
# 프로그래밍 기초

## 12장. 문자와 문자열 Part 2

# 학습목표

- 문자와 문자열을 이해하고 설명할 수 있다.
  - 문자와 문자열의 표현과 저장 방법
- 문자와 문자열 입출력을 이해하고 설명할 수 있다.
  - scanf(), printf(), getchar(), putchar(), getche(), getch(), putch()를 사용하여 문자 입출력
  - scanf(), printf(), gets(), puts()를 사용하여 문자열 입출력
- **문자열 관련 함수를 이해하고 설명할 수 있다.**
  - 문자열 비교 함수 strcmp(), strncmp()를 사용하여 문자열 비교
  - 문자열 연결 함수 strcat(), strncat()를 사용하여 문자열 연결
  - 문자열 토큰 추출 함수 strtok()를 사용하여 문자열에서 토큰 추출
  - 문자열 관련 함수 strlen(), strspn(), strcspn()의 사용 방법 이해
  - 문자열 관련 함수 strlwr(), strupr()의 사용 방법 이해
  - 문자열 관련 함수 strstr(), strchr()의 사용 방법 이해
- **여러 개의 문자열을 처리 방법에 대해 이해하고 설명할 수 있다.**
  - 문자 포인터 배열 방법과 2차원 문자 배열 방법의 차이
  - 명령행 인자의 필요성과 구현 방법 이해

# 다양한 문자열 라이브러리 함수

- **문자열 라이브러리 함수**
  - 헤더파일 <string.h>에 선언된 라이브러리 함수로 제공
  - 문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리 함수

- **함수에서 사용되는 자료형: 64비트 윈도우 시스템인 경우**
  - size_t: 비부호 정수 long long형(unsigned __int64)
  - void *: 아직 정해지지 않은 다양한 포인터를 의미

| 문자열 관련 함수 원형 | 설명 |
|---|---|
| size_t strlen(const char *str) | • str의 문자열 길이 반환(NULL 문자 제외) |
| void *memset(void *dest, int ch, size_t count) | • 지정한 메모리 영역(dest)을 원하는 바이트 값(ch)으로 count 만큼 채움<br>• 메모리 초기화 용도 |
| void *memcpy(void *dest, const void *src, size_t num) | • src의 메모리 내용을 dest 메모리 영역에 바이트 단위로 복사<br>• 메모리 복사 |
| void *memchr(const void *str, int c, size_t n) | • 메모리(str)에서 n 바이트까지 특정 값(c)를 찾고 그 위치 반환 |
| int memcmp(const void *str1, const void *str2, size_t n) | • 메모리 str1과 str2를 첫 n 바이트까지 비교<br>• 같으면 0, 다르면 음수 또는 양수 반환 |
| void *memmove(void *dest, const void *src, size_t n) | • 메모리 복사 (src와 dest 영역이 겹쳐도 안전하게 복사됨: 임시 버퍼 사용) |

# 함수 strcmp()와 strncmp()

- **두 문자열을 비교하는 함수: strcmp(), strncmp()**

문자열 비교 함수: 헤더파일 string.h 삽입

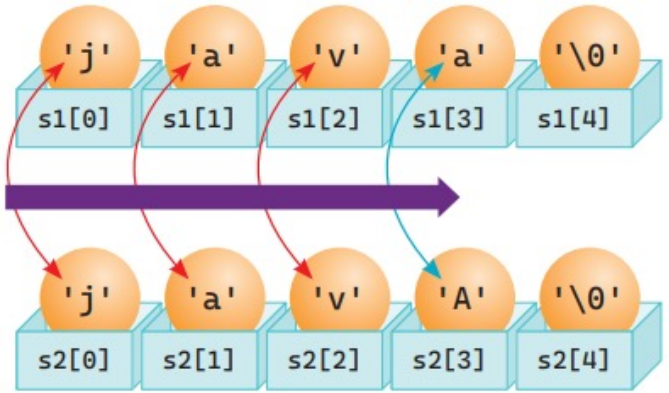
```
int strcmp(const char * s1, const char * s2);
```

두 인자인 문자열에서 같은 위치의 문자를 앞에서부터 다를 때까지 비교하여 같으면 0 을 반환하고, 앞이 크면 양수를, 뒤가 크면 음수를 반환한다.

```
int strncmp(const char * s1, const char * s2, size_t maxn);
```

두 인자 문자열을 같은 위치의 문자를 앞에서부터 다를 때까지 비교하나 최대 n 까지만 비교하여 같으면 0을 반환하고, 앞이 크면 양수를, 뒤가 크면 음수를 반환한다.

- 두 문자열을 사전상의 순서로 비교하는 함수
- strncmp(): 두 문자를 비교할 문자의 최대 수를 지정하는 함수




- strcmp("a", "ab"): 음수
- strcmp("ab", "a"): 양수
- strcmp("ab", "ab"): 0

- strcmp("java", "javA"): 양수
  문자 a가 A보다 크므로 양수 반환

- strncmp("java", "javA", 3): 0
  인자 3인 문자 셋까지 비교하여 같으므로 0

ASCII 값
A: 65
a: 97

# strcmp() 예제 (실습)

- **strcmp()와 strncmp(): 문자열 비교 함수**
  - strcmp(str1, str2) 리턴값
    - str1 > str2 : 양수
    - str1 == str2: 0
    - str1 < str2 : 음수

<문자 빨셈 연산>

| | | | | |
|---|---|---|---|---|
| 'B' (66) | - | 'A' (65) | = | 1 (양수) |
| 'A' (65) | - | 'A' (65) | = | 0 |
| 'A' (65) | - | 'B' (66) | = | -1 (음수) |

<06strcmp.c>

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

int main()
{
    char str1[] = "apple";
    char str2[] = "orange";

    int result = strcmp(str1, str2);

    if (result == 0)
        printf("str1 == str2\n");
    else if (result < 0)
        printf("str1 < str2 (result: %d)\n", result);
    else
        printf("str1 > str2 (result: %d)\n", result);

    return 0;
}
```

```
str1 < str2 (result: -14) <-- 97('a') - 111('o')
```

# 함수 strlen(), memset(), memcpy()

- **strlen(): 문자열의 길이를 반환(string length)**
  - NULL 문자를 제외한 전체 문자열의 길이 반환

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

- **memset(): 배열 및 큰 규모의 메모리(구조체 등) 초기화(memory set)**

```
void *memset(void *b, int c, size_t len);
```

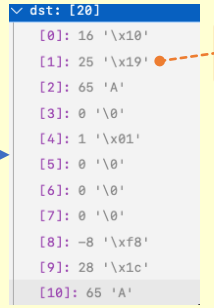
- **memcpy(): 배열 및 메모리 복사를 위한 함수(memory copy)**
  - 포인터 src 위치에서 dest로 n 바이트를 복사
  - 속도가 빠름

```
void *memcpy(void *dest, const void *src, size_t n);
```

# memcpy(), strcmp() 예제 (실습)

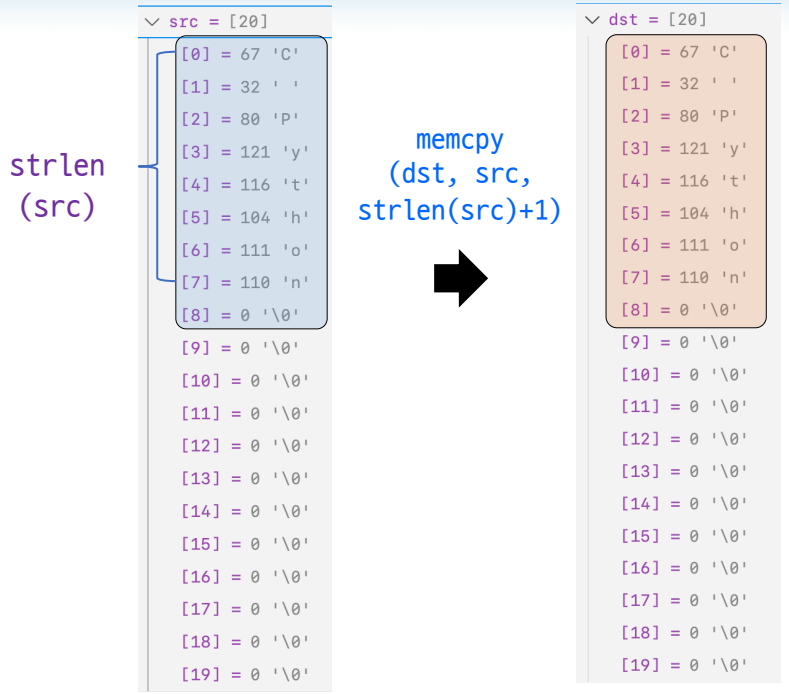
<06strfun.c>

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

int main(void)
{
    char src[20] = "C Python";
    char dst[20];

    memset(dst, 0, sizeof(dst)); // 메모리 초기화 또는 char dst[20] = {0};

    printf("%s\n", src);
    printf("%zu\n", strlen(src));
    memcpy(dst, src, strlen(src) + 1);
    printf("%s\n", dst);
    memcpy(dst, "안녕하세요!", strlen("안녕하세요!") + 1);
    printf("%s\n\n", dst);

    char* s1 = "C lang";
    char* s2 = "C lang";
    printf("strcmp(%s, %s) = %d\n", s1, s2, strcmp(s1, s2));
    s1 = "C lang";
    s2 = "C ";
    printf("strcmp(%s, %s) = %d\n", s1, s2, strcmp(s1, s2));
    printf("strcmp(%s, %s) = %d\n", s2, s1, strcmp(s2, s1));
    printf("strncmp(%s, %s, %d) = %d\n", s1, s2, 2, strncmp(s1, s2, 2));
    return 0;
}
```




garbage 값 저장됨
- 반드시 초기화 수행

NULL 문자까지 복사

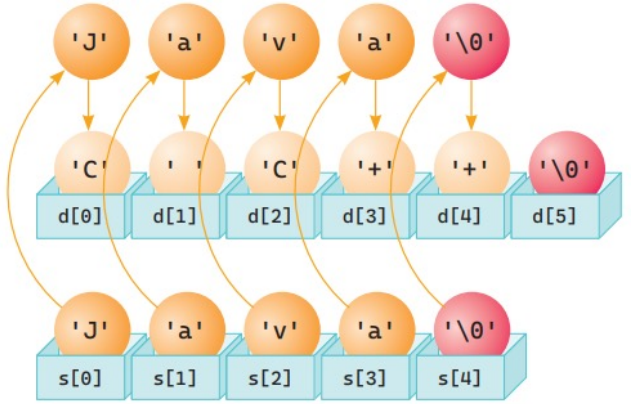



**실행 결과**

```
C Python
8
C Python
안녕하세요!

strcmp(C lang, C lang) = 0
strcmp(C lang, C ) = 108
strcmp(C , C lang) = -108
strncmp(C lang, C , 2) = 0
```

# 문자열 복사 함수: strcpy(), strncpy()

- **strcpy(dest, source): 문자열 복사 함수**
  - source의 문자열 전체를 dest에 복사
    - ➢ 마지막 NULL 문자까지 복사
    - ➢ 문자열의 길이 확인이 없이 복사: 버퍼 오버플로우 발생 가능




결과는 d에도 "java"가 저장된다.

```
char d[] = "C C++";
char s[] = "Java";
strcpy(d, s);
```

- **strncpy(dest, source, maxn)**
  - source 문자열을 처음부터 최대 maxn 까지 dest에 복사
  - 복사할 문자의 크기 제한을 고려시 사용

문자열 복사 함수

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

- 앞 문자열 dest에 처음에 뒤 문자열 null 문자를 포함한 source를 복사하여 그 복사된 문자열을 반환한다.
- 앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.

```
char * strncpy(char * dest, const char * source, size_t maxn);
```

- 앞 문자열 dest에 처음에 뒤 문자열 source 에서 n개 문자를 복사하여 그 복사된 문자열을 반환한다.
- 만일 지정된 maxn이 source의 길이보다 길면 나머지는 모두 널 문자가 복사된다. 앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.

```
errno_t strcpy_s(char * dest, size_t sizedest, const char * source);
errno_t strncpy_s(char * dest, size_t sizedest, const char * source, size_t maxn);
```

- 두 번째 인자인 sizedest는 정수형으로 dset 의 크기를 입력한다.
- 반환형 errno_t 는 정수형이며 반환값은 오류번호로 성공하면 0을 반환한다.
- Visual C++에서는 앞으로 함수 strcpy_s()와 strncpy_s()의 사용을 권장한다.

# 문자열 연결 함수: strcat(), strncat()

- **char *strcat(char *dest, char *source)**
  - dest 문자열 끝에 source 문자열의 NULL 문자까지 연결
  - 리턴값: 앞 문자열 주소(dest의 주소)를 반환
  - dest 배열은 연결된 문자열 전체(dest + source + NULL)를 모두 저장할 수 있는 공간이 필요

- **char *strncat(dest, source, maxn)**
  - source 문자열 중 처음부터 최대 maxn 개의 문자까지 dest 문자열 끝에 연결
    - ➢ 여기서 지정하는 문자 수는 NULL 문자를 제외한 수

문자열 연결 함수

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

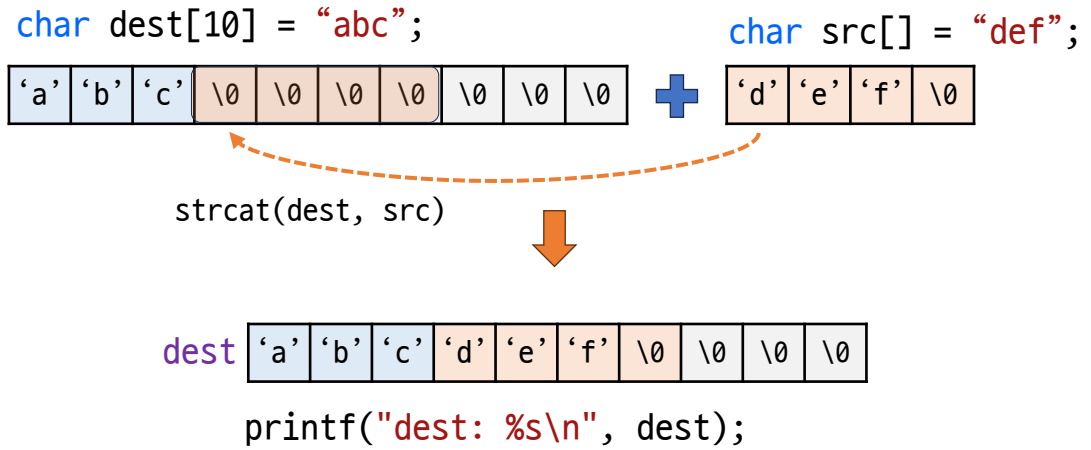
• 앞 문자열 dest에 뒤 문자열 source를 연결(concatenate)해 저장하며, 이 연결된 문자열을 반환하고 뒤 문자열은 수정될 수 없다.

```
char * strncat(char * dest, const char * source, size_t maxn);
```

• 앞 문자열 dest에 뒤 문자열 source중에서 n개의 크기만큼을 연결(concatenate)해 저장하며, 이 연결된 문자열을 반환하고 뒤 문자열은 수정될 수 없다.
• 지정한 maxn이 문자열 길이보다 크면 null 문자까지 연결한다.

```
errno_t strcat_s(char * dest, size_t sizedest, const char * source);
errno_t strncat_s(char * dest, size_t sizedest, const char * source, size_t maxn);
```

• 두 번째 인자인 sizedest는 정수형으로 dest의 크기를 입력한다.
• 반환형 errno_t는 정수형이며 반환값은 오류번호로 성공하면 0을 반환한다.
• Visual C++에서는 앞으로 함수strcat_s()와 strncat_s()의 사용을 권장한다.

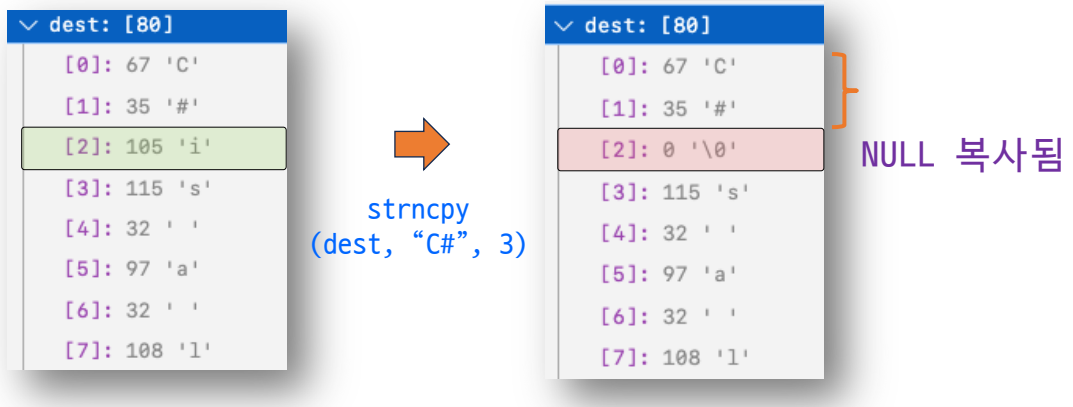


# strcpy()와 strcat() 예제 (실습)

<07strcpycat.c>

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

int main(void)
{
    char dest[80] = "Java";
    char source[80] = "C is a language.";

    printf("%s\n", strcpy(dest, source));
    // printf("%d\n", strcpy_s(dest, 80, source));
    printf("%s\n", dest);
    printf("%s\n", strncpy(dest, "C#", 2));

    printf("%s\n\n", strncpy(dest, "C#", 3)); // “C#” +’\0’ 복사
```

```c
    char data[80] = "C";

    strcat(data, " is ");
    printf("%s\n", data);
    // printf("%d\n", strcat_s(data, 80, " is "));
    // printf("%s\n", data);

    printf("%s\n", strncat(data, "a java", 2)); //“a ”까지 복사
    // printf("%d\n", strncat_s(data, 80, "a proce", 2));
    // printf("%s\n", data);

    printf("%s\n", strcat(data, "procedural "));
    printf("%s\n", strcat(data, "language."));

    return 0;
}
```




```
C is a language.
C is a language.
C#is a language.
C#

C is
C is a
C is a procedural
C is a procedural language.
```

# strcpy(), strcat() 사용시 주의점

- strcpy(char *dest, char* src), strcat(char *dest, char *src) 주의사항
  - dest는 포인터 변수를 사용할 수 없음: 메모리 공간(배열)이 필요
    - 복사 및 연결 결과를 저장할 수 있도록 충분한 메모리 공간을 확보해야 됨

```
char *stpcpy(char *dst, const char *src);
char *strcat(char *dst, const char *src);
```

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

int main()
{
    char dest[5] = "C";
    char *destc = "C";

    strcpy(dest, "Java language"); // 실행 시 오류발생
    strcpy(destc, " Java language"); // 실행 시 오류발생
    strcat(dest, " is a language."); // 실행 시 오류발생
    strcat(destc, " is a language."); // 실행 시 오류발생

    return 0;
}
```

# 문자열 분리 함수: strtok()

- **strtok(char *str, const char *delim) 함수:**
  - 문자열에서 구분자(delimiter)를 기준으로 분리하여 토큰(분리된 문자열)을 추출하는 함수
    - ➢ 구분자는 분리된 토근에서 제거됨
  - 첫 번째 인자인 str: 대상 문자열로 str은 문자 배열에 저장된 문자열을 사용
  - 두 번째 인자인 delim은 구분자로 문자의 모임인 문자열
    - ➢ 구분자로 여러 개의 문자를 지정할 수 있음: 예: ";,\t " 문자열 상수
  - 더 이상 분리할 문자열이 없으면 NULL 반환: 반복문을 이용하여 검색

문자열 분리 함수

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

- 앞 문자열 str에서 뒤 문자열delim을 구성하는 구분자를 기준으로 순서대로 토큰을 추출하여 반환하는 함수이며, 뒤 문자열 delim은 수정될 수 없다.

```
char * strtok_s(char * str, const char * delim, char ** context);
```

- 마지막 인자인 context는 함수 호출에 사용되는 위치 정보를 위한 인자이며, Visual C++에서는 앞으로 함수 strtok_s()의 사용을 권장한다.

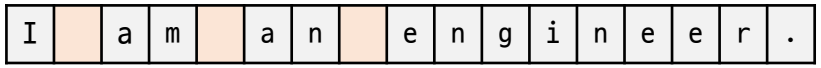
# 함수 strtok() 예제 (실습)

<strtok01.c>

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

int main()
{
    char s[] = "I am an engineer.";
    char *ptoken = NULL;

    ptoken = strtok(s, " ");
    while (ptoken != NULL)
    {
        printf("%s\n", ptoken);
        ptoken = strtok(NULL, " ");
    }
    return 0;
}
```

문자열에서 공백을 기준으로 첫 토큰을 추출

두 번째 호출부터는 원본 문자열 대신 NULL 입력




```
I
am
an
engineer.
```

# 참고: strtok(char *s, const char *delim) 소스 코드 분석

- 소스 링크: http://www.beedub.com/Sprite093/src/lib/c/string/strtok.c

```c
/*  Parse s into tokens separated by characters in DELIM.
    If s is NULL, the last string strtok() was called with is used. For example:
 */
char *strtok(char *s, const char *delim)
{
    static char *lasts;
    register int ch;

    if (s == 0) // s가 NULL 인 경우
        s = lasts;
    do
    {
        if ((ch = *s++) == '\0')
            return 0;
    } while (strchr(delim, ch));   // strchr(): 문자열(delim)내에 일치하는 문자(ch) 찾기
    --s;
    lasts = s + strcspn(s, delim); // strcspn(): 첫 번째 문자 일치의 오프셋 찾기
    if (*lasts != 0)
        *lasts++ = 0;
    return s;
}
```

s가 NULL인 경우, 기존에 저장된 문자열(lasts)을 s에 할당함

# 문자열 분리 함수: strtok()

- **문자열 "C and C++\t languages are best!“**
  - 구분자를 공백문자 하나인 “ ”로 지정
    - ➢ 토큰을 분리하면 C, and, C++\t, languages, are, best! 총 6개의 토큰이 추출
      - – **공백문자 분리자를 이용하여 토큰을 분리**
    - ➢ 구분자에 더 많은 문자를 삽입
      - – **분리된 토큰이 많아지거나, 분리된 토큰에서 구분자가 사라짐**

문자열: “C and C++\t language are best!”

- 구분자(delim)가 “ “인 경우: C, and, C++\t, language, are, best! (총 6개의 분리된 토큰)
- 구분자(delim)가 “ \t” 인 경우: C, and, C++, language, are, best! (총 6개의 분리된 토큰, ‘\t’ 제거)
- 구분자(delim)가 “ \t!” 인 경우: C, and, C++, language, are, best (총 6개의 분리된 토큰, ‘!’ 제거)

# 함수 strtok()의 사용방법

- **문장 ptoken = strtok(str, delimiter);**
  - 첫 토큰을 추출
  - 결과를 저장한 ptoken
    - NULL이면 더 이상 분리할 토큰이 없는 경우임
  - 계속 토큰을 추출 (ptoken != NULL) 로 검사
    - while 반복으로 추출된 토큰이 있는지 검사
    - strtok(NULL, delimiter)를 호출
      - **NULL을 첫 번째 인자**
      - **그 다음 토큰을 반환**

<strtok02.c>

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

int main(void)
{
    char str[] = "C and C++\t languages are best!";
    char *delimiter = " !\t";

    printf("구분자[%s]를 이용하여 토큰을 추출 >>\n", delimiter);
    char* ptoken = strtok(str, delimiter);

    while (ptoken) //(ptoken != NULL)
    {
        printf("%s\n", ptoken);
        ptoken = strtok(NULL, delimiter); //다음 토큰을 반환

    }

    return 0;
}
```

```
구분자[ !	]를 이용하여 토큰을 추출 >>
C
and
C++
languages
are
best
```

# 문자열의 변환 및 위치 검색

- **함수 strlwr(char *str): C 표준 라이브러리 아님**
  - 인자를 모두 소문자로 변환하여 반환 (string lower)
- **함수 strupr(char *str): C 표준 라이브러리 아님**
  - 인자를 모두 대소문자로 변환하여 반환 (string upper)

표 11-2 다양한 문자열 관련 함수

| 함수원형 | 설명 |
|---|---|
| `char * strlwr(char * str);`<br>`errno_t _strlwr_s(char * str, size_t strsize); //Visual C++ 권장함수` | 문자열 str을 모두 소문자로 변환하고 변환한 문자열을 반환하므로 str은 상수이면 오류가 발생하며, errno_t는 정수형의 오류번호이며, size_t도 정수형으로 strsize는 str의 길이 |
| `char * strupr(char * str);`<br>`errno_t _strupr_s(char * str, size_t strsize); //Visual C++ 권장함수` | 문자열 str을 모두 대문자로 변환하고 변환한 문자열을 반환하므로 str은 상수이면 오류가 발생하며, errno_t는 정수형의 오류번호이며, size_t도 정수형으로 strsize는 str의 길이 |
| `char * strpbrk(const char * str, const char * charset);` | 앞의 문자열 str에서 뒤 문자열 charset에 포함된 문자가 나타나는 처음 위치를 찾아 그 주소값을 반환하며, 만일 찾지 못하면 NULL 포인터를 반환 |
| `char * strstr(const char * str, const char * strsearch);` | 앞의 문자열 str에서 뒤 문자열 strsearch이 나타나는 처음 위치를 찾아 그 주소값을 반환하며, 만일 찾지 못하면 NULL 포인터를 반환 |
| `char * strchr(const char * str, char ch);` | 앞의 문자열 str에서 뒤 문자 ch가 나타나는 처음 위치를 찾아 그 주소값을 반환하며, 만일 찾지 못하면 NULL 포인터를 반환 |

Visual Studio 전용

# 문자열의 위치 검색

- **char *strpbrk(const char *str, const char *charset)**
  - str 에서 문자열 집합(charset)에 포함된 개별 문자 검색
  - 반환값: charset에 포함된 문자들 중 하나라도 처음으로 등장하는 문자의 위치(포인터)
    - 일치하는 문자열이 없으면 NULL 반환

- **char *strstr(const char *str, const char *strsearch)**
  - str 에서 검색어(strsearch)의 포함 여부 검색
  - 반환값: 문자열 str에서 일치하는 시작 위치 포인터 (특정 단어 찾기)
    - 일치하는 문자열이 없으면 NULL 반환

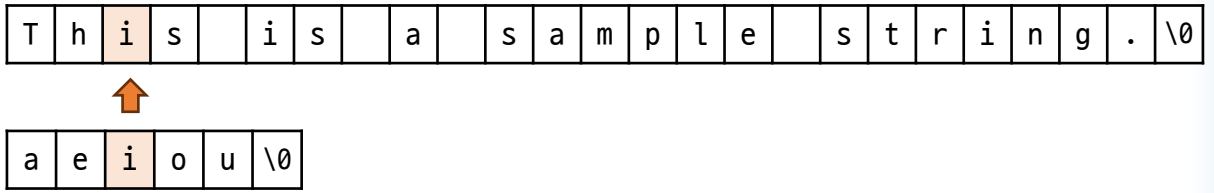
- **char *strchr(const char *str, char ch)**
  - str 내부에서 하나의 문자(ch) 검색
  - 반환값: 문자열 str에서 특정 문자 ch가 처음 나타나는 위치(포인터)
    - 일치하는 문자가 없으면 NULL 반환

# strpbrk() 예제

<strpbrk_ex.c>

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

int main() {
    char str[] = "This is a sample string.";
    char charset[] = "aeiou";

    char *p = strpbrk(str, charset);

    if (p != NULL)
    {
        printf("맨 처음 검색된 문자: %c\n", *p);
        printf("검색된 위치의 문자열: %s\n", p);
    }
    else
    {
        printf("모음 문자가 문자열에 없습니다.\n");
    }
    return 0;
}
```

"aeiou" 중 처음 발견되는 문자의 위치 리턴

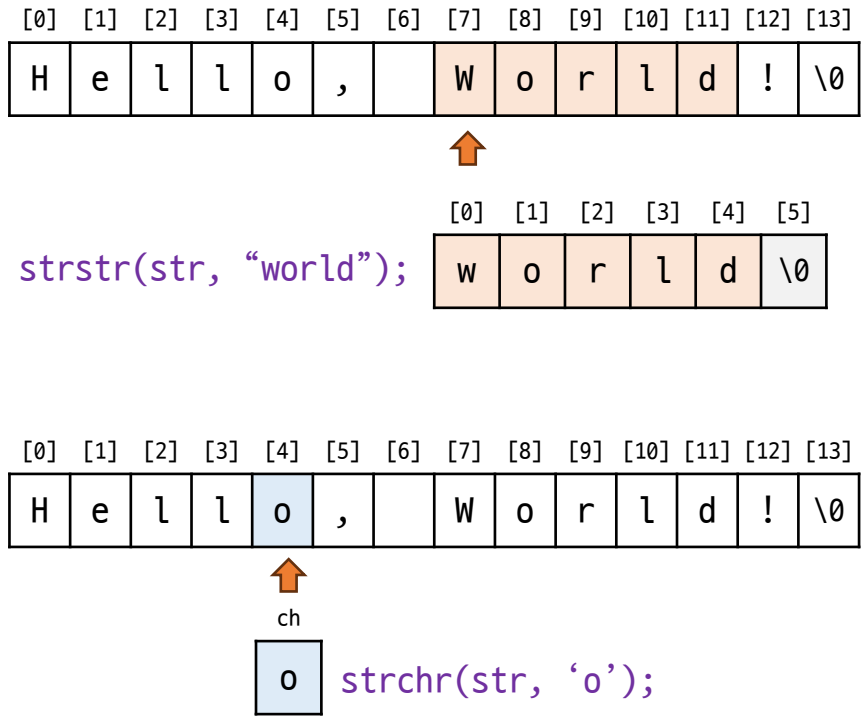



```
맨 처음 검색된 문자: i
검색된 위치의 문자열: is is a sample string.
```

# strstr(), strchr() 예제 (실습)

<strstr_strchr_ex.c>

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

int main() {
    char str[] = "Hello, world!";
    char *strsearch = "world";
    char ch = 'o ' ;
    char *p1, *p2;

    // 문장 내에서 단어 "world" 찾기
    p1 = strstr(str, strsearch);
    if (p1)
        printf("strstr(%s) 위치: %ld\n", strsearch, (p1 - str));
    else
        printf("strstr(%s) 없음\n", strsearch);

    // 문장 내에서 문자 'o' 찾기
    p2 = strchr(str, ch);
    if (p2)
        printf("strchr(%c) 위치: %ld\n", ch, (p2 - str));
    else
        printf("strchr(%c) 없음\n", ch);

    return 0;
}
```




```
strstr(world) 위치: 7
strchr(o) 위치: 4
```

# strlwr(), strupr() 예제

<09strstr.c>

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

char *_strupr(char *str); // Mac, Linux
char *_strlwr(char *str); // Mac, Linux

int main(void)
{
    char str[] = "Java 2022 Python C";

    printf("str: %s\n", str); // 원본 문자열 출력
    printf("_strlwr(): %s\n", _strlwr(str)); // 소문자로 변환
    printf("_strupr(): %s\n", _strupr(str)); // 대문자로 변환

    return 0;
}
```

```
str: Java 2022 Python C
_strlwr(): java 2022 python c
_strupr(): JAVA 2022 PYTHON C
```

Linux/Mac 사용자를 위한
_strupr(), _strlwr() 구현

```c
char *_strupr(char *str)
{
    for (int i = 0; i < strlen(str); i++)
    {
        *(str + i) = toupper(*(str + i));
    }
    return str;
}

char *_strlwr(char *str)
{
    while (*str)
    {
        *str = tolower(*str);
        str++;
    }
    return str;
}
```

# Lab 문자열을 역순으로 저장하는 함수 reverse() 구현 (실습)

<lab2reversestr.c>

- **함수 memcpy()를 사용**
  - 문자열 상수 char* str = "Hello World";
  - char s[50]; 일차원 배열 선언
    - ➢ memcpy()로 문자열 str을 s[50]에 복사
- **함수 reverse()를 호출**
  - 문자열을 역순으로 저장한 후 그 결과를 출력
  - 함수 reverse()
    - ➢ 문자열 배열을 역순으로 저장하는 함수
- **원본 문자열과 역순 문자열을 출력**

```
Hello World
dlroW olleH
```

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

void reverse(char []);

int main(void)
{
    char s[50];
    char* str = "Hello World";
    memcpy(s, str, strlen(str) + 1);
    printf("%s\n", s);

    reverse(s);
    printf("%s\n", s);
    return 0;
}

void reverse(char str[])
{
    int j = strlen(str)-1;
    for (int i = 0; i < j; i++, j--)
    {
        char c = str[i];
        str[i] = str[j];
        str[j] = c;
    }
}
```

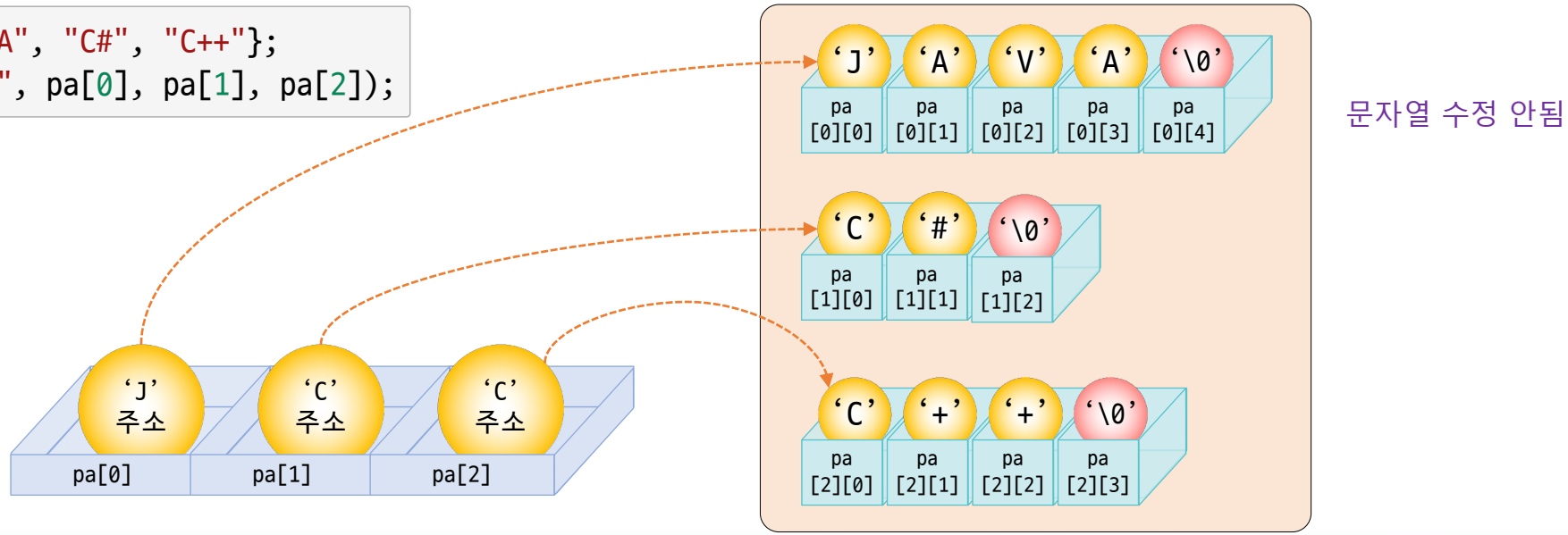
NULL 문자 추가
: strlen(str) + 1

# 여러 문자열 처리: 문자 포인터 배열

- **문자 포인터 배열**
  - 여러 개의 문자열을 처리하는 하나의 방법
  - 하나의 문자 포인터가 하나의 문자열을 참조 가능
    - ➢ 문자 포인터 배열은 여러 개의 문자열을 참조 가능

- **문자 포인터 배열의 장/단점**
  - 장점: 문자 포인터 배열 방법은 각각의 문자열을 저장하기 위한 최적의 공간을 사용
  - 단점: 문자 포인터를 사용해서는 문자열 상수의 수정은 불가능
    - ➢ 문장 pa[0][2] = 'v';와 같이 문자열의 수정은 실행 오류 발생

```
char *pa[] = {"JAVA", "C#", "C++"};
printf("%s %s %s\n", pa[0], pa[1], pa[2]);
```




문자열 수정 안됨

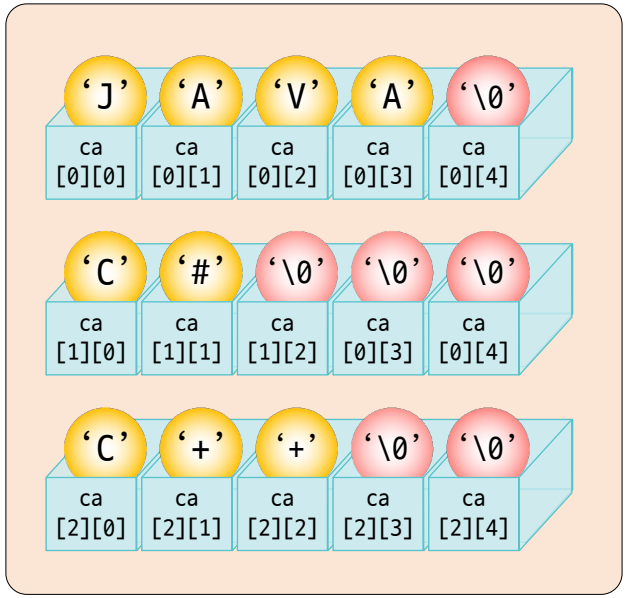
# 여러 문자열 처리: 2차원 문자 배열

- **2차원 문자 배열**
  - 컬럼의 크기: 문자열 중에서 가장 긴 문자열의 길이보다 1 크게 지정
  - 행의 크기: 문자열 개수

- **2차원 문자 배열의 장/단점**
  - 모든 컬럼 수가 동일하게 메모리에 할당: 메모리 낭비
    - ➢ 컬럼의 길이가 서로 다른 경우에는 '\0' 문자가 들어가 낭비
  - 문자열 수정 가능: ca[0][2] = 'v';

[5]: 가장 긴 문자열의 길이 +1

```
char ca[][5] = {"JAVA", "C#", "C++"};
printf("%s %s %s\n", ca[0], ca[1], ca[2]);
```




문자열 수정

# 여러 개의 문자열을 선언과 동시에 저장하고 처리하는 방법

<10strary.c>

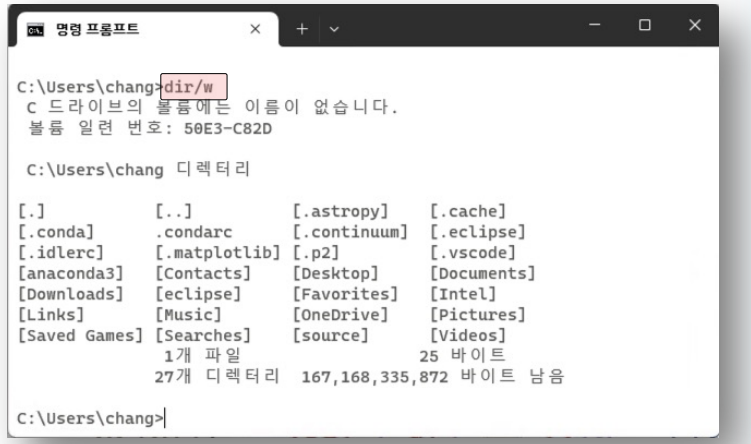
```c
#include <stdio.h>

int main(void)
{
    char *pa[] = { "JAVA", "C#", "C++" };
    char ca[][5] = { "JAVA", "C#", "C++" };

    //pa[0][2] = 'v'; //실행 문제 발생
    //ca[0][2] = 'v'; //수정 가능
    printf("%s %s %s\n", pa[0], pa[1], pa[2]);
    printf("%s %s %s\n", ca[0], ca[1], ca[2]);

    //문자 출력
    printf("%c %c %c\n", pa[0][1], pa[1][1], pa[2][1]);
    printf("%c %c %c\n", ca[0][1], ca[1][1], ca[2][1]);

    ca[0][0] = 'j'; // 수정 가능
    ca[1][0] = 'D';
    ca[2][0] = 'E';
    printf("%s %s %s\n", ca[0], ca[1], ca[2]);

    return 0;
}
```

```
JAVA C# C++
JAVA C# C++
A # +
A # +
jAVA D# E++
```

# 명령행 인자

- int main(int argc, char *argv[])
  - 도스 명령어 dir를 프로그램으로 개발한다면 옵션 “/w”, “/b” 등은 어떻게 인식할까?
    - 명령행 인자(command line arguments)를 사용하는 방법
      - **명령행에서 입력하는 문자열을 프로그램으로 전달하는 방법**

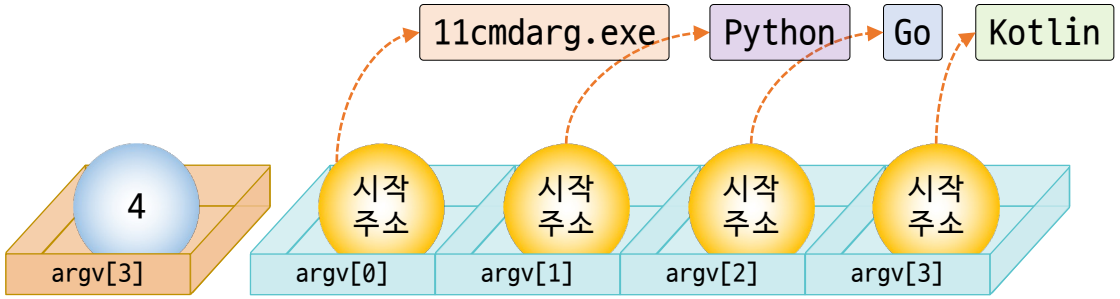
- **프로그램에서 명령행 인자를 받으려면**
  - 두 개의 인자 argc와 argv를 (int argc, char *argv[])로 기술
    - 매개변수 int argc
      - **명령행에서 입력한 문자열의 수**
    - char *argv[] 또는 char **argv
      - **명령행에서 입력한 문자열을 전달받는 문자 포인터 배열**
    - 실행 프로그램 이름도 하나의 명령행 인자에 포함
      - **argv[0]에 실행 프로그램 이름 저장**




# 명령행 인자 출력 (실습)

<11cmdarg.c>

```c
#include <stdio.h>

int main(int argc, char* argv[])
{
    int i = 0;

    printf("실행 명령행 인자(command line arguments) >>\n");
    printf("argc = %d\n", argc);

    for (i = 0; i < argc; i++)
        printf("argv[%d] = %s\n", i, argv[i]);

    return 0;
}
```




Windows 터미널 컴파일 및 실행 (한글 설정)

```
> chcp 65001
> cd C:\workspace_cprog\chap12
> gcc 11cmdarg.c -o 11cmdarg.exe
```

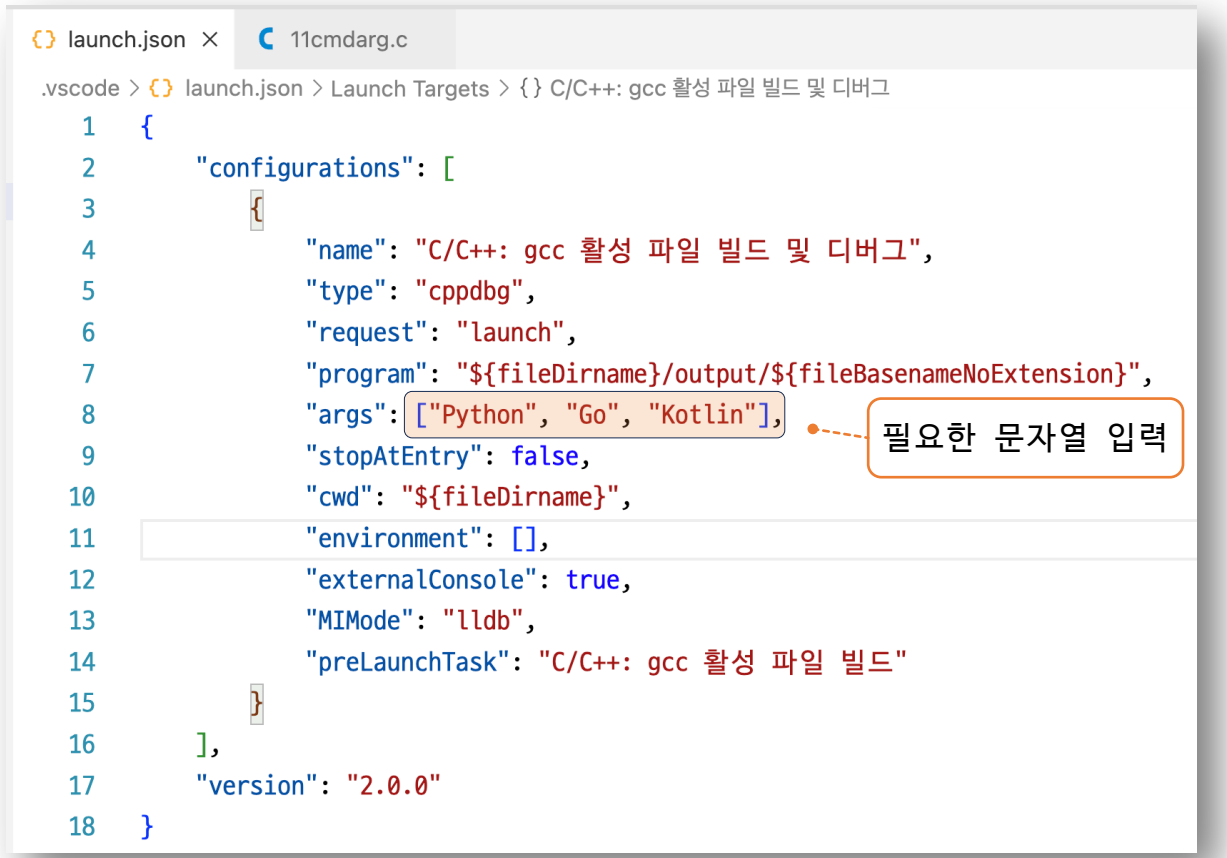
```
> 11cmdarg.exe Python Go Kotlin
실행 명령행 인자(command line arguments) >>
argc = 4
argv[0] = C:\workspace_cprog\chap12\output\11cmdarg.exe
argv[1] = Python
argv[2] = Go
argv[3] = Kotlin
```

Mac 컴파일 및 실행

```
% gcc 11cmdarg.c -o 11cmdarg
% ./11cmdarg Python Go Kotlin
실행 명령행 인자(command line arguments) >>
argc = 4
argv[0] = ./11cmdarg
argv[1] = Python
argv[2] = Go
argv[3] = Kotlin
```

# VS Code에서 명령행 인자 설정

- launch.json 파일에서 “args”:[] 항목에 문자열 입력



# LAB 여러 문자열 처리

- **일차원 문자배열**
  - str1, str2, str3 선언
  - 문자열 “Python”, “Kotlin”, “Tensorflow”를 저장
- **문자 포인터 배열**
  - pstr을 선언
  - 변수 str1, str2, str3 저장
- **pstr을 사용**
  - 저장된 문자열과 문자를 적절히 출력

<lab3strprocess.c>

```c
#include <stdio.h>

int main(void)
{
    char str1[] = "Python";
    char str2[] = "Kotlin";
    char str3[] = "Tensorflow";

    char* pstr[] = { str1, str2, str3 };

    //각각의 3개 문자열 출력
    printf("%s ", pstr[0]);
    printf("%s ", pstr[1]);
    printf("%s\n", pstr[2]);

    //문자 출력
    printf("%c %c %c\n", str1[0], str2[1], str3[2]);
    printf("%c %c %c\n", pstr[0][1], pstr[1][1], pstr[2][1]);

    return 0;
}
```

```
Python Kotlin Tensorflow
P o n
y o e
```

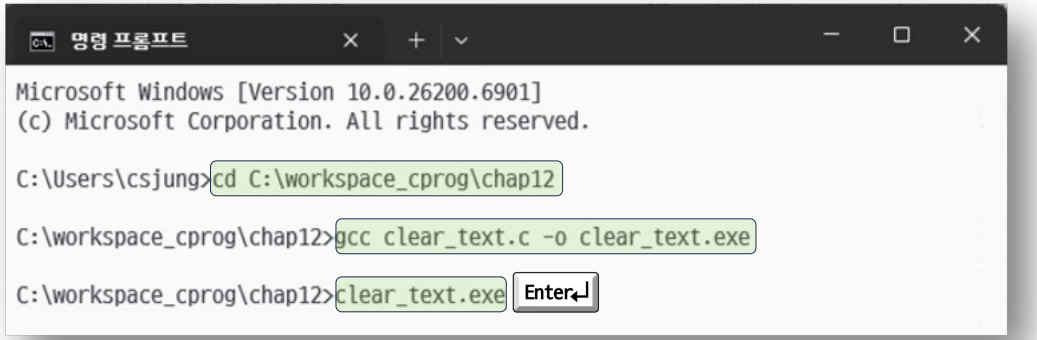
# 화면 텍스트 출력 및 화면 지우기

<clear_text.c>

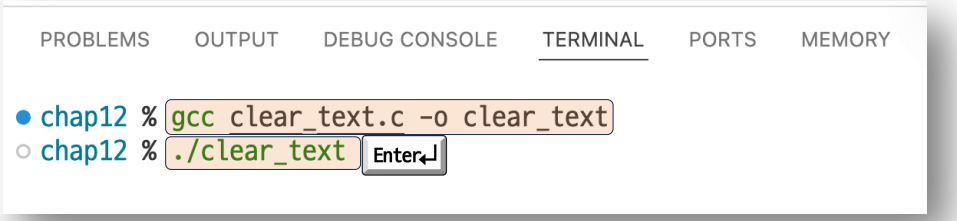
```c
/*
    텍스트 출력 및 화면 지우기
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main()
{
    char message[] = "123456789";
    int len = strlen(message);
    for (int i = 0; i < len; i++)
    {
        system("cls"); // Windows 사용자
        //system("clear"); // Linux/ Mac 사용자

        printf("%c\n", message[i]);
        usleep(1000 * 1000); // 1000 msec delay
    }
    return 0;
}
```

"cls" 화면 지우기 명령어:
운영체제 제공

**Windows 명령 프롬프트에서 실행:**
**- VS Code에서 정상 동작 안됨**




**Mac 사용자: VS Code터미널 사용**



# Questions?