("length: %d\n", strlen(str));
    return 0;
}
```

결과:

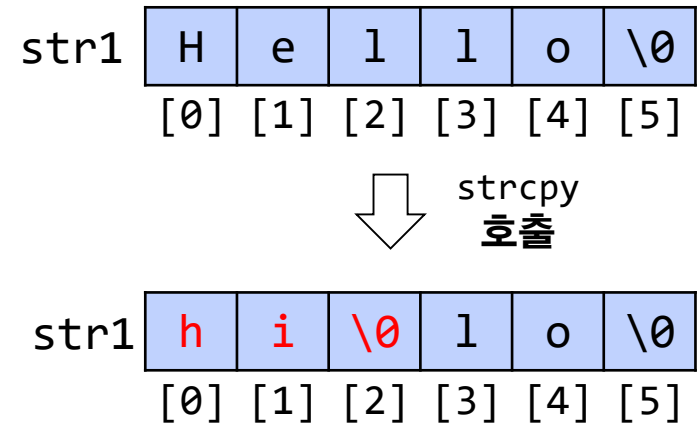
length: 11

5) 문자열 및 문자 처리 함수

■ 문자열 복사하기

- `char *strcpy(char *dest, char *src)`
- 기능: dest의 공간에 src의 문자열 복사 (문자열 대입)
src는 변화 없음

```
char str1[6] = "Hello";  
  
strcpy( str1, "hi");  
  
printf("str1: %s!!\n", str1);  
결과:  
str1: hi!!
```



- 참고) `strncpy()` 함수: 복사할 문자열의 길이를 지정하는 함수

5) 문자열 및 문자 처리 함수

- **strcpy(dest, src) 사용 시 주의사항**
 - dest의 공간이 **src의 문자열 길이+1(널 문자) 이상** 이어야 함
 - ✓ 그렇지 않으면, 런타임 오류의 원인

```
char a[10], b[5] = "hi";
```

```
char *c = NULL;
```

```
strcpy( a, b);    // 정상 작동
```

```
strcpy( b, "Hello");    // 런타임 오류 유발
```

```
strcpy( c, "Hello");    // 런타임 오류 유발
```

```
c = a;
```

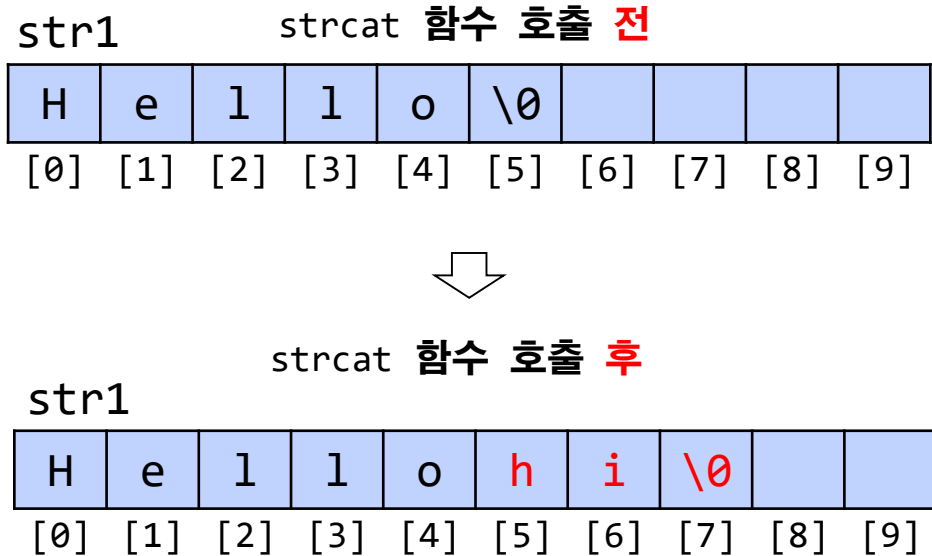
```
strcpy( c, "Hello");    // 정상 작동
```

5) 문자열 및 문자 처리 함수

▪ 문자열 접합하기

- `char *strcat(char *dest, char *src)`
- 기능: 문자열 dest 뒤에 src의 문자열 **접합**
src는 변화 없음

```
char str1[10] = "Hello";  
strcat( str1, "hi");  
printf("str1: %s!!\n", str1);  
결과:  
str1: Hellohi!!
```



- 참고) `strncat()` 함수: 접합할 문자열의 길이를 지정하는 함수

5) 문자열 및 문자 처리 함수

- **strcat(dest, src) 사용 시 주의사항**
 - dest에 접합 결과를 저장하기에 충분한 공간이 할당되어 있어야 함
 - ✓ 그렇지 않으면, 런타임 오류 유발

```
char s1[10] = "Hello";  
char s2[5] = "hi";  
char *s3 = NULL;  
char s4[20];  
  
strcat( s2, s1);      // 런타임 에러 유발  
strcat( s3, s1);      // 런타임 에러 유발  
strcat( s4, s1);      // 런타임 에러 유발 (why?)
```

5) 문자열 및 문자 처리 함수

■ 문자열 비교하기

- `int strcmp(char *lhs, char *rhs)`
- 기능: 사전 순으로 lhs와 rhs를 **비교**하여
문자열 lhs < rhs이면 **음수**,
문자열 lhs == rhs이면 **0**,
문자열 lhs > rhs이면 **양수** 반환
✓ 어떤 음수, 어떤 양수를 반환하는지는 컴파일러마다 다를 수 있음

- 문자열 비교는
처음부터 문자 별로 비교

h	i	\0
---	---	----

|| ✓

h	e	l	l	o	\0
---	---	---	---	---	----

[0] [1] [2] [3] [4] [5]

- 참고) `strncmp()` 함수: 비교할 문자열의 길이를 지정하는 함수

5) 문자열 및 문자 처리 함수

- 문자열 비교 예제

```
char s1[50] = "hi", s2[50] = "hello";  
int cmp_result = strcmp(s1, s2); // 문자열 비교  
  
if( cmp_result < 0 )  
    printf("%s가 %s보다 앞에 있습니다.\n", s1, s2);  
else if( cmp_result == 0 )  
    printf("%s가 %s와 같습니다.\n", s1, s2);  
else // cmp_result > 0  
    printf("%s가 %s보다 뒤에 있습니다.\n", s1, s2);
```

결과:

hi가 hello보다 뒤에 있습니다.

h	i	\0
---	---	----

||



h	e	l	l	o	\0
---	---	---	---	---	----

[0] [1] [2] [3] [4] [5]

5) 문자열 및 문자 처리 함수

- 문자열 비교 결과의 추가 예시
 - ✓ 문자는 단순히 아스키 코드 값 비교

```
char *str = "hi";
```

```
strcmp(str, str);    ⇒ 문자열 동일
```

```
strcmp(str, "hi");   ⇒ 문자열 동일
```

```
strcmp(str, "Hi");   ⇒ 소문자가 대문자보다 큼: 'h' > 'H'
```

```
strcmp("hi", ".");   ⇒ 첫 문자끼리 비교: 'h' > '.'
```

```
strcmp(str, "hi~");  ⇒ hi까지 동일. 다음 문자 '\0' < '~'
```

```
strcmp("hi", "high"); ⇒ hi까지 동일. 다음 문자 '\0' < 'g'
```

5) 문자열 및 문자 처리 함수

- [예제 10.6] 사용자로부터 두 개의 문자열 A와 B를 입력 받아 다음 과정을 수행하는 프로그램을 작성하시오.
 - 1) 문자열 A와 B의 길이를 각각 출력
 - 2) A와 B 중 사전 순으로 빠른 문자열 출력
 - 3) ABA 형태의 새로운 문자열 C를 생성하고 출력
- A와 B의 길이는 20 이내이고, 공백, 탭, 개행 문자는 없다고 가정
- 두 문자열은 서로 다르다고 가정

입력 예시

```
welcome  
helloworld!!
```

출력 예시

```
7 12  
helloworld!!  
welcomehelloworld!!welcome
```

5) 문자열 및 문자 처리 함수

- 10진수로 표현된 문자열을 수로 변환

- `int atoi(char *str)` : `int`형으로 계산하여 반환
- `long atol(char *str)` : `long`형으로 계산하여 반환
- `double atof(char *str)` : `double`형으로 계산하여 반환
- `<stdlib.h>`에 원형 선언

실행 결과

```
printf("%d\n", atoi("123") );  
printf("%d\n", atoi("-123") );  
  
printf("%f\n", atof("-123") );  
printf("%f\n", atof("123.45") );
```

```
123  
-123  
  
-123.000000  
123.450000
```

5) 문자열 및 문자 처리 함수

- 주요 문자열 처리 함수 (요약)

함수 원형	함수 기능 설명
<code>unsigned int strlen(s)</code>	문자열 <code>s</code> 의 길이 반환
<code>char *strcpy(s1, s2)</code>	문자열 <code>s1</code> 에 <code>s2</code> 를 복사
<code>char *strcat(s1, s2)</code>	문자열 <code>s1</code> 의 끝에 <code>s2</code> 를 접합
<code>int strcmp(s1, s2)</code>	문자열 <code>s1</code> 과 <code>s2</code> 를 사전 순으로 비교
<code>int atoi(s)</code>	문자열(<code>s</code>)로 표현된 수를 <code>int</code> 형, <code>long</code> 형, <code>double</code> 형으로 계산하여 반환 예) <code>atoi("12")</code> 는 정수 12 반환
<code>long atol(s)</code>	
<code>double atof(s)</code>	

목차

- 1) 문자열 개요
- 2) 문자열 저장 및 기본 입출력
- 3) 문자열과 포인터
- 4) 문자열의 배열
- 5) 문자열 및 문자 처리 함수
- 6) 문자열 및 문자 입출력

6) 문자열 및 문자 입출력

▪ 입출력 함수

- printf 와 scanf : 다양한 기능을 가진 범용 입출력 함수
 - ✓ 함수의 크기가 크고, 속도 느림
- C언어에서는 문자열과 문자에 특화된 입출력 함수 제공
 - ✓ 속도 빠르고, 문자 또는 문자열 입출력에 적합
 - ✓ 문자열 입출력함수: puts, gets (gets_s, fputs)
 - ✓ 문자 입출력 함수: putchar, getchar
- 위 함수들은 모두 <stdio.h>에 선언되어 있음

6) 문자열 및 문자 입출력

- 문자열 출력 함수: `int puts(char *str)`
 - `str`이 가리키는 문자열을 화면에 출력하고, **마지막에 '\n' 출력**
 - 반환 값: 출력에 성공하면 음수가 아닌 값, 실패하면 EOF
 - ✓ 참고) EOF (End Of File): 파일의 끝을 나타내는 상수로 정수 -1의 값을 가짐(14장에서 학습)

```
char str[10] = "Hi World";  
int ret=1;  
  
ret = puts(str);  
printf("return: %d\n", ret);
```

실행 결과

**개행문자 '\n'이
출력되어 줄이 바뀜**

```
Hi World  
return: 0
```

- 위 코드에서 `puts` 대신 `printf`를 사용하여 `str`을 출력해보자.
차이점이 있는가?

6) 문자열 및 문자 입출력

- 문자열 입력 함수: `char *gets(char *s)`
 - 사용자로부터 문자열을 입력 받아, `s`가 가리키는 메모리 영역에 저장하고, 포인터 `s`를 리턴
 - ✓ 엔터('\n')가 입력될 때까지 입력된 모든 문자들을 저장
 - ✓ 마지막에 입력된 '\n'은 무시하고, 맨 뒤에 '\0'를 붙임
 - ✓ 문자열을 저장할 충분한 메모리 공간이 확보되어 있어야 함

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

```
gets(str); // 또는 gets_s(str, 10);  
printf("str: %s!!", str );
```

실행 예시

```
Hi World    ← 입력  
str: Hi World!!
```

- ✓ 참고) **gets_s() 함수**: gets() 함수의 보안 버전으로, 문자열을 저장할 배열 크기를 인자로 전달

6) 문자열 및 문자 입출력

- (참고) 보안 상의 문제로 gets() 함수는 표준에서 제외되고, gets_s() 함수가 표준에 추가됨
 - Visual Studio의 경우 2015 버전부터 gets() 함수 지원 안 함
 - ✓ gets_s 사용 ()
 - gcc의 경우, 아직 gets()를 지원하고, gets_s()는 지원 하지 않음
 - ✓ gcc를 사용하는 온라인 채점 시스템에서는 gets() 사용
 - 또는 대안으로 fgets 함수 사용(14장에서 학습)
 - ✓ 개행 문자도 문자열에 저장

```
char str[10];  
  
fgets(str, 10, stdin);           // 사용법  
printf("str: %s!!", str );
```

6) 문자열 및 문자 입출력

■ 문자 입출력 함수:

- int `putchar`(int c) : 인자 c의 문자를 화면에 출력
- int `getchar`(void) : 사용자로부터 입력된 문자 반환
- 성공하면 입출력된 문자 반환, 실패하면 EOF 반환
 - ✓ 참고) EOF (End Of File): 파일의 끝을 나타내는 상수로 정수 -1의 값을 가짐(14장에서 학습)

```
int c;  
  
c = getchar();  
putchar(c);
```

실행 예시

```
H           ← 입력  
H
```

```
int a, b, c;  
scanf("    %d %d %d", &a, &b, &c);
```

```
int a, b, c;  
  
scanf("      %d ", &a);  
scanf("%d", &b);  
scanf("    %d  ", &c);
```

입력

70 80

90

```
char A[50];
```

```
gets(A);  
printf("<%s>",A);
```

□□Today is a gift.□□↵
<엔터>

<□□Today is a gift.□□>

버퍼에 안 남음

```
char A[10];
```

```
scanf("%s",A);  
printf("<%s>",A);
```

□□Today□□↵
<Today>

버퍼에 남음

□□↵

버림
□
남김
□□

```
char A[10],B[10];  
scanf("%s",A); // scanf("%d",a);  
gets(B);  
printf("%d-%d-%d",B[0],B[1],B[2]);
```

□□Today□□↵

32-32-0

32는 공백문자
0은 null 문자
엔터를 읽어서 0으로 바꾸어 저장

```
char A[10],B[10]; (X) 실습7번  
scanf("%s",A); 엔터 남김  
//getchar( );  
gets(B); 원하는 문자열 대신  
엔터를 받아들임
```

```
char A[10],B[10]; (X)  
int a;  
scanf("%d",a); 엔터 남김  
gets(B);
```

3. 버퍼에 남김
hello

```
char A[10],B[10]; (0)
```

```
scanf("%s",A); 엔터남김  
scanf("%s",B); 입력 문자 앞의 공백문자 제거한다. 엔터 남김
```

```
char A[10],B[10]; (0)
```

```
gets(A); 엔터 안 남김  
scanf("%s",B); 엔터 남김
```



```
int a; (0)
scanf("%d",a); 엔터 남김
scanf("%d",a); 엔터 남김
```

```
int a; (0)
char A[10];
scanf("%d",&a); 엔터 남김
scanf("%s",A); 입력 문자 앞의 공백문자 제거한다. 엔터 남김
```

```
int a; (X)
char ch;
scanf("%d",a); 엔터 남김 // getchar();
scanf("%c",ch); 엔터 남김
```

```
int a; (X)
char ch[10];
scanf("%d",a); 엔터 남김
//getchar( );
gets(ch); 엔터 처리
```

strcat() crashes if using same array as both parameters

```
s          d
a b c d e f \0
      s          d
a b c d e f a
          s          d
a b c d e f a b
          s          d
a b c d e f a b c
          s          d
a b c d e f a b c d e
          s          d
a b c d e f a b c d e f
          s          d
a b c d e f a b c d e f a
```

```
char *strcat(char *dest,
             const char *src )
{
    char *ret = dest;
    if (dest && src)
    {
        while (*dest != 0)
            ++dest;
        while (*src != 0)
            *dest++ = *src++;
        *dest = 0;
    }
    return ret;
}
```

strcat's definition does not allow the two strings to overlap.

계속 반복 되므로

strcpy(i, i+1) 계속 반복 되므로

-
- OJ에서 run-time error가 발생하는 이유
VS에서 예시에서는 잘 실행되지만, 테스트 벡터에 따라
무한반복하기 때문이다. 반복문 부분을 다시 검토하기 바랍니다.
 - 배열이 부족한 경우에도 발생
 - `#include <stdio.h>`
`max()` 함수를 만들면 중복이 되어, 컴파일 오류가 발생합니다.