

## <9장> 문자열, 포인터 2

# 학습 목표

- 문자열을 이해하고, 입출력 방법을 알아본다.
- 포인터 배열과 포인터의 포인터 등을 이해한다.
- 표준 C 라이브러리에서 제공하는 문자열 함수와 메모리 관련 함수들을 살펴본다.
- 명령행 인자를 사용한다.

# 목차

- 01 문자열 선언과 초기화
- 02 문자열과 포인터
- 03 문자열 입출력
- 04 문자열 배열
- 05 포인터 배열
- 06 포인터의 포인터
- 07 문자열과 숫자의 변환
- 08 문자열 함수
- 09 메모리 관련 함수
- 10 문자 관련 함수
- 11 명령행 인자

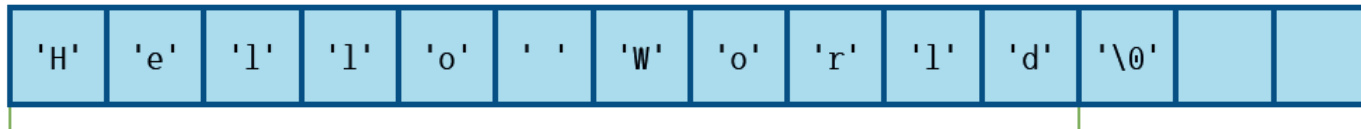
01

# 문자열 선언과 초기화

# 1. 문자열 선언과 초기화

- 문자열(string)은 널 문자('\0')로 종료되는 일련의 문자 배열

"hello"      "a"    "string"    "what a beautiful day!"



아스키(ASCII) 또는 다른 문자 코드

그림 9-1 문자열이 메모리에 저장되는 모습

- 문자열을 선언하는 방법

```
char 변수_이름[] = { 문자1, 문자2, ..., '\0' };  
char 변수_이름[] = 문자열_상수;  
char* 변수_이름 = 문자열_상수;
```

# 1. 문자열 선언과 초기화

## ■ 문자 배열 형태로 선언되는 문자열

- 크기 없이 문자 배열을 선언하고 문자들로 초기화

```
char str1[] = { 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '\0' };
```

- 정해진 크기의 배열을 선언하고 문자열 초기화

```
char str2[14] = { 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '\0' };
```

str1 또는 str3

'H'	'e'	'l'	'l'	'o'	' '	'W'	'o'	'r'	'l'	'd'	'\0'
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------

str2 또는 str4

'H'	'e'	'l'	'l'	'o'	' '	'W'	'o'	'r'	'l'	'd'	'\0'	'\0'	'\0'
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------

그림 9-2 문자 배열이 메모리에 저장되는 모습

# 1. 문자열 선언과 초기화

## ■ 문자 배열 형태로 선언되는 문자열

- 문자 배열을 선언하고 문자열 상수로 초기화

```
char str3[] = "Hello World";  
char str4[14] = "Hello World";
```

- str1과 str2를 선언한 것과 동일한 결과

# 1. 문자열 선언과 초기화

## ■ 포인터 변수로 선언되는 문자열

- char 형 포인터 변수를 선언하고 문자열 상수로 초기화
- 포인터 변수에 문자열 상수의 주소가 저장됨

```
char* pstr = "Hello World";
```

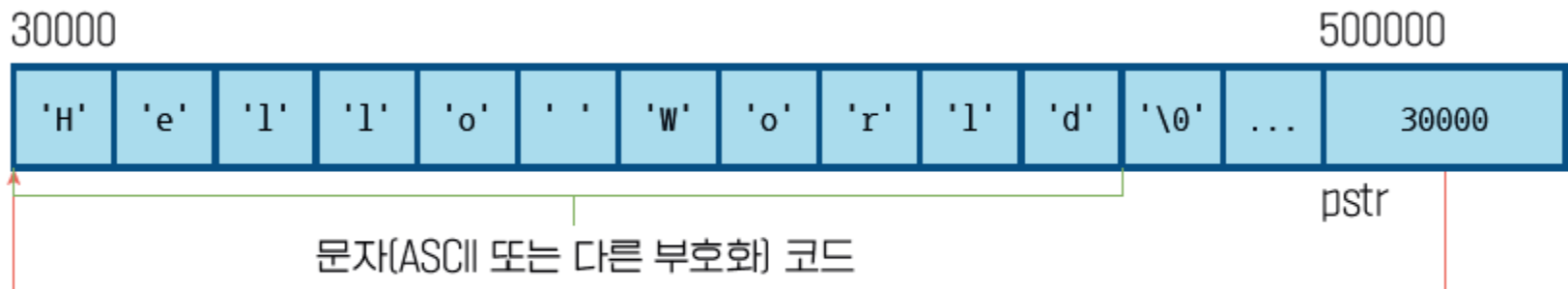


그림 9-3 포인터 변수에 문자열 상수를 초기화할 때 메모리에 저장되는 모습



※ printf( )의 서식 %s를 사용하는 코드

- ➔ [코드 9-1](#)
- ➔ [실행 결과](#)

# 1. 문자열 선언과 초기화

## ■ 문자열 변경

- "Hello"로 초기화한 문자 배열을 수정해 "Help"로 변경

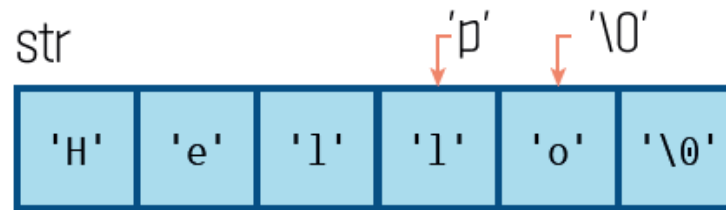


그림 9-4 "Hello"를 "Help"로 변경

# 1. 문자열 선언과 초기화

## ■ 문자열 변경

- "Hello"로 초기화한 문자 배열을 수정해 "Help"로 변경하는 코드

코드 9-2 StrDeclaration2.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main()
5  {
6      char str[] = "hello";
7
8      printf("str = %s\n", str);
9
10     str[3] = 'p'; ← 'l'을 'p'로 변경
11     str[4] = '\0'; ← 문자열을 끝내는 널 문자로 변경
12
13     printf("str = %s\n", str);
14     return 0;
15 }
```

〈실행 결과〉

```
str = Hello
str = Help
```

# 1. 문자열 선언과 초기화

## ■ 문자열 상수는 변경 안됨

- 컴파일 후 실행했을 때 오류가 발생하는 코드

코드 9-3 StrDeclaration3.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main()
5  {
6      char* pstr = "Hello";
7
8      printf("pstr = %s\n", pstr);
9
10     pstr[3] = 'p';
11     pstr[4] = '\0';
12
13     printf("pstr = %s\n", pstr);
14     return 0;
15 }
```

# 1. 문자열 선언과 초기화

## ■ 문자열 변경

- 오류가 발생하는 비주얼 스튜디오 2022 화면

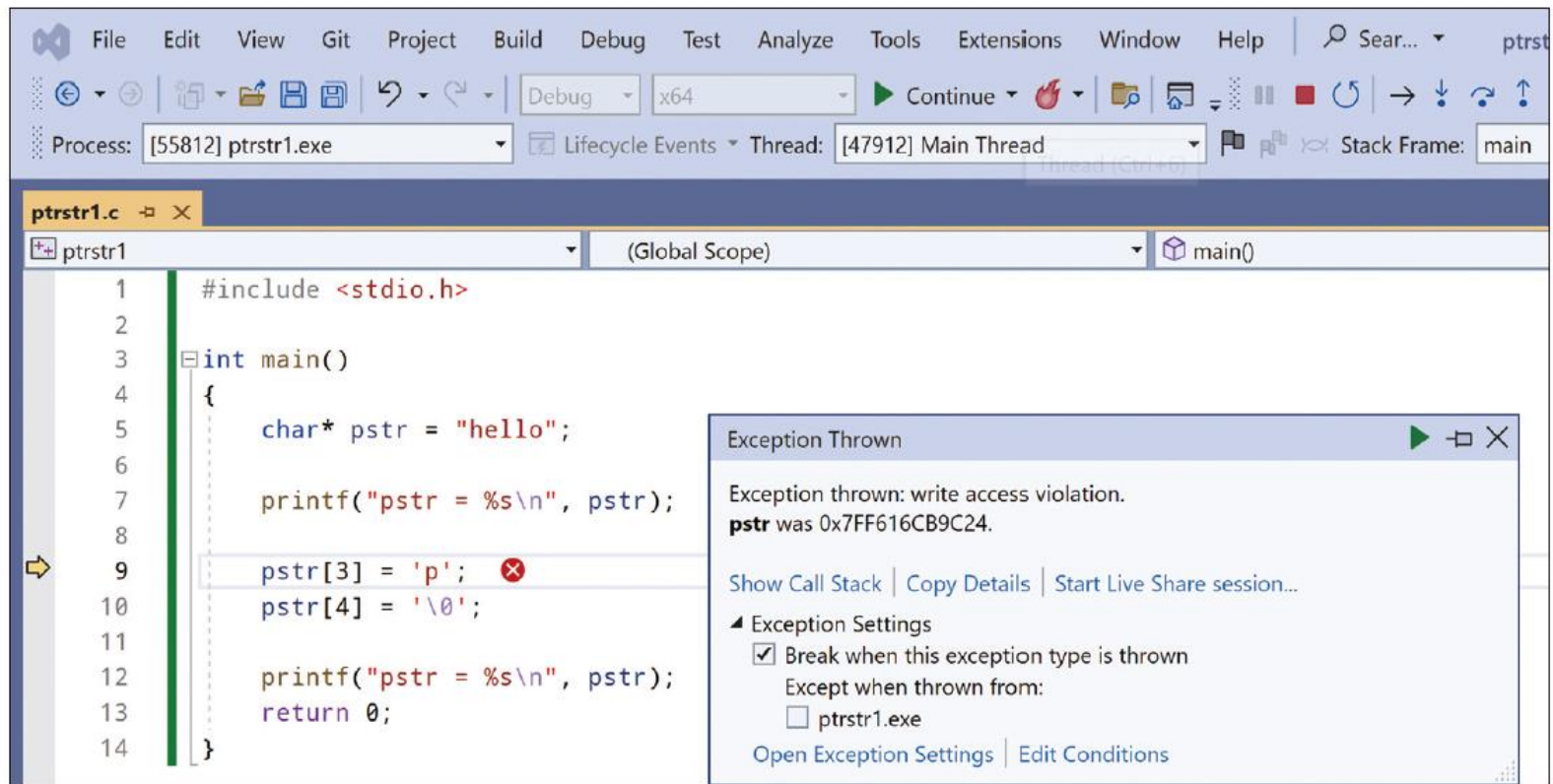


그림 9-5 포인터 문자열을 수정할 때 실행 오류 발생

# 1. 문자열 선언과 초기화

## ■ 문자열과 대입 연산

- 컴파일 오류가 발생하는 코드

```
char str[20] = "hello";  
char st2[20];  
  
str = "hello world";  
str2 = str;
```

- 문자열을 다른 문자 배열에 저장하려면 strcpy() 함수 사용
- strcpy() 함수 원형

```
#include <string.h>  
char* strcpy(char* dest, const char* src);
```

# 1. 문자열 선언과 초기화

## ■ 문자열과 대입 연산

- strcpy() 함수를 사용하는 코드

코드 9-4 StrAssign.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4
5  int main()
6  {
7      char str[] = "hello";
8      char str2[10];
9
10     printf("str = %s\n", str);
11     strcpy(str2, str);
12     printf("str2 = %s\n", str2);
13     return 0;
14 }
```

표준 헤더 파일인 string.h를 포함  
string.h는 문자열 관련 함수들을 선언

원본 문자열을 출력

str2에 str의 문자열을 복사(저장)

str2에 복사된 문자열을 출력

<실행 결과>

str = hello  
str2 = hello

# 1. 문자열 선언과 초기화

## ■ 문자열과 대입 연산

- 포인터 변수를 선언하고 str 저장
  - pstr은 str 배열의 시작 주소만 저장 가능

```
char str[] = "hello";  
char* pstr = str;
```

- strcpy()함수는 문자열 전체를 복사

```
char str2[10];  
strcpy(str2, pstr);
```

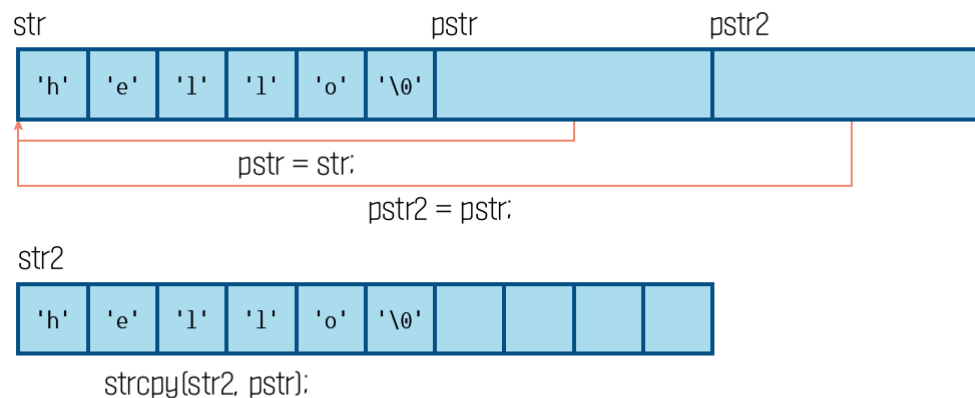


그림 9-6 문자열 대입 연산 동작 방법



# 1. 문자열 선언과 초기화

## ■ 문자열과 메모리 구조

- 배열로 선언되는 문자열은 일반 변수처럼 메모리에 저장됨
- 함수 밖이나 정적 지역변수로 선언한 문자 배열은 전역변수가 저장되는 데이터 영역에 만들어짐
- 반대로 지역변수로 선언하는 문자 배열은 스택에 공간 확보
- 문자열 상수는 프로그램에서 사용하는 상수들을 저장하는 곳에 생성됨
- 일반적으로 상수들은 데이터 영역에 저장됨
- 포인터 변수에 문자열 상수를 저장할 때 포인터 변수는 변수 선언 위치에 따라 데이터나 스택에 저장됨

02

문자열과 포인터

## 2. 문자열과 포인터

### ■ 문자열 출력

- 문자열의 각 문자들을 출력하는 코드

코드 9-5 PrintChars1.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define ARRAY_SIZE(arr, element) (sizeof((arr)) / sizeof((element)))
5  ↗ 배열의 크기 계산
6  int main(void)
7  {
8      char str[] = "Hello";
9      for (int i = 0; i < ARRAY_SIZE(str, str[0]); i++) { ←
10         printf("%c\t", str[i]); // 문자를 한 개씩 탭문자로 분리해서 출력
11     } ←
12     return 0;   ↘ 인덱스를 0부터 배열 크기 - 1까지 1씩 증가시키며 문자 출력
13 }
```

<실행 결과>

H   e   l   l   o

## 2. 문자열과 포인터

### ■ 문자열 출력

- 배열의 크기가 문자열의 길이보다 클 때 발생하는 문제

코드 9-6 PrintChars2.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4
5  #define ARRAY_SIZE(arr, element) (sizeof((arr)) / sizeof((element)))
6
7  int main(void)
8  {
9      char str[8] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h' };
10     strcpy(str, "Hello");
11     for (int i = 0; i < ARRAY_SIZE(str, str[0]); i++) {
12         printf("%c\t", str[i]); // 문자를 한 개씩 탭 문자로 분리해서 출력
13     }
14     return 0;
15 }
```

원하지 않는 문자들이 출력되는 것을 나타내기 위해 배열의 문자들을 초기화

strcpy() 함수는 원본 문자열(source)의 널 문자까지만 배열(destination)에 복사

<실행 결과>

H   e   l   l   o            g   h

## 2. 문자열과 포인터

### ■ 문자열 출력

- 포인터 변수를 ++/-- 연산자와 함께 사용하는 코드

코드 9-7 PrintChars3.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4
5  int main()
6  {
7      char str[8] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h' };
8      strcpy(str, "Hello");
9      for (char* p = str; *p != '\0'; p++) {
10         printf("%c\t", *p); // 문자를 한 개씩 탭문자로 분리해서 출력
11     }
12     return 0;
13 }
```

p를 str[0]의 주소부터 시작해서 1씩 증가시키며 문자를 각각 출력  
\*p가 널 문자일 때까지 반복

#### <실행 결과>

H       e       l       l       o

## 2. 문자열과 포인터

### ■ 문자열 출력

- 문자열 출력 함수의 코드

코드 9-8 PrintChars4.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  void printCharsInStr(const char* pstr) ← 문자열의 시작 주소를 pstr 변수로 전달받음
5  {
6      for (const char* p = pstr; *p != '\0'; p++) { ←
7          printf("%c\t", *p); // 문자를 한 개씩 탭 문자로 분리해서 출력
8      } ←
9  }      ← 포인터 변수 p를 pstr의 주소부터 시작해서 널 문자를 만나기 전까지 한 글자씩 출력
10
11  int main()
12  {
13      char str[] = "Hello";
14      printCharsInStr(str); ← printCharsInStr() 함수에 문자열 str의 시작 주소를 전달
15      return 0;
16  }
```

03

문자열 입출력

### 3. 문자열 입출력

#### ■ 키보드로 문자열 입력받기

- scanf() 함수를 사용하는 코드

코드 9-9 ScanfStr1.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6      char str[20]; ← 문자열을 입력받는데 충분한 공간을 선언
7
8      printf("문자열을 입력하세요: ");
9      scanf("%s", str); ← 문자열 입력. 배열 이름이 주소를 나타내므로 주소 연산자(&)를 사용하지 않음
10     printf("str = %s\n", str);
11     return 0;
12 }
```

##### 〈실행 결과〉

문자열을 입력하세요: hello  
str = hello

##### 〈실행 결과〉

문자열을 입력하세요: hello world  
str = hello



### 3. 문자열 입출력

#### ■ 키보드로 문자열 입력받기

- scanf() 함수를 두 번 실행하는 코드

코드 9-10    ScanfStr2.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6      char str[20];
7
8      printf("문자열을 입력하세요: ");
9      scanf("%s", str);
10     printf("first str = %s\n", str);
11     scanf("%s", str);
12     printf("second str = %s\n", str);
13     return 0;
14 }
```

#### 〈실행 결과〉

문자열을 입력하세요: hello world  
first str = hello  
second str = world

### 3. 문자열 입출력

#### ■ 키보드로 문자열 입력받기

- 다른 변수에 저장하는 코드

코드 9-11 Scanf1.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6      char subject[20];
7      int midterm;
8      int final;
9      printf("과목_이름,중간고사 성적,기말고사 성적순으로 입력하세요: ");
10     int converted = scanf("%s,%d,%d", subject, &midterm, &final);
11     printf("subject = %s, midterm = %d, final = %d, converted = %d\n",
subject, midterm, final, converted);
12     return 0;
13 }
```

입력 서식이 문자열, 정수, 정수 형태로 되어 있어 %s,%d,%d로 입력받음

과목\_이름,중간고사 성적,기말고사 성적순으로 입력하세요: CProgramming,90,85

subject = CProgramming,90,85, midterm = 32758, final = 1087051017, converted = 1

### 3. 문자열 입출력

#### ■ 키보드로 문자열 입력받기

- `scanf()` 사이에 공백을 넣었을 때

```
과목_이름,중간고사 성적,기말고사 성적순으로 입력하세요: CProgramming, 90, 85  
subject = CProgramming,, midterm = 32759, final = -545516279, converted = 1
```

- 공백을 `scanf()` 앞에 넣었을 때

```
과목_이름,중간고사 성적,기말고사 성적순으로 입력하세요: CProgramming , 90 , 85  
subject = CProgramming, midterm = 32758, final = -1260186359, converted = 1
```

- `scanf()` 전까지 나오는 모든 글자를 문자열로 읽는 코드
  - `[^,]`는 ,가 아닌 모든 글자를 읽도록 처리함(^는 not의 의미로 해석)

```
scanf("%[^,],%d,%d", subject, &midterm, &final);
```

### 3. 문자열 입출력

#### ■ 키보드로 문자열 입력받기

- 코드 다시 작성하고 두 번 실행하기

코드 9-12 Scanf2.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6      char subject[20];
7      int midterm;
8      int final;
9      printf("과목_이름,중간고사 성적,기말고사 성적순으로 입력하세요: ");
10     int converted = scanf("%[^,],%d,%d", subject, &midterm, &final);
11     printf("subject = %s, midterm = %d, final = %d, converted = %d\n",
subject, midterm, final, converted);
12     return 0;
13 }
```

##### <실행 결과>

과목\_이름,중간고사 성적,기말고사 성적순으로 입력하세요: CProgramming,90,85  
subject = CProgramming, midterm = 90, final = 85, converted = 3

##### <실행 결과>

과목\_이름,중간고사 성적,기말고사 성적순으로 입력하세요: C Programming , 90, 85  
subject = C Programming , midterm = 90, final = 85, converted = 3

첫 번째 쉼표를 만나면 문자열을 subject에 저장하는 것을 멈춤

### 3. 문자열 입출력

#### ■ 키보드로 문자열 입력받기

- 줄바꿈 문자가 입력 되기 전까지 문자열로 입력받는 코드

```
scanf("%[^\n]", subject);
```

- gets( )과 fgets( ) 함수 사용하기

```
char* gets(char* s);  
char* fgets(char* s, int n, FILE* stream)
```

### 3. 문자열 입출력

#### ■ 키보드로 문자열 입력받기

##### ■ gets() 함수 코드

```
char s[20];  
gets(s);  
printf("s = %s\n", s);
```

- 20자 이상이어도 배열 크기를 넘치는지 확인하지 않음
- 오류를 발생시킬 수 있고, 보안 문제가 있어 사용하는 것을 권하지 않음 (fgets() 함수 사용 권장)

### 3. 문자열 입출력

#### ■ 키보드로 문자열 입력받기

- `fgets()` 함수가 함수 실행을 종료하고 반환하는 경우

- 사용자가 줄바꿈 문자를 입력(엔터키)한다.
- 파일에서 문자열을 입력받는 중이었다면 파일에 더 이상 읽을 데이터가 없다(End Of File, EOF).
- 입력 버퍼에  $n$ 개 이상의 문자가 있을 때 배열에  $n - 1$ 개의 문자를 채운다. 입력 버퍼에 있는 나머지 문자들은 그대로 버퍼에 있다.

### 3. 문자열 입출력

#### ■ 키보드로 문자열 입력받기

##### ■ fgets() 함수 코드

코드 9-13 fgets1.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6      char s[20];
7      if (fgets(s, 20, stdin) != NULL) {
8          printf("s = %s\n", s);
9      }
10     return 0;
11 }
```

최대 19글자까지만 문자를 위해 한 개의 요소를 남김  
표준 입력 장치에서 문자열 입력받기

##### <실행 결과>

```
hello world
s = hello world
```

##### <실행 결과>

```
C is a general programming language
s = C is a general prog
```



### 3. 문자열 입출력

#### ■ 문자열에서 입력받기

- 기말고사 성적들이 문자열 형태로 있다고 가정

```
"CProgramming1 90 86"
```

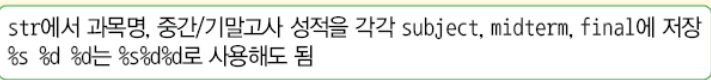
- sscanf()는 scanf()가 키보드로 입력받는 것처럼 문자열에서 서식에 맞춰 값을 입력받음
- sprintf()는 printf()가 서식에 맞춰 화면에 값을 출력하는 것과 비슷하게 문자열에 출력(저장)

### 3. 문자열 입출력

#### ■ 문자열에서 입력받기

- 한 과목의 성적과 점수를 읽는 코드

코드 9-14 Sscanf1.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6      char str[] = "CProgramming1 90 86";
7      char subject[20];
8      int midterm;
9      int final;
10     
11     int converted = sscanf(str, "%s %d %d", subject, &midterm, &final);
12     printf("subject = %s midterm = %d final = %d, converted = %d\n",
13           subject, midterm, final, converted);
14     return 0;
15 }
```

#### <실행 결과>

subject = CProgramming1 midterm = 90 final = 86, converted = 3

### 3. 문자열 입출력

#### ■ 서식에 맞춰 문자열에 출력

- sprintf() 함수로 값을 저장한 후 생성된 문자열을 sscanf() 함수에 전달해서 다시 값을 읽는 코드

코드 9-16 Sprintf1.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6      char str[50];
7      sprintf(str, "%s, %d, %d", "C programming 1", 90, 86);
8      printf("str = %s\n", str);
9
10     char subject[20];
11     int midterm;
12     int final;
13
14     int converted = sscanf(str, "%[^,],%d,%d", subject, &midterm, &final);
15     printf("subject = %s, midterm = %d, final = %d, converted = %d\n",
subject, midterm, final, converted);
16     return 0;
17 }
```

str 배열에 주어진 값을 서식에 맞춰 저장  
문자열을 종료하는 널 문자도 삽입함

str을 확인하기 위해 출력

#### <실행 결과>

str = C Programming 1, 90, 86

subject = C Programming 1, midterm = 90, final = 86, converted = 3

### 3. 문자열 입출력

#### ■ 서식에 맞춰 문자열에 출력

- snprintf() 함수를 사용할 수 있게 하는 코드
  - snprintf()는 배열의 크기 안에서 문자열을 저장(문자 배열의 범위를 넘치지 않도록 함)

코드 9-17 Snprintf1.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6      char str[50];
7      snprintf(str, sizeof(str), "%s, %d, %d", "C Programming 1", 90, 86);
8      printf("str = %s\n", str);
9
10     char subject[20];
11     int midterm;
12     int final;
13     int converted = sscanf(str, "%[^,],%d,%d", subject, &midterm, &final);
14     printf("subject = %s midterm = %d final = %d, converted = %d\n",
subject, midterm, final, converted);
15     return 0;
16 }
```

#### <실행 결과>

str = C Programming 1, 90, 86

subject = C Programming 1 midterm = 90 final = 86, converted = 3

※ 배열의 크기가 문자열 길이보다 작은 경우

➔ [코드 9-18](#)

➔ [실행 결과](#)

04

문자열 배열

## 4. 문자열 배열

### ■ 2차원 문자 배열을 이용해서 문자열 배열 구현

- 10글자를 저장할 수 있는 문자 배열을 선언하고 문자열을 초기화

```
char animal[10] = "Dog";
```

- 2차원 문자열 배열 만들기

코드 9-19

```
char animals[][10] = { "Dog", "Cat", "Racoon", "Duck", "Iguana" };
```

## 4. 문자열 배열

### ■ 2차원 문자 배열을 이용해서 문자열 배열 구현

- char[10] 배열이 n개 있는 형태의 2차원 배열

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
animals	'D'	'o'	'g'	'\0'						
	'C'	'a'	't'	'\0'						
	'R'	'a'	'c'	'o'	'o'	'n'	'\0'			
	'D'	'u'	'c'	'k'	'\0'					
	'I'	'g'	'u'	'a'	'n'	'a'	'\0'			

그림 9-8 2차원 배열로 만들어지는 문자열 배열

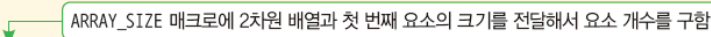


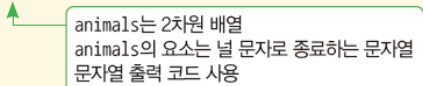
## 4. 문자열 배열

### ■ 2차원 문자 배열을 이용해서 문자열 배열 구현

- animals를 출력하는 프로그램

코드 9-20 Char2DArray.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define ARRAY_SIZE(arr, element) (sizeof((arr)) / sizeof((element)))
5
6  int main()
7  {
8      char animals[][10] = { "Dog", "Cat", "Racoon", "Duck", "Iguana" };
9       ARRAY_SIZE 매크로에 2차원 배열과 첫 번째 요소의 크기를 전달해서 요소 개수를 구함
10     for (int i = 0; i < ARRAY_SIZE(animals, animals[0]); i++) {
11         printf("animals[%d] = %s\n", i, animals[i]);
12     }
13     return 0;
14 }
```

 animals는 2차원 배열  
animals의 요소는 널 문자로 종료하는 문자열  
문자열 출력 코드 사용

#### <실행 결과>

```
animals[0] = Dog
animals[1] = Cat
animals[2] = Racoon
animals[3] = Duck
animals[4] = Iguana
```

## 4. 문자열 배열

### ■ 2차원 문자 배열을 이용해서 문자열 배열 구현

- animals 배열을 함수의 매개변수로 전달받아 출력하는 프로그램

코드 9-21 Char2DArray2.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define ARRAY_SIZE(arr, element) (sizeof((arr)) / sizeof((element)))
5
6  void printStrArray(char strs[][10], int size)
7  {
8      for (int i = 0; i < size; i++) {
9          printf("strs[%d] = %s\n", i, strs[i]);
10     }
11 }
12
13 int main()
14 {
15     char animals[][10] = { "Dog", "Cat", "Racoon", "Duck", "Iguana" };
16
17     printStrArray(animals, ARRAY_SIZE(animals, animals[0]));
18     return 0;
19 }
```

animals[][10]을 전달받는 매개변수를 animals와 동일한 형태로 선언

## 4. 문자열 배열

### ■ 2차원 문자 배열을 이용해서 문자열 배열 구현

- const 붙이기

코드 9-22 Char2DArray3.c

```
1 void printStrArray(const char strs[][10], int size)
2 {
3     // strs[0][0] = 'a'; 컴파일 오류 발생
4     for (int i = 0; i < size; i++) {
5         printf("strs[%d] = %s\n", i, strs[i]);
6     }
7 }
```

2차원 배열 전체를 const로 수식하기 때문에  
strs[0][0] = 'a'; 코드는 컴파일 오류 발생

## 4. 문자열 배열

### ■ 포인터를 이용해서 문자열 배열 구현

- char 형 포인터로 문자열 초기화

```
char* animal = "Dog";
```

- char\* 형 배열을 생성하고 문자열 초기화

코드 9-23

```
char* animals[5] = { "Dog", "Cat", "Racoon", "Duck", "Iguana" };
```

## 4. 문자열 배열

### ■ 포인터를 이용해서 문자열 배열 구현

- animals를 선언하면서 배열 요소들을 초기화하므로 배열 개수는 생략 가능

```
char* animals[] = { "Dog", "Cat", "Racoon", "Duck", "Iguana" }; // 5는 생략 가능
```

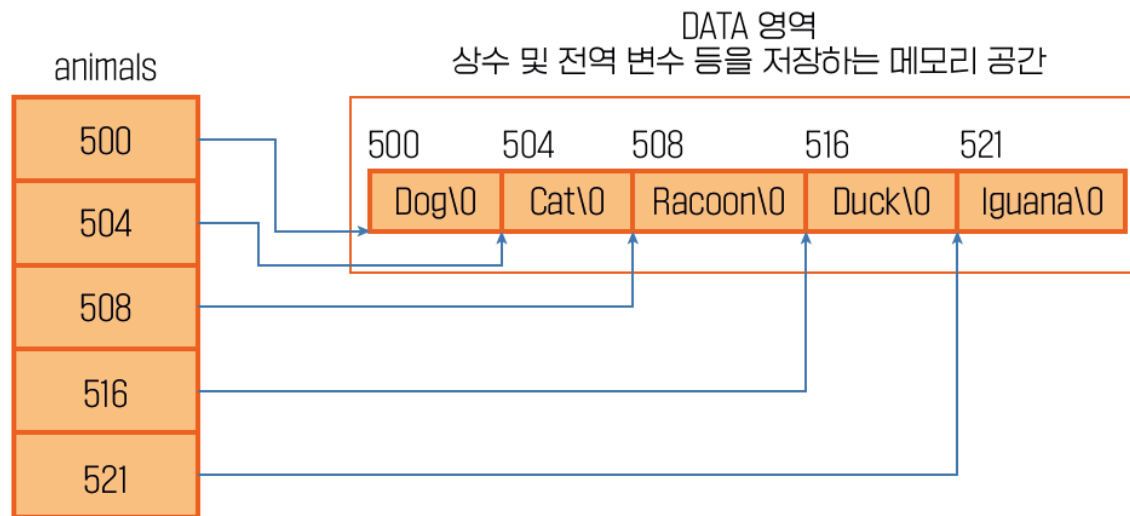


그림 9-9 문자열의 위치를 저장하는 animals 배열의 요소

## 4. 문자열 배열

### ■ 포인터를 이용해서 문자열 배열 구현

- animals의 요소들을 출력하는 코드

코드 9-24 PtrArray.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define ARRAY_SIZE(arr, element) (sizeof((arr)) / sizeof((element)))
5
6  int main()
7  {
8      char* animals[] = { "Dog", "Cat", "Racoon", "Duck", "Iguana" };
9
10     for (int i = 0; i < ARRAY_SIZE(animals, animals[0]); i++) {
11         printf("animals[%d] = %s\n", i, animals[i]);
12     }
13     return 0;
14 }
```

#### <실행 결과>

```
animals[0] = Dog
animals[1] = Cat
animals[2] = Racoon
animals[3] = Duck
animals[4] = Iguana
```

## 4. 문자열 배열

### ■ 포인터를 이용해서 문자열 배열 구현

- 포인터로 된 문자열 배열 함수로 전달하기

코드 9-25 PtrArray2.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  #define ARRAY_SIZE(arr, element) (sizeof((arr)) / sizeof((element)))
5
6  void printStrArray(char* strs[], int size)
7  {
8      for (int i = 0; i < size; i++) {
9          printf("strs[%d] = %s\n", i, strs[i]);
10     }
11 }
12
13 int main()
14 {
15     char* animals[] = { "Dog", "Cat", "Racoon", "Duck", "Iguana" };
16
17     printStrArray(animals, ARRAY_SIZE(animals, animals[0]));
18     return 0;
19 }
```

#### <실행 결과>

```
animals[0] = Dog
animals[1] = Cat
animals[2] = Racoon
animals[3] = Duck
animals[4] = Iguana
```

## 4. 문자열 배열

### ■ 포인터를 이용해서 문자열 배열 구현

- 함수 원형

```
void printStrArray(char* strs[], int size);  
void printStrArray(char** strs, int size);
```

- const 붙이기

코드 9-26 PtrArray3.c

```
1 //void printStrArray(char* const * strs, int size)  
2 void printStrArray(char* const strs[], int size)  
3 {  
4 //   strs[0] = "Rabbit"; ← 주석을 해제하면 컴파일 오류 발생  
5     for (int i = 0; i < size; i++) {  
6         printf("strs[%d] = %s\n", i, strs[i]);  
7     }  
8 }
```



05

포인터 배열

## 5. 포인터 배열

- 정수형 포인터 배열

코드 9-29

```
1  int a = 1;
2  int b = 2;
3  int c = 3;
4  int* pInts[] = { &a, &b, &c };
```

- 포인터 배열 목적

- 문자열 여러 개를 배열로 관리
- 여러 개 함수 포인터 관리
- 동적 할당받은 여러 개 메모리 공간을 관리(11장)

## 5. 포인터 배열

### ■ 함수 포인터 배열

- 인터럽트(interrupt)
  - 보통 번호가 붙으며 다양한 종류가 있음
  - 예를 들면, 키보드 입력, 마우스 입력같은 작업들이 인터럽트로 처리
  - 숫자를 0으로 나누었을 때 인터럽트가 발생
- 인터럽트 핸들러(interrupt handler)
  - 인터럽트 핸들러 또는 인터럽트 서비스 루틴(interrupt service routine)
  - 특정 인터럽트가 걸렸을 때 실행되는 코드
- 인터럽트 벡터(interrupt vector)
  - 인터럽트 핸들러의 코드 위치를 저장한 1차원 배열

## 5. 포인터 배열

### ■ 함수 포인터 배열

- 인터럽트 핸들러 함수 포인터의 자료형

```
typedef void (*InterruptHandler)(void*);
```

- InterruptHandler() 함수

#### 코드 9-30

```
1 void InterruptHandler0(void* p)
2 {
3     int* intPtr = (int*) p;
4
5     printf("Servicing Interrupt %d\n", *intPtr);
6     printf("Servicing Interrupt %d\n", *((int*) p)); // 짧은 버전
7 }
```

## 5. 포인터 배열

### ■ 함수 포인터 배열

- 어떤 번호의 인터럽트를 처리하는지 출력하는 함수

코드 9-31

```
1 void InterruptHandler1(void* p)
2 {
3     double* dblPtr = (double*) p;
4
5     printf("Handling Interrupt: value = %f\n", *dblPtr);
6     printf("Handling Interrupt: value = %f\n", *((double*) p)); // 짧은 버전
7 }
8
9 void InterruptHandler2(void* p)
10 {
11     char* pStr = (char*) p;
12
13     printf("Servicing Interrupt with \"%s\"\n", pStr);
14     printf("Servicing Interrupt with \"%s\"\n", ((char*) p)); // 짧은 버전
15 }
```

## 5. 포인터 배열

### ■ 함수 포인터 배열

- InterruptHandler() 함수의 배열 선언

```
InterruptHandler handlers[] = { InterruptHandler0, InterruptHandler1,  
InterruptHandler2 };
```

- 0~2 사이의 정수 한 개를 입력받고 해당 번호로 등록된 인터럽트 핸들러를 호출하는 함수

코드 9-32

```
1 void callInterruptHandler(int interruptNo)  
2 {  
3     if (interruptNo == 0) {  
4         handlers[interruptNo](&interruptNo);  
5     }  
6     else if (interruptNo == 1) {  
7         double pi = 3.1415;  
8         handlers[interruptNo](&pi);  
9     }  
10    else if (interruptNo == 2) {  
11        handlers[interruptNo]("Interrupt 2 Handler");  
12    }  
13 }
```

인터럽트 번호를 메모리 주소인 것처럼 변환해서 인터럽트 핸들러에 전달  
인터럽트 핸들러가 실행되고 종료할 때까지 callInterruptHandler 변수는 사라지지 않음  
따라서 callInterruptHandler()의 매개변수의 주소를 전달할 수 있음

실숫값을 변수에 저장한 후 전달

인터럽트 핸들러를 호출하면서 문자열 전달  
void\*는 어떤 종류의 메모리 주소도 모두 받을 수 있어 형 변환 필요 없음

※ 사용자로부터 인터럽트 번호를 입력받고  
함수를 실행시키는 전체 프로그램

➔ [코드 9-33](#)

➔ [실행 결과](#)

06

포인터의 포인터



## 6. 포인터의 포인터

### ■ 포인터의 포인터

- 포인터의 포인터는 포인터 변수를 가리키는 포인터
- '이중 포인터(double pointer)'라고도 함
- 포인터의 포인터는 변수를 선언할 때 '\*' 기호를 두 개 붙임
- 포인터를 사용하는 문자 배열처럼 typedef를 사용하면 일반 포인터처럼 단순화도 가능

```
int** pptr;
```

```
typedef int* IPTR;
```

```
IPTR* pptr2; // pptr과 동일한 자료형으로 선언
```

## 6. 포인터의 포인터

### ■ 함수에서 포인터의 포인터 사용

- 두 함수의 strs는 동일한 매개변수

```
void printStrArray(char* strs[], int size);  
void printStrArray(char** strs, int size);
```

- 문자열에서 새로운 줄의 시작을 찾는 프로그램 구현
- 새로운 줄의 시작 주소를 반환하는 함수 원형

```
int findNewLineChar(char* pstr, char** newLinePos);
```

- findNewLineChar()에 전달할 문자열 선언

```
char str[] = "C Programming 1, 90, 86\nMath, 85, 80\nSW English, 90, 90";
```

- 새로운 줄의 시작 주소를 newLinePos 포인터 변수로 반환

## 6. 포인터의 포인터

### ■ 함수에서 포인터의 포인터 사용

- findNewLineChar() 함수 구현 - 알고리즘

\*pstr이 줄바꿈 문자가 아니고 널 문자도 아니면 계속 반복  
    pstr을 1 증가  
\*pstr이 줄바꿈 문자면 \*newLinePos를 pstr로 저장 후 1 반환  
\*pstr이 널 문자면 \*newLinePos를 NULL로 저장 후 0 반환

## 6. 포인터의 포인터

### ■ 함수에서 포인터의 포인터 사용

- findNewLineChar() 함수 코드

코드 9-35

```
1  int findNewLineChar(char* pstr, char** newLinePos)
2  {
3      while (*pstr != '\n' && *pstr != '\0') { pstr++; }
4      if (*pstr == '\n') {
5          *newLinePos = pstr;
6          return 1;
7      }
8      else if (*pstr == '\0') {
9          *newLinePos = NULL;
10         return 0;
11     }
12     return 0;
13 }
```

psr을 1씩 증가시키며, 문자열에서 줄바꿈 문자 또는 널 문자를 찾을 때까지 반복

줄바꿈 문자를 찾았으면, \*newLinePos의 값을 변경하고 1 반환

줄바꿈 문자 없이 문자열이 종료된다면, newLinePos를 NULL로 지정하고 0 반환

※ 함수를 사용하는 코드

➔ [코드 9-36](#)

➔ [실행 결과](#)

07

# 문자열과 숫자의 변환

## 7. 문자열과 숫자의 변환

- 숫자를 문자열로 변환하는 것은 sprintf()나 snprintf()를 사용
- 문자열을 숫자로 변환하는 것은 sscanf() 함수를 사용
- C 표준 라이브러리는 이것 외에 문자열을 숫자로 변환하는 여러 가지 함수들을 제공

### ■ ANSI C의 표준 변환 함수

```
double strtod(const char* str, char** ptr);  
long strtol(const char* str, char** ptr, int base);  
unsigned long strtoul(const char* str, char** ptr, int base);
```

## 7. 문자열과 숫자의 변환

- str

- 변환할 숫자를 저장한 문자열

- ptr

- 변환 과정에서 오류가 없는지 확인하는 데 필요
- 변환된 값이 자료형의 범위가 넘어 오버플로(overflow) 또는 언더플로(underflow)가 발생하면 정해진 값들이 반환

- errno

- C 표준 라이브러리에서 오류를 표기하기 위해 사용하는 전역변수



※ 변환 함수를 사용하는 프로그램

➔ [코드 9-37](#)

➔ [실행 결과](#)

08

문자열 함수

## 8. 문자열 함수

### ■ 문자열 복사

#### ■ strncpy() 함수

```
char* strncpy(char* dest, const char* src, size_t n);
```

#### • 동작 방법

표 9-1 strncpy() 함수 동작 방법

조건	동작 방법
$src\_len < n$	src를 dest에 src_len개만큼 복사하고 dest[src_len]부터 $n - src\_len$ 개만큼 널 문자로 dest을 채운다.
$src\_len \geq n$	src에서 dest에 n개의 문자를 복사한다. 널 문자는 dest에 복사되지 않는다. 문자열이 종료되지 않는다.
$n \leq 0$	아무것도 하지 않는다.

※ 문자열 복사 코드

➔ [코드 9-38](#)

➔ [실행 결과](#)

## 8. 문자열 함수

### ■ 문자열 길이 확인

#### ■ str

- str의 길이를 바이트 단위로 반환
- 빈 문자열(널 문자가 가장 첫 번째 문자인 문자열)은 0을 반환

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

※ 문자열 길이 확인 코드

➔ [코드 9-39](#)  
➔ [실행 결과](#)

## 8. 문자열 함수

### ■ 문자열 연결

#### ■ strcat() 함수

- dest 문자열 뒤에 src를 덧붙이고 dest를 반환
- src가 빈 문자열이라면 strcpy()와 동일한 효과

```
char* strcat(char* dest, const char* src);  
char* strncat(char* dest, const char* src, size_t n);
```

#### • 동작 방법

표 9-2 strncat() 함수 동작 방법

조건	동작 방법
src_len < n	src를 dest 뒤에 연결하고 n 문자를 붙여 종료시킨다.
src_len >= n	src에서 dest 뒤에 n개의 문자를 연결해서 붙인 다음 n 문자를 붙여 종료시킨다. n+1개 문자가 dest 뒤에 연결되는 것이다.
n <= 0	아무것도 하지 않는다.

※ 문자열 연결 코드

➔ [코드 9-40](#)

➔ [실행 결과](#)

## 8. 문자열 함수

### ■ 문자열 비교

#### ■ strcmp()와 strncmp() 함수

- 두 함수는 문자열 s1과 s2를 사전식으로 비교
- src가 빈 문자열이라면 strcpy()와 동일한 효과

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

- s1이 s2보다 작다는 것은 두 가지로 구분

- s1과 s2 문자열의 문자들을 비교하면서 처음으로 달라지는 문자를 s1\_ch와 s2\_ch라고 가정했을 때, s1\_ch가 s2\_ch보다 사전에서 먼저 나타난다면 s1이 s2보다 작다. 단, 영문 소문자는 대문자보다 크다.
  - 예를 들어, "abcd"와 "abce"를 비교하면 "abcd"가 작음
  - "Abcd"는 "abcd"보다 작음
- s1의 길이가 s2보다 짧지만 s1의 내용이 s2의 앞부분과 동일하다면 s1이 s2보다 작다.
  - 예를 들어, "abcd"와 "abcde"를 비교하면 "abcd"가 작음



※ 문자열 비교 코드

➔ [코드 9-41](#)

➔ [실행 결과](#)

## 8. 문자열 함수

### ■ 문자열 잘라내기

#### ■ strtok() 함수

- 함수 원형

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

- 구분자

```
char str[] = "C is a general programming language";  
char* token = strtok(str, " \\t\\n");
```

- 토큰이 NULL이 아니라고 가정할 때

```
token = strtok(NULL, " \\t\\n");
```

※ 문자열 잘라내기 코드

➔ [코드 9-42](#)

➔ [실행 결과](#)

## 8. 문자열 함수

### ■ 기타 문자열 함수

#### ■ 표준 라이브러리에서 제공하는 다른 문자열 관련 함수

표 9-3 C 표준 라이브러리에서 제공하는 문자열 관련 함수들

함수 원형	설명
<code>char* strchr(const char* s, int c);</code>	문자열 <code>s</code> 에서 문자 <code>c</code> 를 찾는다. 있으면 첫 번째로 'c'가 나타난 위치를 메모리 주소로 반환하고, 없으면 <code>NULL</code> 을 반환한다.
<code>char* strrchr(const char* s, int c);</code>	문자열 <code>s</code> 에서 문자 <code>c</code> 를 찾는다. 있으면 가장 마지막으로 'c'가 나타난 위치를 메모리 주소로 반환하고, 없으면 <code>NULL</code> 을 반환한다. 이 함수는 끝에서 처음으로 향하는 방향으로 검색한다.
<code>size_t strspn(const char *s, const char* set);</code>	문자열 <code>s</code> 를 검색해서 문자열 <code>set</code> 의 글자들이 없는 첫 번째 인덱스를 반환한다. 예를 들어, <code>s</code> 가 "dabcd"고 <code>set</code> 이 "ad"면, 'a' 또는 'd'가 아닌 문자의 첫 번째 인덱스인 2가 반환된다.
<code>size_t strcspn(const char *s, const char* set);</code>	<code>strspan()</code> 과 비슷하게 <code>set</code> 에 포함된 문자의 첫 번째 인덱스를 반환한다. 예를 들어, <code>s</code> 가 "abdc"고 <code>set</code> 이 "dc"라면, 첫 번째 'c'의 인덱스에 해당하는 2를 반환한다.
<code>char* strpbrk(const char *s, const char* set);</code>	<code>set</code> 에 포함된 문자의 첫 번째 포인터(메모리 주소)를 반환한다. 예를 들어, <code>s</code> 가 "abdc"고 <code>set</code> 이 "dc"라면, 첫 번째 'c'의 주소인 <code>&amp;str[2]</code> 를 반환한다.
<code>char* strstr(const char* src, const char* sub);</code>	문자열 <code>src</code> 에서 문자열 <code>sub</code> 를 찾는다. 있으면 <code>sub</code> 가 처음으로 나타나는 메모리 주소를 반환하고, 없으면 <code>NULL</code> 을 반환한다.

## 8. 문자열 함수

### ■ 간단한 scanf() 함수 구현

- getSingleInput() 함수
  - 원형

```
void getSingleInput(const char* format, void* p);
```

※ scanf() 함수 코드

➔ [코드 9-43](#)

09

# 메모리 관련 함수

## 9. 메모리 관련 함수

### ■ 일정한 값으로 메모리 공간 채우기

#### ■ memset() 함수

```
void* memset(void* ptr, int val, size_t len);
```

※ memset() 함수 코드

➔ [코드 9-44](#)

## 9. 메모리 관련 함수

### ■ 다른 메모리 공간과 메모리 공간의 비교

#### ■ memcmp() 함수

```
int memcmp(const void* ptr1, const void* ptr2, size_t len);
```

※ memcmp() 함수 코드

➔ [코드 9-45](#)

➔ [실행 결과](#)



## 9. 메모리 관련 함수

### ■ 다른 메모리 공간에 메모리 공간 복사

- memcpy()와 memmove() 함수
  - 정수 배열 100개를 arr로 선언
  - 초기화한 다음 다른 배열 arr2에 memcpy( ) 함수를 이용해서 복사

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



그림 9-11 memcpy() 함수로 정수 배열 복사

※ memcpy() 함수 코드

➔ [코드 9-46](#)

➔ [실행 결과](#)

## 9. 메모리 관련 함수

### ■ 다른 메모리 공간에 메모리 공간 복사

- memcpy()와 memmove() 함수
  - 배열을 선언하고 1~100으로 초기화하고 합을 구해서 출력

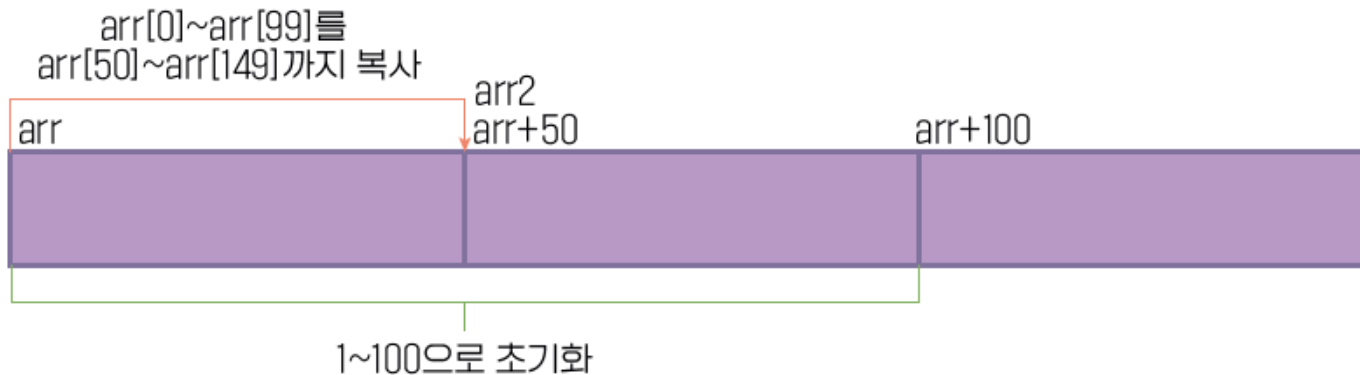


그림 9-12 배열을 선언하고 포인터를 이용해서 중첩되는 또 다른 배열 구성

## 9. 메모리 관련 함수

### ■ 다른 메모리 공간에 메모리 공간 복사

- memcpy()와 memmove() 함수
  - 정수 배열 150개를 선언하고, arr[0]~arr[99]까지 1~100으로 초기화하는 코드

코드 9-47

```
1  for (int i = 0; i < 100; i++) {  
2      arr[i + 50] = a[i];  
3  }
```

- arr[99]~arr[0]부터 arr[149]~arr[50]처럼 반대 방향으로 복사해서 문제가 없는 코드

코드 9-48

```
1  for (int i = 99; i >= 0; i--) {  
2      arr[i + 50] = a[i];  
3  }
```

※ memove() 함수 코드

- ➔ [코드 9-49](#)
- ➔ [실행 결과](#)

## 9. 메모리 관련 함수

### ■ 메모리 영역에서 특정 값 찾기

#### ■ memchr() 함수

```
void* memchr(const void* ptr, int val, size_t len);
```

※ memchr() 함수

➔ [코드 9-50](#)

➔ [실행 결과](#)

10

문자 관련 함수

## 10. 문자 관련 함수

- ctype.h에 선언된 문자 관련 함수들 중에서도 자주 사용하는 함수

표 9-4 ctype.h에 선언된 문자 관련 함수들

함수	설명
int isalpha(int c)	c가 알파벳에 해당되면 true를 반환하고, 해당되지 않으면 false를 반환한다.
int isalnum(int c)	c가 알파벳이거나 숫자에 해당되면 true를 반환하고, 해당되지 않으면 false를 반환한다.
int isdigit(int c)	c가 숫자에 해당되면 true를 반환하고, 해당되지 않으면 false를 반환한다.
int islower(int c)	c가 소문자면 true를 반환하고, 해당되지 않으면 false를 반환한다.
int isupper(int c)	c가 대문자이면 true를 반환하고, 해당되지 않으면 false를 반환한다.
int isspace(int c)	c가 공백 문자, 탭 또는 줄바꿈 문자 중 한 개면 true를 반환하고, 해당되지 않으면 false를 반환한다.
int ispunct(int c)	c가 쉼표, 마침표, 따옴표 등 특수 기호에 해당되면 true를 반환하고, 해당되지 않으면 false를 반환한다.
int toupper(int c)	c가 소문자면 대문자로 변환해서 반환하고, 해당되지 않으면 c 값을 그대로 반환한다.
int tolower(int c)	c가 대문자면 소문자로 변환해서 반환하고, 해당되지 않으면 c 값을 그대로 반환한다.



11

명령행 인자

# 11. 명령행 인자

- 프로그램을 실행하면서 필요한 정보를 전달하는 문자열
- 윈도우와 리눅스(혹은 맥) 운영체제의 터미널 환경에서 파일을 복사할 때 사용하는 명령

```
copy filename1.txt filename2.txt  
cp filename1.txt filename2.txt
```

- C 프로그램은 명령행 인자 정보를 전달 받는 코드

```
int main(int argc, char* argv[]) // (1)  
int main(int argc, char** argv)  // (2)
```

# 11. 명령행 인자

- 명령행 인자를 출력하는 예제 프로그램

코드 9-51 CmdArgs1.c

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(int argc, char* argv[])
5  {
6      printf("argc = %d\n", argc);
7      for (int i = 0; i < argc; i++) {
8          printf("argv[%d] = %s\n", i, argv[i]);
9      }
10     return 0;
11 }
```

## <실행 결과>

D:\CmdArgs1 filename1.txt filename2.txt

argc = 3

argv[0] = CmdArgs1

argv[1] = filename1.txt

argv[2] = filename2.txt

※ 사용자가 입력한 정수의 약수를 구하는 프로그램

➔ [코드 9-52](#)

➔ [실행 결과](#)