

# 컴퓨터 프로그래밍2

## 문자열

---

장서윤 [pineai@cnu.ac.kr](mailto:pineai@cnu.ac.kr)

# 문자열과 포인터

---

## ❖ 문자열과 포인터

- ▶ 문자열은 배열의 구조를 가짐
- ▶ 첫 번째 문자의 주소로 쓰임

표 12-1 문자열을 저장하는 배열과 포인터의 차이

구분	char 포인터	char 배열
초기화	<code>char *pc = "mango";</code>	<code>char str[80] = "mango";</code>
대입	<code>pc = "banana";</code>	<code>strcpy(str, "banana");</code>
크기	<code>sizeof(pc)</code> → 4바이트	<code>sizeof(str)</code> → 80바이트
수정	<code>pc[0] = 't';</code> → ( X )	<code>str[0] = 't';</code> → ( O )
입력	<code>scanf("%s", pc);</code> → ( X )	<code>scanf("%s", str);</code> → ( O )

# 문자열과 포인터

---

## ❖ 문자열 상수 구현 방법

- ▶ 문자열은 크기가 일정하지 않음
  - ▶ 컴파일러는 문자열 상수를 독특한 방법으로 처리
  - ▶ 문자열은 컴파일 과정에서 첫 번째 문자의 주소로 바뀜
- ▶ 컴파일러는 문자열 char 배열에 따로 보관
  - ▶ 문자열 상수가 있던 곳에는 배열의 위치값 사용

# 문자열 상수 구현 방법

## 예제 12-1 문자열 상수가 주소란 증거

```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     printf("주소값 : %p\n", "mango");
6.     printf("첫 번째 문자 : %c\n", *"mango");
7.     printf("두 번째 문자 : %c\n", *("mango" + 1));
8.     printf("세 번째 문자 : %c\n", "mango"[2]);
9.
10.    return 0;
11. }
```

실행  
결과

주소값 : 00EA58F8

첫 번째 문자 : m

두 번째 문자 : a

세 번째 문자 : n

// 주소값 출력

// 간접참조 연산

// 포인터 연산식

// 배열 표현식

`printf("주소값 : %p\n", "mango" );`

문자열은 따로 배열에 저장

문자열이 저장된 시작 위치값  
(끝의 두 자리만 표시)

첫 번째 문자의 주소 전달

`printf("주소값 : %p\n", F8 );`

F8	F9	FA	FB	FC	FD
m	a	n	g	o	\0

# 문자열과 포인터

---

- ▶ 문자열이 저장된 곳의 위치값 출력
  - ▶ 문자열 “mango”는 배열 형태로 따로 저장
  - ▶ printf 함수의 인수로 그 첫 번째 문자의 주소 사용
  - ▶ printf 함수에서 %p로 출력하면 그 값 16진수로 확인
    - ▶ p는 주로 포인터를 출력할 때 사용하는 변환문자
    - ▶ 포인터 값을 16진수 대문자로 출력
- ▶ 문자열은 char 변수의 주소로 바뀜
  - ▶ 직접 포인터 연산 수행
  - ▶ 간접참조 연산 수행하면 첫 번째 문자 가리킴
  - ▶ 정수 더하는 연산
  - ▶ 배열명처럼 사용 가능

# 문자열 상수 구현 방법

---

- ▶ 주소로 접근하여 문자열 바꾸는 것 금지
  - ▶ ex) \*`"mango"` = `'t'`;
    - ▶ 첫 번째 문자가 저장된 공간에 다른 문자를 대입하여 그 값을 바꾸려는 시도는 위험
    - ▶ 정상적으로 컴파일
    - ▶ 실행할 때 운영체제에 의해서 강제 종료될 가능성
- ▶ 운영체제는 문자열 상수 읽기 전용 메모리 영역에 저장
  - ▶ 그 값을 바꾸는 명령 실행 제한

# 문자열 상수 구현 방법

---

- ▶ 문자열을 주소로 바꾸면?
  - ▶ 포인터 연산 통해 문자열의 시작 위치부터 길이 제한 없이 사용 가능
    - ▶ 문자열의 끝을 알아야 하므로 문자열의 끝 표시하기 위해 널문자 사용
  - ▶ 컴파일러는 문자열 상수를 따로 저장할 때 마지막에 항상 널문자 붙임

# CHAR 포인터로 문자열 사용

---

▶ 문자열은 주소이므로 char 포인터에 대입하여 사용 가능

- ▶ 문자열에 이름을 붙여 사용할 수 있음
- ▶ 다른 문자열로 쉽게 바꿀 수 있음



# CHAR 포인터로 문자열 사용

---

## 예제 12-2 포인터로 문자열을 사용하는 방법

---

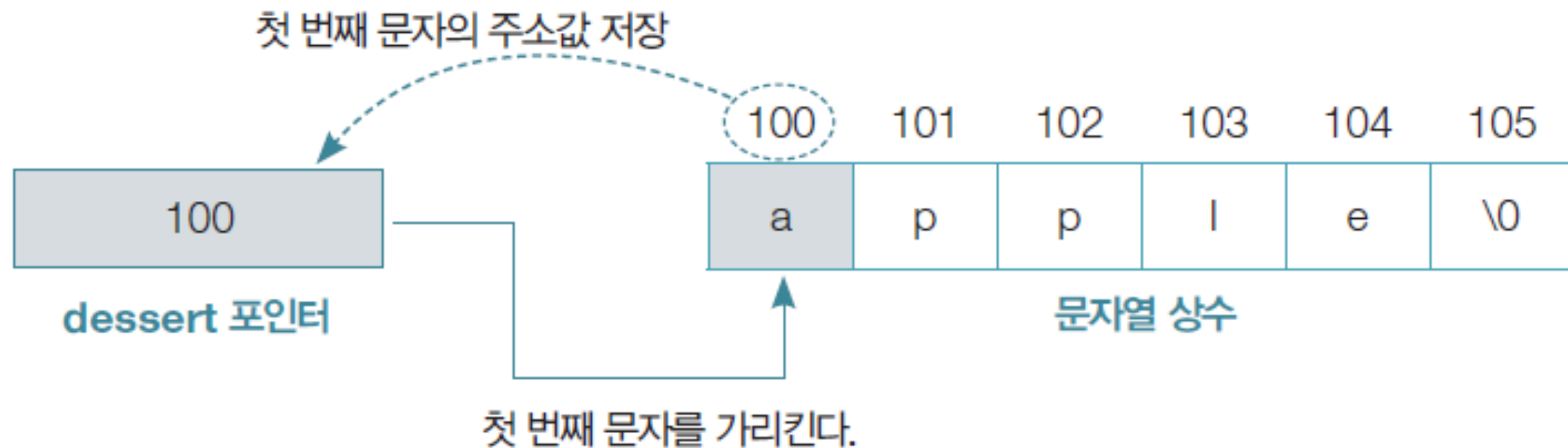
```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     char *dessert = "apple";           // 포인터에 문자열 초기화
6.
7.     printf("오늘 후식은 %s입니다.\n", dessert); // 문자열 출력
8.     dessert = "banana";                // 새로운 문자열 대입
9.     printf("내일 후식은 %s입니다.\n", dessert); // 바뀐 문자열 출력
10.
11.     return 0;
12. }
```

실행 결과  
오늘 후식은 apple입니다.  
내일 후식은 banana입니다.

# CHAR 포인터로 문자열 사용

---

- ▶ char 포인터 선언하고 문자열 상수로 초기화
  - ▶ 문자열은 컴파일 과정에서 별도로 보관
    - ▶ 첫 번째 문자의 주소로 바뀜
  - ▶ 포인터에는 문자열의 시작 위치값만 저장
  - ▶ 포인터 연산 통해 얼마든지 해당 문자열 전체 사용



# CHAR 포인터로 문자열 사용

---

- ▶ printf 함수의 %s 변환문자 - 포인터 연산으로 문자열 출력

---

```
while(*dessert != '\0')    // dessert가 가리키는 문자가 널문자가 아닌 동안
{
    putchar(*dessert);    // dessert가 가리키는 문자 출력
    dessert++;            // dessert로 다음 문자를 가리킨다.
}
```

---

# CHAR 포인터로 문자열 사용

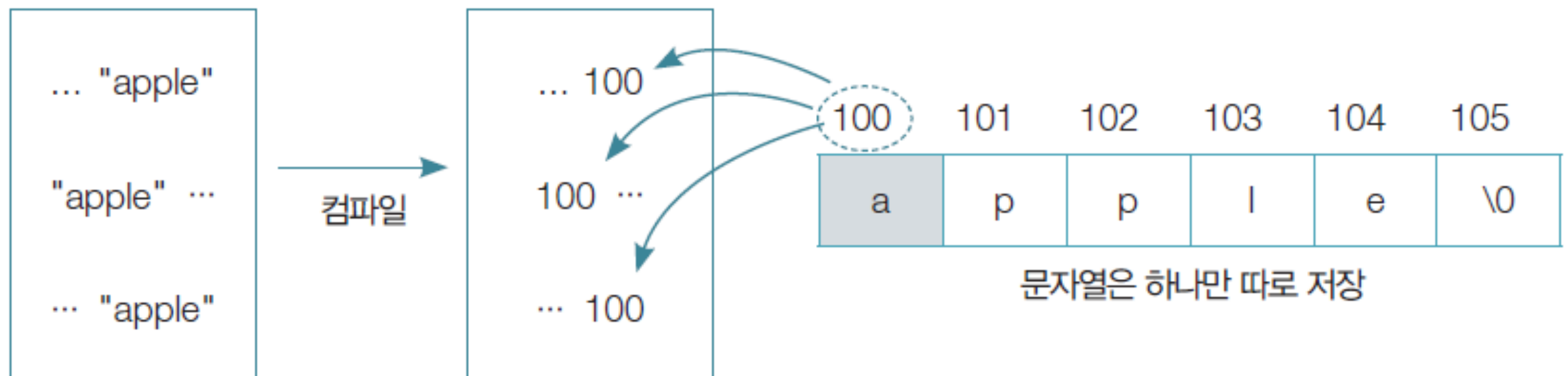
---

- ▶ dessert에 다른 문자열 대입
  - ▶ dessert는 문자열 “banana”의 위치 기억
  - ▶ 이후부터 문자열 “banana”로 사용 가능
- ▶ 같은 문자열 상수는 여러 번 사용해도 하나만 저장
  - ▶ 컴파일러는 같은 문자열을 여러 번 사용한 경우 하나의 문자열만 메모리에 저장
  - ▶ 그 주소 공유하도록 연결

# CHAR 포인터로 문자열 사용

---

- ▶ 같은 문자열을 계속 사용해도 프로그램의 크기가 커지지 않음



# CHAR 포인터로 문자열 사용

---

- ▶ 최적화 여부를 알아보려면 ?
  - ▶ 소스코드에 같은 문자열을 두 번 이상 사용
  - ▶ 그 주소를 출력하여 같은 값인지를 살펴볼 것

---

```
char *pa = "apple";  
char *pb = "apple";  
printf("%p, %p", pa, pb);
```

---

- ▶ 같은 문자열 반복해 사용할 때는 헤더 파일에 매크로명으로 정의하는 편이 관리할 때 효율적

---

```
#define MSG "Be happy!"    // 문자열 "Be happy!"를 이후부터 MSG 이름으로 사용
```

---

# SCANF 함수를 사용한 문자열 입력

---

- ▶ 문자열 상수는 값을 바꿀 수 없음
  - ▶ 바꿀 수 있는 문자열을 원한다면 char 배열 사용
- ▶ scanf 함수는 %s 사용하여 공백 없는 연속된 문자들을 입력 받음

# SCANF 함수를 사용한 문자열 입력

## 예제 12-3 scanf 함수를 사용한 문자열 입력

```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     char str[80];
6.
7.     printf("문자열 입력 : ");
8.     scanf("%s", str);           // %s를 사용하고 배열명을 준다.
9.     printf("첫 번째 단어 : %s\n", str); // 배열에 입력된 문자열 출력
10.    scanf("%s", str);
11.    printf("버퍼에 남아 있는 두 번째 단어 : %s\n", str);
12.
13.    return 0;
14. }
```

실행  
결과

문자열 입력 : apple jam

첫 번째 단어 : apple

버퍼에 남아 있는 두 번째 단어 : jam



# SCANF 함수를 사용한 문자열 입력

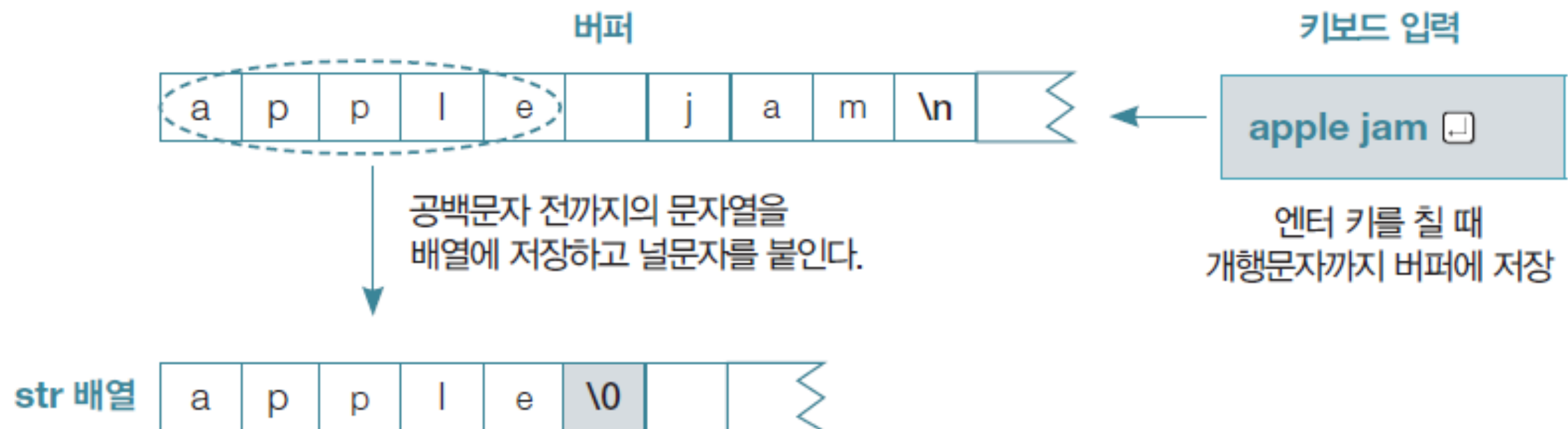
---

- ▶ scanf 함수가 문자열 입력하는 과정과 특징 예제 풀이
  - ▶ scanf 함수는 버퍼 사용
    - ▶ 키보드로 입력한 문자열은 엔터를 칠 때 버퍼에 저장
    - ▶ 버퍼에서 문자열을 가져와 배열에 저장
      - ▶ 중간에 공백문자, 탭문자, 개행문자 있으면 그 이전까지만 저장
    - ▶ 배열에 저장한 문자열의 끝에는 널문자 붙여 문자열 완성

# SCANF 함수를 사용한 문자열 입력

---

- ▶ 최초 입력할 때 apple jam
  - ▶ 공백문자 이전까지만 저장
- ▶ apple만 출력
  - ▶ 나머지 문자열은 버퍼에 남아 있으며 다음에 호출되는 함수가 입력에 사용

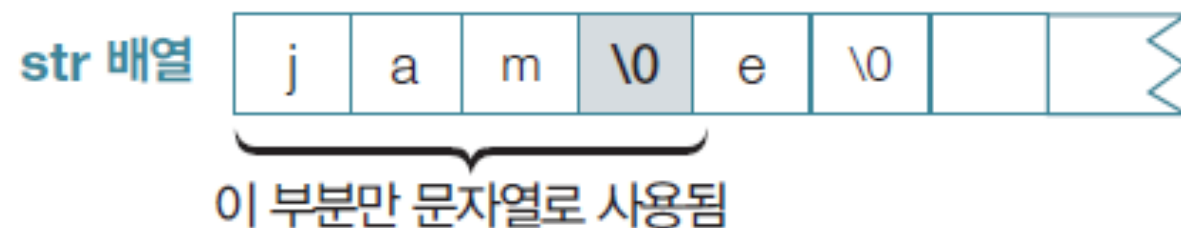


# SCANF 함수를 사용한 문자열 입력

---

## ➤ scanf 함수 호출

- 버퍼에 문자열이 남아 있으므로 남아 있는 문자열 가져와 배열에 저장
- 중간에 있는 공백문자나 탭문자, 개행문자는 모두 건너뛰므로 다음 단어인 jam 입력
- str 배열에 jam이 입력되고 끝에 널문자 추가



# SCANF 함수를 사용한 문자열 입력

---

- ▶ scanf 함수는 문자열 저장할 배열의 크기를 알지 못함
  - ▶ 주소인 배열명을 인수로 받으므로 배열의 시작 위치만 알고 있음
  - ▶ 그 주소를 증가시키면서 버퍼로부터 가져온 문자열을 배열에 저장
  - ▶ 배열의 크기보다 큰 문자열을 입력하면 포인터 연산을 통해 할당된 메모리 공간을 넘어서 저장
    - ▶ 메모리 침범이 발생하므로 프로그램 오류가 발생
    - ▶ 비정상 동작 할 가능성

# SCANF 함수를 사용한 문자열 입력

---

- ▶ scanf 함수로 문자열을 입력할 때
  - ▶ 널문자까지 고려하여 배열의 크기를 넘지 않도록 주의

---

```
char str[5]; // 마지막에 널문자를 붙여야 하므로 최대 4자까지만 입력한다.
```

---

# GETS 함수를 사용한 문자열 입력

---

- ▶ scanf 함수

- ▶ 중간에 공백이 포함된 문자열 한 번에 입력할 수 없음

- ▶ gets 함수

- ▶ **중간의 공백이나 탭문자를 포함하여 문자열 한 줄 입력**

---

```
char *gets(char *str)
```

---


# GETS 함수를 사용한 문자열 입력

---

## 예제 12-4 gets 함수로 한 줄의 문자열 입력

---

```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     char str[80];
6.
7.     printf("공백이 포함된 문자열 입력 : ");
8.     gets(str); // 배열명으로 주고 함수 호출
9.     printf("입력한 문자열은 %s입니다.", str);
10.
11.     return 0;
12. }
```

**실행 결과** 공백이 포함된 문자열 입력 : apple jam   
입력한 문자열은 apple jam입니다.

# GETS 함수를 사용한 문자열 입력

---

- ▶ gets 함수 - 키보드로 엔터 키를 칠 때까지 입력한 한 줄을 char 배열에 저장
- ▶ 버퍼 사용 - 키보드로 입력한 데이터는 일단 버퍼에 저장된 후에 gets 함수가 가져옴
  - ▶ 중간에 있는 공백이나 탭문자도 모두 가져옴
  - ▶ 문장이나 문단도 가져올 수 있음



# GETS 함수를 사용한 문자열 입력

## ▶ gets 함수의 개행문자 처리법

▶ 버퍼에서 개행문자 가져오지만 배열에는 널문자로 바꿔 저장

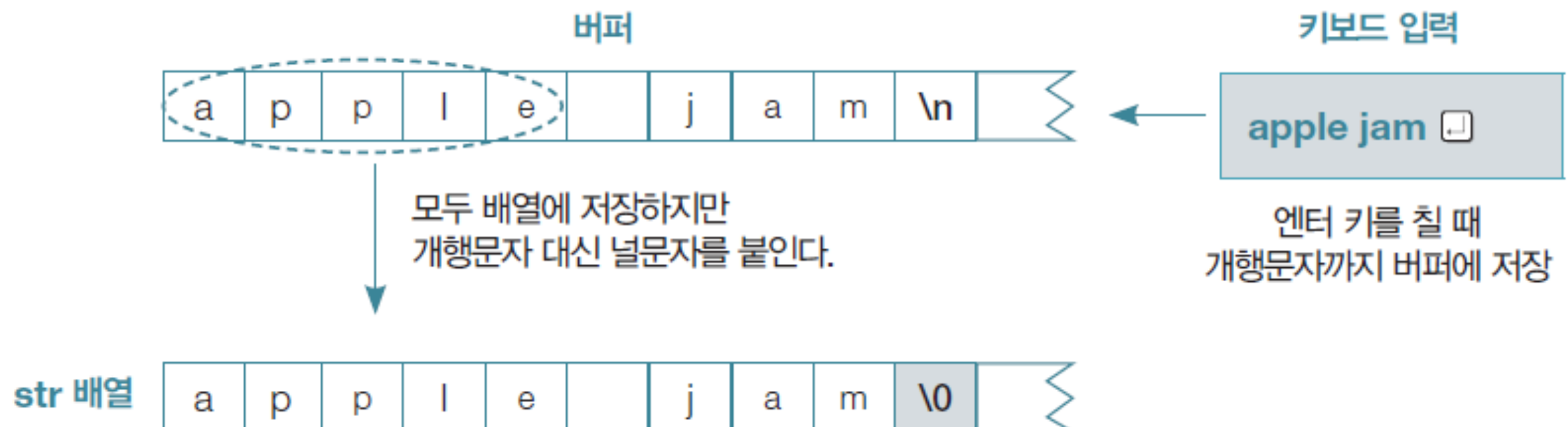
## ▶ 개행문자 처리 방식, 문자열을 배열에 입력하는 방식은 동일

▶ scanf 함수가 문자열을 입력할 때와 같음

## ▶ gets 함수 - 엔터 키만 쳐도 입력 종료

▶ scanf 함수 - white space 입력 시 계속 입력 기다림

▶ 문자열을 구분하는 용도로 쓰고 실제 데이터로 입력하지는 않기 때문

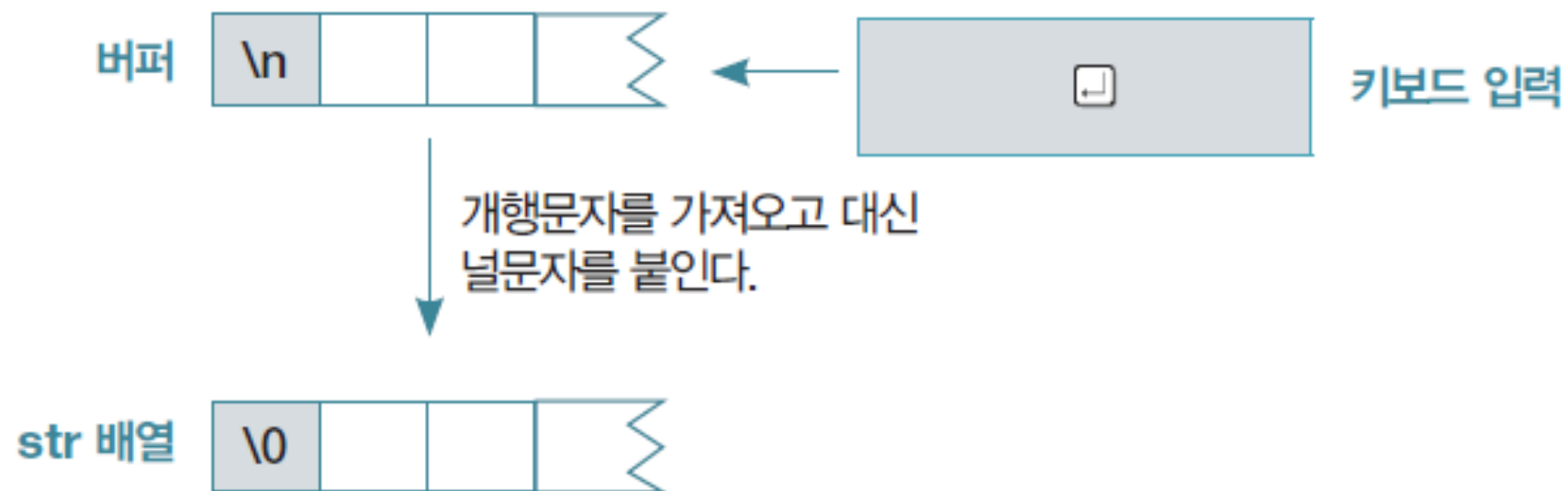


# GETS 함수를 사용한 문자열 입력

---

▶ gets 함수는 문자열의 일부로 white space 입력

▶ 이때 배열의 첫 요소에는 개행문자대신 널문자 입력



# GETS 함수를 사용한 문자열 입력

---

```
1. void my_gets(char *ps)           // ps는 첫 번째 배열 요소를 가리킨다.
2. {
3.     char ch;
4.
5.     while((ch = getchar()) != '\n') // 입력한 문자가 개행문자가 아닌 동안 반복
6.     {
7.         *ps = ch;                 // 배열에 저장하고
8.         ps++;                     // 다음 배열 요소로 이동
9.     }
10.    *ps = '\0';                   // 마지막에 널문자로 마무리한다.
11. }
```

---

gets(str)과 같음

# FGETS 함수를 사용한 문자열 입력

---

## 예제 12-5 fgets 함수의 문자열 입력 방법

---

```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     char str[80];
6.
7.     printf("공백이 포함된 문자열 입력 : ");
8.     fgets(str, sizeof(str), stdin);      // 문자열 입력
9.     printf("입력된 문자열은 %s입니다\n", str); // 문자열 출력
10.
11.     return 0;
12. }
```

실행 결과  
공백이 포함된 문자열 입력 : apple jam   
입력된 문자열은 apple jam  
입니다

# FGETS 함수를 사용한 문자열 입력

---

## ➤ fgets 함수

---

```
char *fgets(char *str, int n, FILE *stream)
```

---

- 문자열을 저장할 배열명 외에 배열의 크기와 표준 입력버퍼를 뜻하는 stdin을 함께 사용
- 두 번째 인수로 배열의 크기 알려줌
  - 배열의 크기 넘는 문자열 입력해도 배열의 크기 만큼만 저장
  - 마지막에 항상 널문자 붙이므로 최대 '배열의 크기 - 1개'의 문자만 저장
- 배열의 크기가 5바이트라면 str 배열에는 appl까지만 저장

str 배열

a	p	p	l	\0
---	---	---	---	----

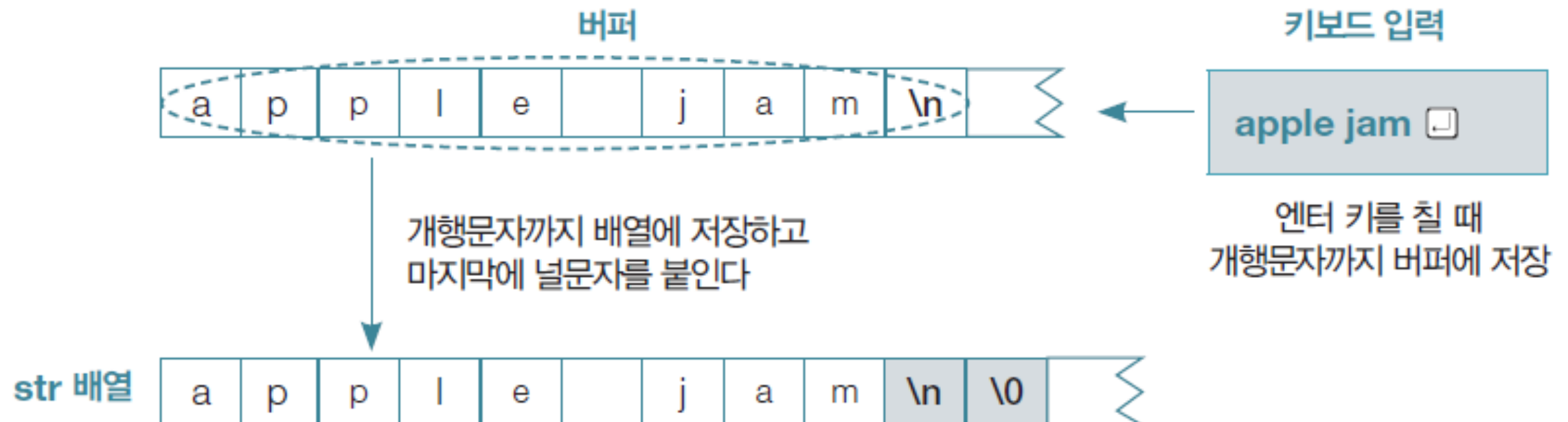
# FGETS 함수를 사용한 문자열 입력

## ▶ 문자열 입력하는 방식

▶ gets와 같이 버퍼 사용, 공백이나 탭문자 포함해 한 줄 입력

## ▶ 개행문자의 처리 방식은 다름

▶ 버퍼에 있는 개행문자도 배열에 저장하고 널문자 붙여 문자열 완성



# FGETS 함수를 사용한 문자열 입력

---

▶ 입력된 개행문자는 불필요한 경우 다음 공식에 따라 제거

- ▶ strlen 함수는 배열명 인수로 받아 널문자 이전까지의 문자 수 세어 반환 (사용할 때 string.h 헤더 파일을 인클루드)

```
str [ strlen ( str ) - 1 ] = '\0'
```

널문자 이전까지 문자 수 10개

9는 배열에서 개행문자가 저장된 곳의 위치

개행문자를 널문자로 바꿈

# 표준 입력 함수의 버퍼 공유 문제

---

- ▶ 표준 입력 함수들은 입력 버퍼 공유
  - ▶ gets나 fgets 함수에서 개행문자 입력 종료 조건으로 사용시
    - ▶ 앞서 입력한 함수가 버퍼에 개행문자 남겨 놓는 경우
    - ▶ 호출되는 함수가 버퍼에서 개행문자만 가져오고 입력 끝내는 문제가 생기기 때문



# 표준 입력 함수의 버퍼 공유 문제

---

예제 12-6 개행문자로 인해 gets 함수가 입력을 못하는 경우

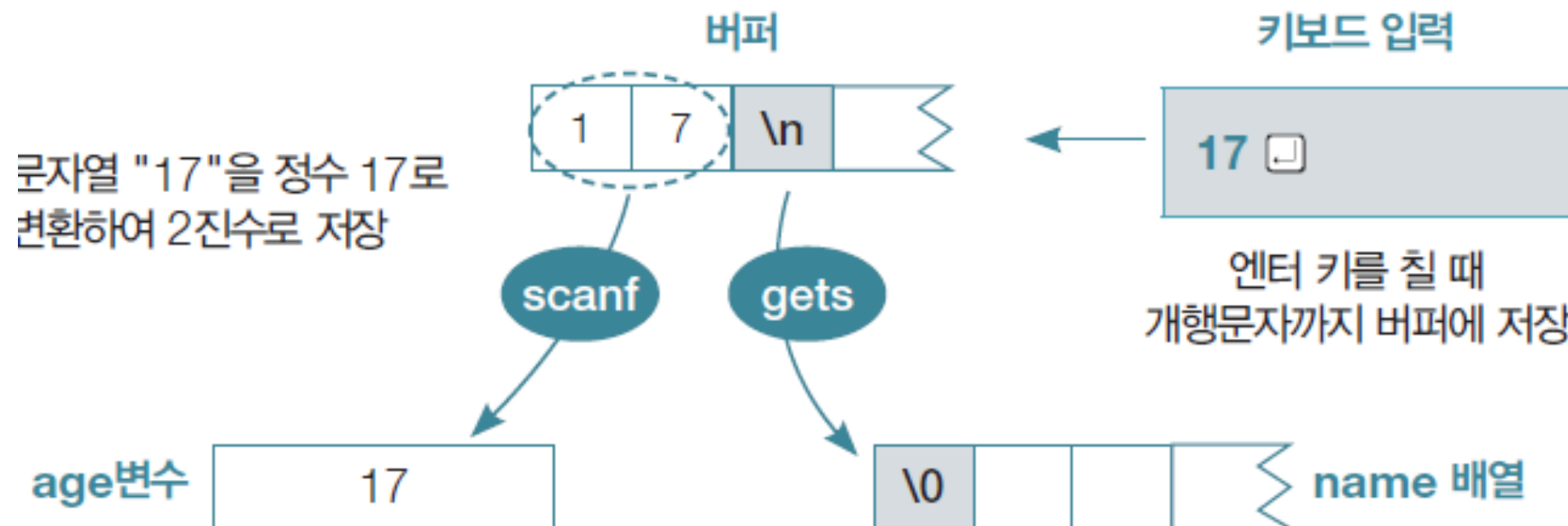
```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     int age;                // 나이를 저장할 변수
6.     char name[20];          // 이름을 저장할 배열
7.
8.     printf("나이 입력 : ");
9.     scanf("%d", &age);      // 나이 입력
10.    printf("이름 입력 : ");
11.    gets(name);              // 이름 입력
12.    printf("나이 : %d, 이름 : %s\n", age, name);
13.
14.    return 0;
15. }
```

실행 결과  
나이 입력 : 17   
이름 입력 : 나이 : 17, 이름 :

# 표준 입력 함수의 버퍼 공유 문제

---

- ▶ 키보드로 입력한 나이는 문자열로 버퍼에 저장
  - ▶ scanf 함수가 숫자로 변환하여 age 변수에 저장
- ▶ 버퍼에 남아 있는 개행문자가 11행의 gets 함수 입력으로 쓰임



# 표준 입력 함수의 버퍼 공유 문제

---

- ▶ gets 함수 - 버퍼에서 개행문자를 가져와 입력 끝냄
  - ▶ 키보드로 이름을 입력하는 과정 생략
  - ▶ name 배열의 첫 번째 요소에는 널문자 저장
  - ▶ 이름으로는 아무것도 출력되지 않음
    - ▶ 버퍼 내용 지워야 하는 필요성
- ▶ 버퍼에 남아 있는 개행문자 지우는 방법
  - ▶ 개행문자를 읽어 들이는 문자 입력 함수 호출
    - ▶ 불필요한 데이터 많다면 반복호출 필요

---

<code>getchar();</code>	// 버퍼에서 하나의 문자를 읽어서 반환, 반환 문자는 버림
<code>scanf("%*c");</code>	// 버퍼에서 하나의 문자를 읽어서 버림, 변수는 필요 없음
<code>fgetc(stdin);</code>	// 버퍼에서 하나의 문자를 읽어서 반환, 반환 문자는 버림

---

# 문자열을 출력하는 PUTS, FPUTS 함수

---

- ▶ 화면에 문자열을 출력할 때 사용하는 전용 출력 함수
- ▶ 정상 출력된 경우 0 반환, 출력에 실패하면 -1 (EOF) 반환

---

// str 배열에 저장된 문자열을 출력하고 자동으로 줄을 바꿉니다.

int puts(const char \*str)

// str 배열에 저장된 문자열을 출력하고 줄은 바꾸지 않습니다.

int fputs(const char \*str, FILE \*stream)

---

# 문자열을 출력하는 PUTS, FPUTS 함수

---

## 예제 12-7 문자열을 출력하는 puts와 fputs 함수

---

```
1. #include <stdio.h>
2.
3. int main(void)
4. {
5.     char str[80] = "apple juice"; // 배열에 문자열 초기화
6.     char *ps = "banana";         // 포인터에 문자열 연결
7.
8.     puts(str);                    // apple juice 출력하고 줄 바꿈
9.     fputs(ps, stdout);            // banana만 출력
10.    puts("milk");                  // banana에 이어 milk 출력
11.
12.    return 0;
13. }
```

실행  
결과 apple juice  
          bananamilk

# 문자열을 출력하는 PUTS, FPUTS 함수

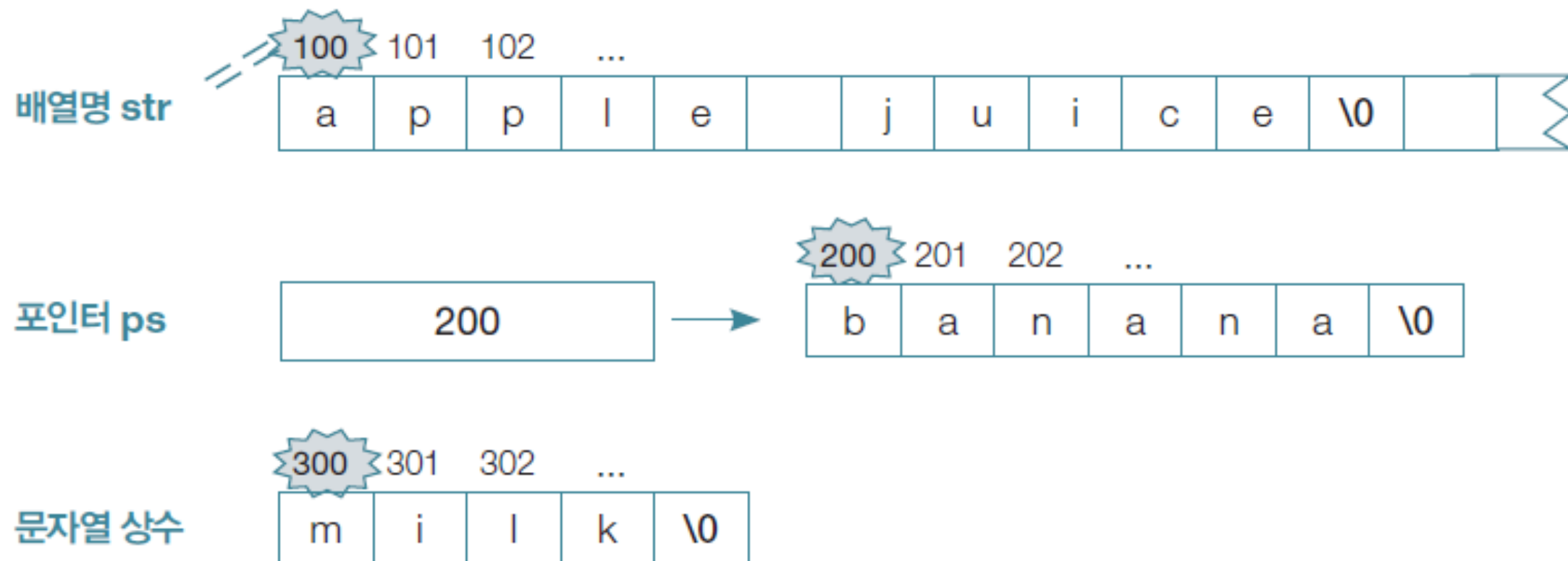
---

## ▶ puts와 fputs

- ▶ 문자열의 시작 위치부터 널문자가 나올 때까지 모든 문자 출력
- ▶ char 배열의 배열명이나 문자열 상수를 연결하고 있는 포인터를 인수로 줄 수 있음
- ▶ 10행처럼 문자열 상수를 직접 사용하는 것도 가능
- ▶ 어떤 경우든 문자열에서 첫 번째 문자의 주소가 되므로 결국 문자열 출력

# 문자열을 출력하는 PUTS, FPUTS 함수

- ▶ puts 함수 - fputs 함수와 달리 문자열 출력한 후 자동 줄 바꿈
  - ▶ 줄 바꿈 기능 원하지 않을 때는 쓰지 말 것



# 문자열 연산 함수

---

## ❖ 문자열 연산은 모두 함수로 수행

표 12-2 기본적인 문자열 연산 함수

연산 기능	사용 방법	실행 결과
대입	<code>strcpy(str1, str2)</code>	문자열 str2를 str1에 복사
길이 계산	<code>strlen(str);</code>	문자열 str의 길이(문자 수)를 구해 반환
붙이기	<code>strcat(str1, str2);</code>	문자열 str2를 str1문자열 뒤에 이어 붙임
비교	<code>strcmp(str1, str2);</code>	문자열 str1이 str2보다 크면 1 반환 문자열 str1이 str2보다 작으면 -1 반환 str1과 str2가 같은 문자열이면 0 반환

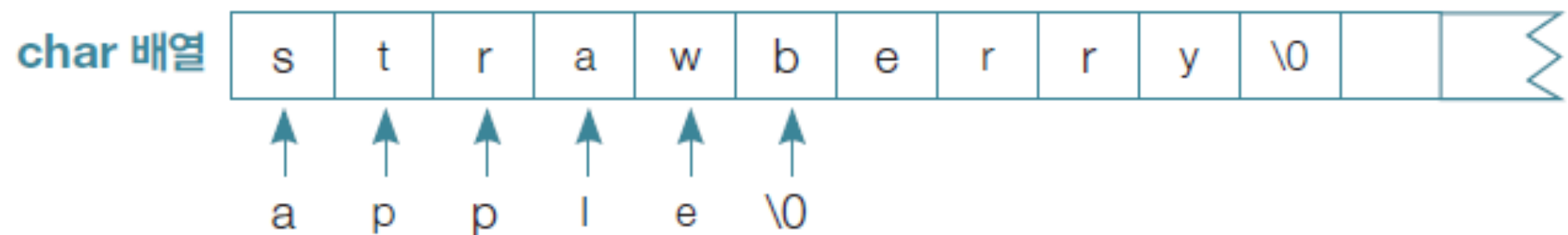


# 문자열을 대입하는 STRCPY 함수

---

## ▶ char 배열

- ▶ 문자열을 저장하는 변수의 역할 하며 쉽게 문자열로 초기화
- ▶ 다른 문자열로 바꾸는 것은 문자를 하나씩 옮겨야
  - ▶ 문자열 "strawberry"가 저장된 배열을 "apple"로
    - ▶ 문자를 하나씩 대입하고 마지막에 널문자도 저장



# 문자열을 대입하는 STRCPY 함수

---

➤ strcpy 함수 - char 배열에 문자열 복사하는 대입 연산 수행

## 예제 12-8 strcpy 함수의 사용법

---

```
1. #include <stdio.h>
2. #include <string.h>                // strcpy 함수를 사용하기 위해 인클루드함
3.
4. int main(void)
5. {
6.     char str1[80] = "strawberry";    // char 배열에 문자열 초기화
7.     char str2[80] = "apple";         // char 배열에 문자열 초기화
8.     char *ps1 = "banana";           // 포인터로 문자열 상수 연결
9.     char *ps2 = str2;               // 포인터로 배열 연결
10.
11.    printf("최초 문자열 : %s\n", str1);
12.    strcpy(str1, str2);              // 다른 char 배열의 문자열 복사
13.    printf("바뀐 문자열 : %s\n", str1);
```

# 문자열을 대입하는 STRCPY 함수

---

```
14.  
15.     strcpy(str1, ps1);           // 문자열 상수를 연결한 포인터 사용  
16.     printf("바뀐 문자열 : %s\n", str1);  
17.  
18.     strcpy(str1, ps2);           // 배열을 연결한 포인터 사용  
19.     printf("바뀐 문자열 : %s\n", str1);  
20.  
21.     strcpy(str1, "banana");       // 문자열 상수 사용  
22.     printf("바뀐 문자열 : %s\n", str1);  
23.  
24.     return 0;  
25. }
```

실행  
결과

```
최초 문자열 : strawberry  
바뀐 문자열 : apple  
바뀐 문자열 : banana  
바뀐 문자열 : apple  
바뀐 문자열 : banana
```

# 문자열을 대입하는 STRCPY 함수

---

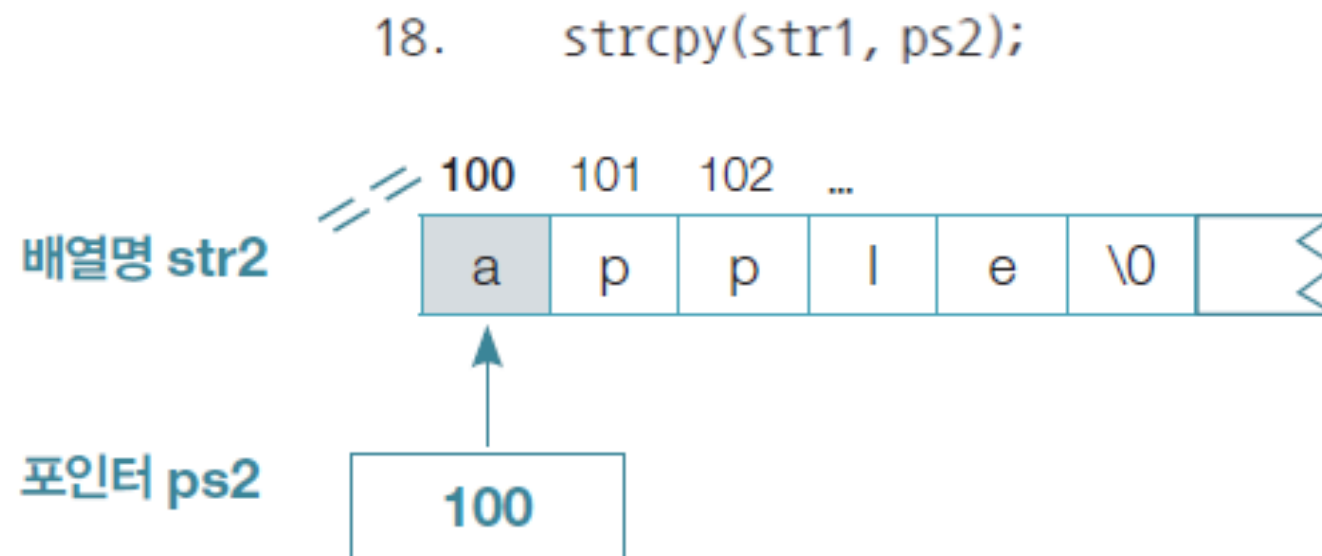
## ➤ strcpy 함수

➤ 복사 받을 곳의 배열명 첫 번째 인수

➤ 복사할 문자열을 두 번째 인수

➤ 문자열 복사 방식은 문자열의 첫 번째 문자부터 널문자가 나올 때까지 문자를 하나씩 배열에 옮겨 저장

➤ str2는 char 배열의 배열명이므로 첫 번째 문자의 주소, ps2는 그 값을 저장한 포인터



# 문자열을 대입하는 STRCPY 함수

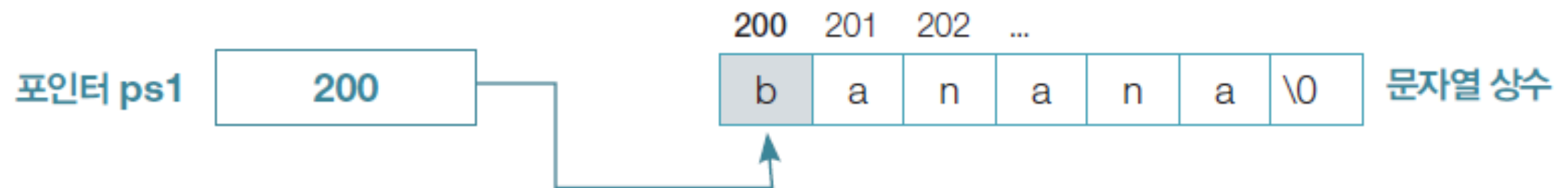
---

➤ ps1은 문자열 상수 “banana”의 위치 저장

➤ 첫 번째 문자의 주소      15.      strcpy(str1, ps1);

➤ 문자열 상수는 첫 번째 문자의 주소로 바뀜

➤ 직접 사용 가능      21.      strcpy(str1, "banana");



# 문자열을 대입하는 STRCPY 함수

---

## ▶ strcpy 함수의 잘못된 사용 예

---

```
strcpy("banana", "apple");    // 문자열 상수를 바꾸고자 함  
strcpy(ps1, "apple");         // ps1이 연결하고 있는 문자열 상수가 바뀜
```

---

## ▶ 두 번째 인수는 다양한 값 사용 가능

## ▶ 첫 번째 인수로 사용할 수 있는 값은 제한적

- ▶ char 배열이나 그 배열명 저장한 포인터만 가능
- ▶ 문자열 상수는 값을 바꿀 수 없으므로 첫 번째 인수로 사용하면 프로그램을 실행할 때 오류 발생
- ▶ 문자열 상수를 연결하고 있는 포인터도 사용 불가

# STRCPY 함수 구현 방법

.....

## 예제 12-9 strcpy와 기능이 같은 함수의 구현

---

```
1. #include <stdio.h>
2.
3. char *my_strcpy(char *pd, char *ps);    // 함수 선언
4.
5. int main(void)
6. {
7.     char str[80] = "strawberry";
8.
9.     printf("바꾸기 전 문자열 : %s\n", str);
10.    my_strcpy(str, "apple");              // 문자열 "apple" 복사
11.    printf("바꾼 후 문자열 : %s\n", str);
12.    printf("다른 문자열 대입 : %s\n", my_strcpy(str, "kiwi"));    // 반환값으로 출력
13.
14.    return 0;
15. }
16.
```

# STRCPY 함수 구현 방법

str은 char배열의 배열명이므로 첫번째 요소의 주소인데 첫번째 요소가 char형이므로 char 포인터로 받아야 한다.

```
17. char *my_strcpy(char *pd, char *ps)    // 복사받을 곳(pd)과 복사할 곳(ps)의 포인터
18. {
19.     char *po = pd;                    // pd값을 나중에 반환하기 위해 보관
20.
21.     while(*ps != '\0')                // ps가 가리키는 문자가 널문자가 아닌 동안
22.     {
23.         *pd = *ps;                     // ps가 가리키는 문자를 pd가 가리키는 위치에 대입
24.         pd++;                          // 복사받을 다음 위치로 포인터 증가
25.         ps++;                          // 복사할 다음 문자의 위치로 포인터 증가
26.     }
27.     *pd = '\0';                       // 복사가 모두 끝난 후 복사받을 곳에 널문자로 마무리
28.
29.     return po;                        // 복사가 끝난 저장 공간의 시작 주소 반환
30. }
```

실행 결과  
바꾸기 전 문자열 : strawberry  
바꾼 후 문자열 : apple  
다른 문자열 대입 : kiwi



# STRCPY 함수 구현 방법

## ➤ strcpy 함수

- 포인터 써서 문자열 복사
- 문자열이 저장된 메모리에서 첫 번째 문자 주소 인수로 받음

## ➤ 메모리 상태



# STRCPY 함수 구현 방법

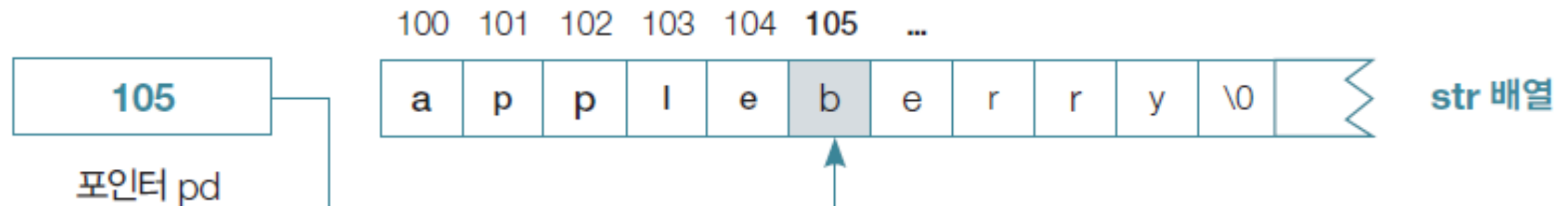
---

▶ ps가 가리키는 문자가 널문자가 되어 반복 종료

▶ 널문자는 저장되지 않으므로 반복을 종료한 후 pd가 가리키는 곳에 반드시 널문자 저장

▶ 이 과정을 생략하면 복사가 끝난 후의 문자열은 appleberry가 되므로 주의

27.     \*pd = '\0';



## 12.2 문자열 연산 함수

---

### ▶ strcpy 함수

- ▶ 포인터 연산으로 문자열을 복사하므로 복사 받을 배열의 크기보다 큰 문자열 복사할 수도 있음
  - ▶ 할당되지 않은 메모리 공간을 침범하므로 오류 발생
  - ▶ 복사할 문자열의 크기는 널문자를 포함 복사 받을 배열의 크기를 넘지 않도록 주의

# 원하는 개수의 문자만을 복사하는 STRNCPY 함수

---

## ▶strncpy 함수

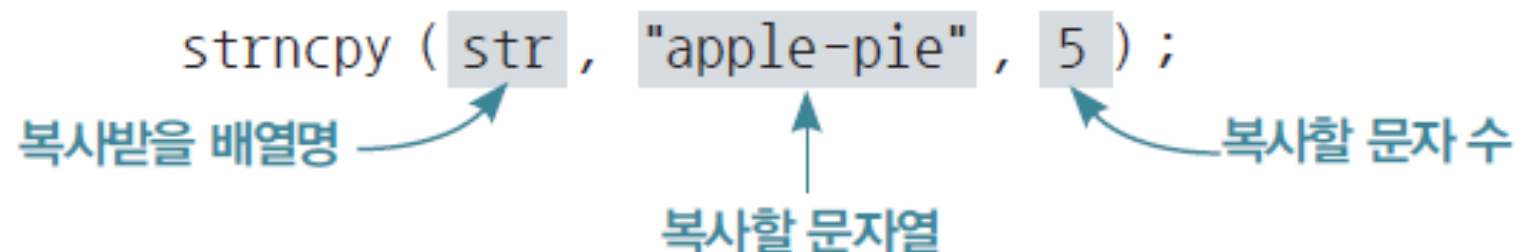
### ▶문자열을 복사할 때 문자의 수 지정할 수 있음

- ▶ Ex) 문자열 “apple-pie”의 앞에서 다섯 개의 문자만 char 배열 str에 복사

---

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

---



The diagram shows the function call `strncpy (str, "apple-pie", 5);` with three annotations:   
1. An arrow points from the text "복사받을 배열명" (Destination array name) to the argument `str`.   
2. An arrow points from the text "복사할 문자열" (String to be copied) to the argument `"apple-pie"`.   
3. An arrow points from the text "복사할 문자 수" (Number of characters to copy) to the argument `5`.

# 원하는 개수의 문자만을 복사하는 STRNCPY 함수

---

## 예제 12-10 strncpy 함수를 사용한 문자열 복사

---

```
1. #include <stdio.h>
2. #include <string.h>           // strncpy 함수 사용을 위한 헤더 파일 포함
3.
4. int main(void)
5. {
6.     char str[20] = "mango tree"; // 배열 초기화
7.
8.     strncpy(str, "apple-pie", 5); // "apple-pie"에서 다섯 문자만 복사
9.     printf("%s\n", str);         // 복사받은 문자열 출력
10.
11.     return 0;
12. }
```

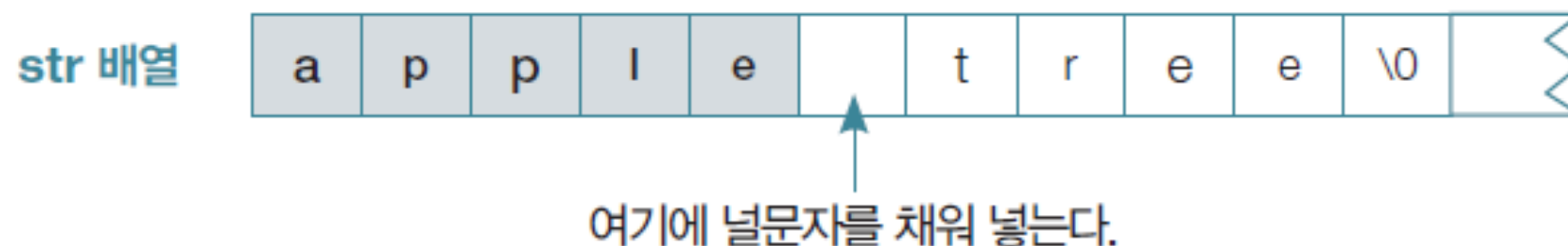
실행  
결과 apple tree

# 원하는 개수의 문자만을 복사하는 STRNCPY 함수

.....

➤ `strncpy` 함수 - 복사할 문자열에서 지정한 개수만큼 문자 복사 후 널 문자 저장하지 않음

- `str` 배열에서 `mango`만 `apple`로 바뀜
  - `str`에 저장된 문자열은 “apple tree”
- `str` 배열이 문자열 “apple”로만 쓰이도록 하려면?
  - `apple`을 복사 후에 널문자 별도 저장해야 함



---

```
str[5] = '\0';
```

---

# 문자열 길이를 계산하는 STRLEN 함수

---

- ▶ 문자열 저장하는 char 배열 - 충분히 크게 선언해 사용
  - ▶ 배열에 저장된 문자열의 길이는 배열의 크기와 다를 수 있음
- ▶ strlen 함수
  - ▶ 배열에 저장된 문자열의 실제 길이 구해 반환


---

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

---

## 예제 12-11 두 문자열 중 길이가 긴 단어 출력

```
1. #include <stdio.h>
2. #include <string.h>           // strlen 함수 사용을 위한 헤더 파일 포함
3.
4. int main(void)
5. {
6.     char str1[80], str2[80];    // 두 문자열을 입력할 배열
7.     char *resp;                // 문자열이 긴 배열을 선택할 포인터
8.
9.     printf("2개의 과일 이름 입력 : ");
10.    scanf("%s%s", str1, str2);  // 2개의 문자열 입력
11.    if( strlen(str1) > strlen(str2) ) // 배열에 입력된 문자열의 길이 비교
12.        resp = str1;            // 첫 번째 배열이 긴 경우 선택
13.    else
14.        resp = str2;            // 두 번째 배열이 긴 경우 선택
15.    printf("이름이 긴 과일은 : %s\n", resp); // 선택된 배열의 문자열 출력
16.
17.    return 0;
18. }
```

**실행 결과** 2개의 과일 이름 입력 : banana strawberry   
이름이 긴 과일은 : strawberry



# 문자열 길이를 계산하는 STRLEN 함수

---

## ▶ 같은 기능하는 함수 예제

---

```
int my_strlen(char *ps)           // ps는 배열명을 저장하고 첫 번째 배열 요소를 가리킴
{
    int cnt = 0;                  // 문자 수를 세기 위한 변수

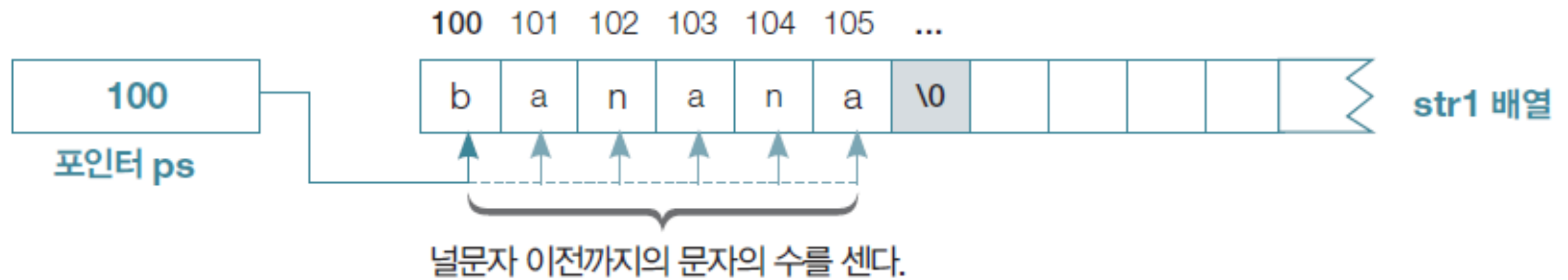
    while(*ps != '\0')            // ps가 가리키는 문자가 널문자가 아닌 동안
    {
        cnt++;                    // 문자 수 증가
        ps++;                     // 포인터를 다음 문자의 위치로 이동
    }
    return cnt;                   // 전체 문자 수 반환
}
```

---

# 문자열 길이를 계산하는 STRLEN 함수

---

- ▶ 같은 기능하는 함수 예제
  - ▶ 메모리 상황



# 문자열 길이를 계산하는 STRLEN 함수

---

- ▶ sizeof 연산자 - 배열 전체 크기 계산
  - ▶ strlen을 sizeof로 바꾸면 어떤 문자열을 입력하더라도 조건식 항상 거짓

---

```
char str[80] = "apple";  
printf("%d", sizeof(str));    // 출력 결과는 80  
printf("%d", strlen(str));    // 출력 결과는 5
```

---

# 문자열을 붙이는 STRCAT, STRNCAT 함수

---

## ➤ strcpy 함수

➤ 초기화된 문자열 지우고 새로운 문자열로 바꿀 때 사용

## ➤ 배열에 있는 문자열의 뒤에 이어 붙일 때

➤ strcat 또는 strncat 함수 사용

➤ strcat 함수는 문자열을 이어 붙이기

➤ strncat 함수는 지정한 문자의 개수만큼 붙이기

---

```
char *strcat(char *dest, const char *src)
```

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

---

# 문자열을 붙이는 STRCAT, STRNCAT 함수

---

## 예제 12-12 strcat, strncat 함수를 사용한 문자열 붙이기

---

```
1. #include <stdio.h>
2. #include <string.h>           // strcat, strncat 함수 사용을 위한 헤더 파일 포함
3.
4. int main(void)
5. {
6.     char str[80] = "straw";    // 문자열 초기화
7.
8.     strcat(str, "berry");       // str 배열에 문자열 붙이기
9.     printf("%s\n", str);
10.    strncat(str, "piece", 3);   // str 배열에 3개의 문자 붙이기
11.    printf("%s\n", str);
12.
13.    return 0;
14. }
```

실행  
결과  
strawberry  
strawberrypie

# 문자열을 붙이는 STRCAT, STRNCAT 함수

---

## ❖ strcat 함수

- ▶ str 배열에 있는 문자열 뒤에 두 번째 인수로 주어지는 문자열을 이어 붙임
- ▶ 먼저 붙여 넣을 배열에서 널문자의 위치를 찾고 그 위치부터 붙여 넣을 문자열 복사
- ▶ 붙여 넣기가 끝난 후에는 널문자 저장하여 마무리

# 문자열을 붙이는 STRCAT, STRNCAT 함수

---

## ▶ 같은 기능을 하는 함수

```
char *my_strcat(char *pd, char *ps)
{
    char *po = pd;           // 배열의 처음 위치 보관

    while(*pd != '\0')       // pd를 널문자의 위치로 이동
    {
        pd++;
    }

    while(*ps != '\0')       // 여기부터는 문자열 복사와 같음
    {
        *pd = *ps;
        pd++;
        ps++;
    }

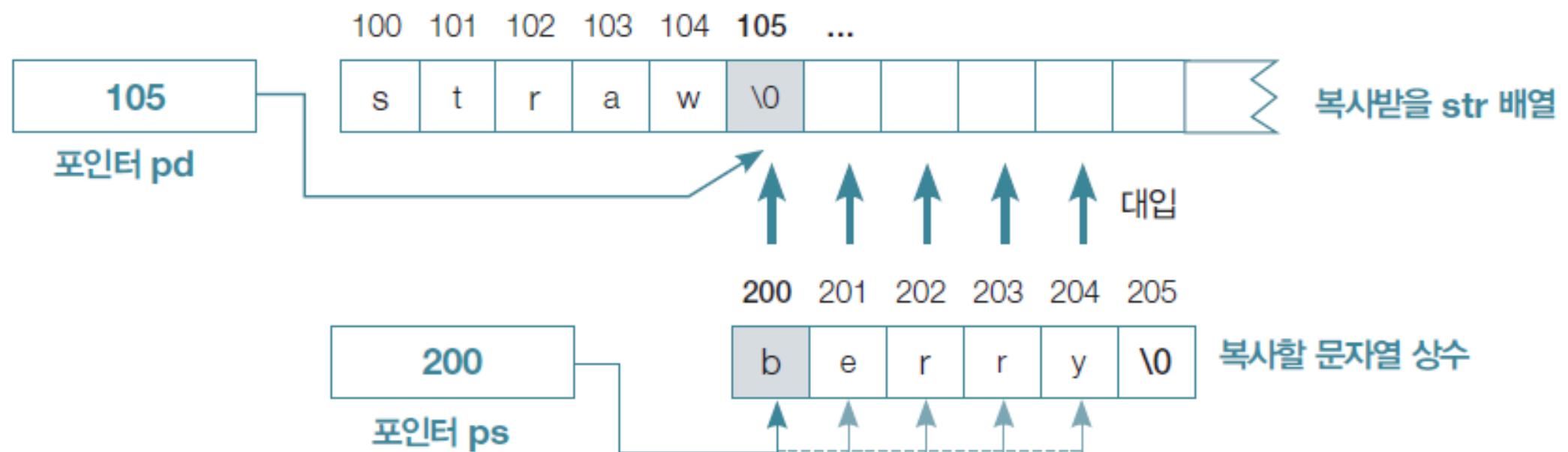
    *pd = '\0';

    return po;               // 붙여넣은 배열의 시작 위치 반환
}
```

---

# 문자열을 붙이는 STRCAT, STRNCAT 함수

.....





# 문자열을 붙이는 STRCAT, STRNCAT 함수

---

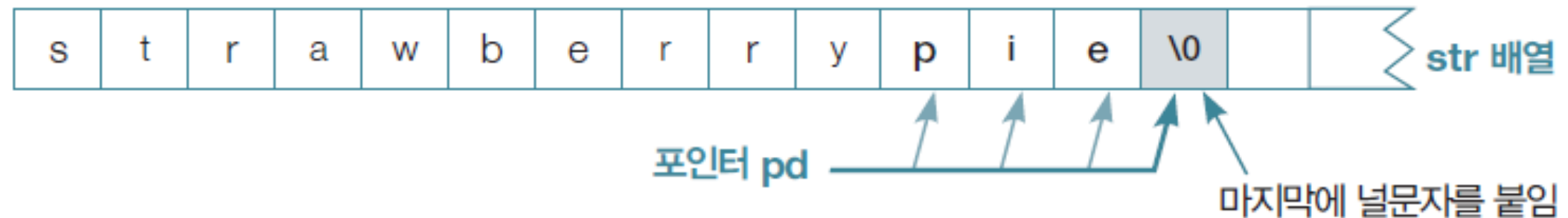
## ▶ strcat 함수

- ▶ 문자열을 덧붙이는 것이므로 붙여 넣기가 되는 배열의 크기가 충분히 커야 함
- ▶ 붙여 넣기 전에 먼저 널문자의 위치를 찾으므로 반드시 초기화를 해야
  - ▶ 쓰레기값의 중간부터 붙여 넣을 가능성이 큼
  - ▶ Ex) 6행처럼 특별한 문자열로 초기화하거나 최소한 첫 번째 문자가 널문자가 되도록 초기화

# 문자열을 붙이는 STRCAT, STRNCAT 함수

---

- ▶ strncat 함수
  - ▶ 붙여 넣을 문자 수 지정 가능
  - ▶ strncat 함수는 strncpy 함수와 달리 붙여 넣은 후 널문자를 저장하여 문자열 완성



# 문자열을 붙이는 STRCAT, STRNCAT 함수

---

## ➤ strcmp 함수

➤ 두 문자열의 사전 순서 판단하여 그 결과 반환


➤ 사전 순서는 사전에 단어가 수록되는 알파벳 순서

➤ 함수의 사용법과 반환값

---

```
strcmp(str1, str2);
```

---

- str1이 str2보다 사전에 나중에 나오면 1 반환 
- str1이 str2보다 사전에 먼저 나오면 -1 반환
- str1과 str2가 같은 문자열이면 0 반환

```

1. #include <stdio.h>
..... 2. #include <string.h> .....
3.
4. int main(void)
5. {
6.     char str1[80] = "pear";
7.     char str2[80] = "peach";
8.
9.     printf("사전에 나중에 나오는 과일 이름 : ");
10.    if( strcmp(str1, str2) > 0 )    // str1이 str2보다 크면(사전에 나중에 나오면)
11.        printf("%s\n", str1);    // str1 출력
12.    else
13.        printf("%s\n", str2);    // 그렇지 않으면 str2 출력
14.
15.    printf("앞에서 3개의 문자를 비교하면 ");
16.    if( strncmp(str1, str2, 3) == 0 )    // 앞에서 3개의 문자가 같으면 0 반환
17.        printf("같다.\n");
18.    else
19.        printf("다르다.\n");
20.
21.    return 0;
22. }

```

실행 결과  
사전에 나중에 나오는 과일 이름 : pear  
앞에서 3개의 문자를 비교하면 같다.

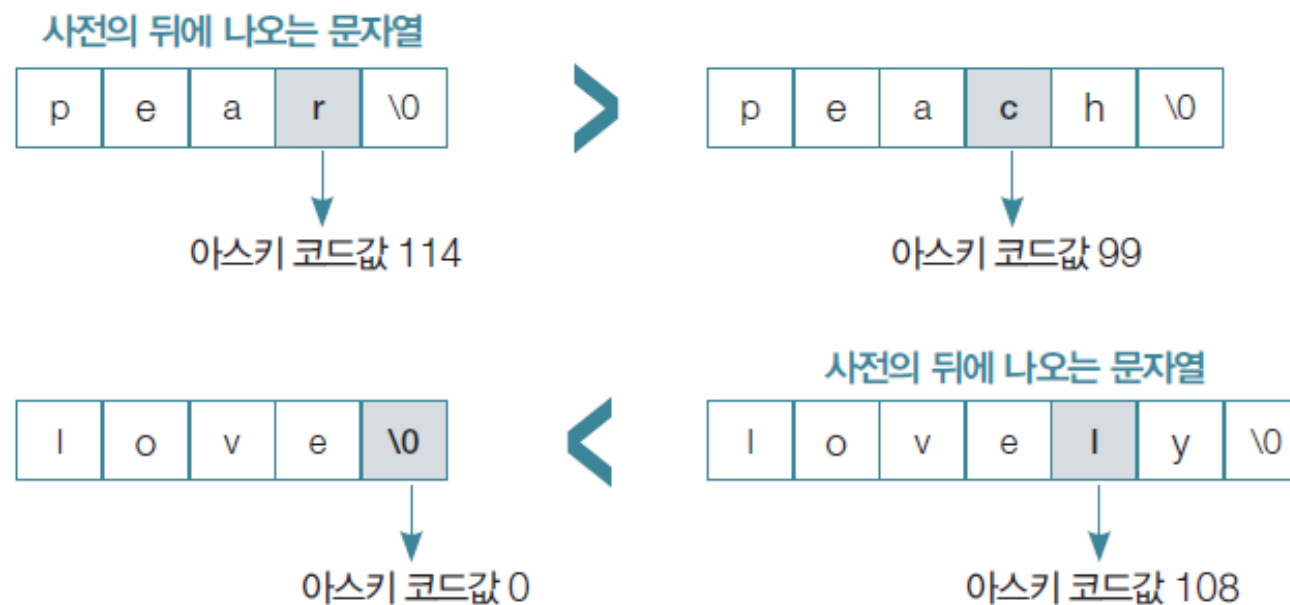
# 문자열을 비교하는 STRCMP, STRNCMP 함수

## ➤ strcmp 함수

- 두 문자열에서 우선 첫 문자의 아스키 코드값 비교
- 아스키 코드값이 크면 사전의 뒤에 나오는 문자열



- 첫 문자가 같으면 다음 문자의 아스키 코드값을 차례로 비교하여 판단합니다.
- 마지막 문자까지 같으면 같은 문자열



# 문자열을 비교하는 STRCMP, STRNCMP 함수

---

- ▶ 문자열 비교 예제 풀이
  - ▶ 사전 순서 판단한 결과는 반환값으로 알려줌
  - ▶ pear가 peach보다 사전의 뒤에 나오는 문자열
    - ▶ 1 반환하고 if문의 조건식이 참이 되어 pear 출력
  - ▶ cf) str1이 apple이고 str2가 banana였다면?
    - ▶ -1 반환하고 조건식은 거짓이 되어 banana 출력
  - ▶ 사전의 뒤에 나오는 문자열 출력

# 문자열을 비교하는 STRCMP, STRNCMP 함수

---

## ➤ strcmp 와 같은 기능 함수

---

```
int my_strcmp(char *pa, char *pb)
{
    while( (*pa == *pb) && (*pa != '\0') )    // 두 문자가 같으나 널문자가 아닌 경우
    {
        pa++;                                // 다음 문자로 이동
        pb++;                                // 다음 문자로 이동
    }
    // 반복문 이후 이 시점에서는 두 문자가 다르거나 둘 다 널문자임
    if(*pa > *pb) return 1;                    // 앞 문자의 아스키 코드값이 크면 1 반환
    else if(*pa < *pb) return -1;              // 뒤 문자의 아스키 코드값이 크면 -1 반환
    else return 0;                            // 둘 다 널문자이므로 같은 문자열
}
```

---

# 문자열을 비교하는 STRCMP, STRNCMP 함수

---

## ▶ strcmp 함수가 문자의 아스키 코드값 비교

- ▶ 대소문자 섞인 경우는 반환값이 사전순서와 다를 수도
  - ▶ Ex) strcmp("apple", "Banana");
  - ▶ Banana가 사전의 뒤에 나오지만 1 반환
  - ▶ 대문자의 아스키 코드값이 소문자보다 작기 때문

## ▶ strcmp 함수의 반환값으로 사전 순서를 판단할 때는 반드시 대소문자를 일치시켜야

- ▶ 숫자나 특수문자, 한글의 경우 예외처리 필요