제11장

문자와 문자열

Perfect C

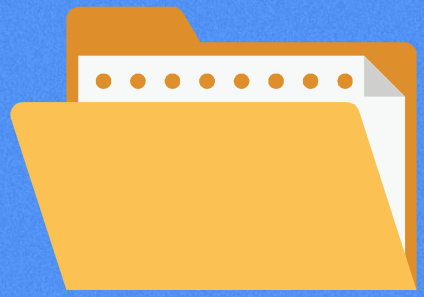
C언어로 배우는 프로그래밍 기초

단원 목표

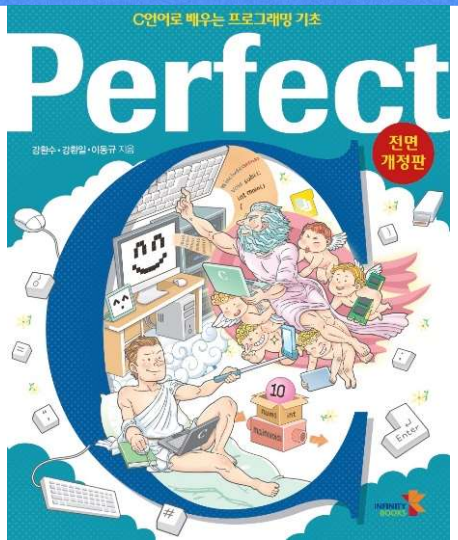
학습목표

- **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 처리하는 방법에 대해 이해하고 설명할 수 있다.**
  - 문자 포인터 배열 방법과 이차원 문자 배열 방법의 차이
  - 명령행 인자의 필요성과 구현 방법 이해

학습목차



01. 문자와 문자열



문자와 문자열의 개념

- **문자**
  - 영어의 알파벳이나 한글의 한 글자를 작은 따옴표로 둘러싸서 'A'와 같이 표기
    - C 언어에서 저장공간 크기 1바이트인 자료형 char로 지원
  - 작은 따옴표에 의해 표기된 문자를 문자 상수

```
char ch ='A';
```

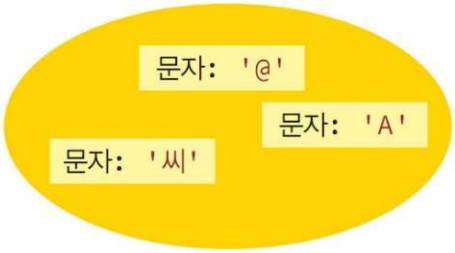


그림 11-1 문자 상수와 선언

- **문자열(string)**
  - 문자의 모임인 일련의 문자
    - 일련의 문자 앞 뒤로 큰 따옴표로 둘러싸서 "java"로 표기
  - 큰 따옴표에 의해 표기된 문자열을 문자열 상수
    - "A"처럼 문자 하나도 큰 따옴표로 둘러싸면 문자열 상수
    - 'ABC'처럼 작은 따옴표로 둘러싸도 문자가 될 수 없으며 오류가 발생

```
char c[] = "C language";
```

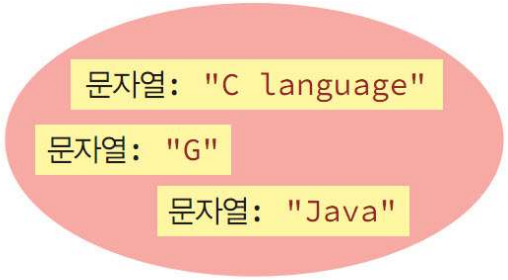


그림 11-2 문자열 상수와 선언

문자와 문자열의 선언

- **char형 변수에 문자를 저장**
  - 문자열을 저장하기 위한 자료형을 따로 제공하지 않음
  - 문자 배열을 선언하여 각각의 원소에 문자를 저장
  - 문자열의 마지막을 의미하는 NULL 문자 '₩0'가 마지막에 저장
  - 문자열이 저장되는 배열크기
    - 반드시 저장될 문자 수보다 1이 커야 널(NULL) 문자를 문자열의 마지막으로 인식
    - 문자열의 마지막에 널(NULL) 문자가 없다면 출력과 같은 문자열 처리에 문제가 발생
  - 배열 csharp의 크기를 3으로 선언한 후 배열 csharp에 문자열 "C#"을 저장
    - 마지막 원소인 csharp[2]에 '₩0'을 저장

```
char ch ='A';

char csharp[3];
csharp[0] = 'C'; csharp[1] = '#'; csharp[2] = '\o';
```

그림 11-3 문자와 문자열 저장

- **배열 선언 시 초기화 방법**
  - 중괄호를 사용
  - 문자 하나 하나를 쉼표로 구분하여 입력하고 마지막 문자로 널(NULL)인 '₩0'을 삽입

```
//문자 하나 하나 저장 시 마지막에 '\0' 문자 저장
char java[] = {'J','A','V','A', '\0'};
```

마지막에 '\0'을 빼면, 대입 시에는 문제가 없으나 출력 등에서 문제가 발생한다.

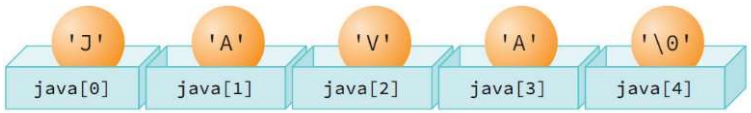


그림 11-4 문자열을 위한 문자 하나 하나의 초기화 선언

문자열을 선언하는 편리한 다른 방법

- **배열 선언 시 저장할 큰 따옴표를 사용해 문자열 상수를 바로 대입**
  - 배열 초기화 시 배열크기는 지정하지 않는 것이 더 편리
    - 지정한다면 마지막 문자인 '₩0'을 고려해 실제 문자 수보다 1이 더 크게 배열크기를 지정
    - 지정한 배열크기가 (문자수+1)보다 크면 나머지 부분은 모두 '₩0' 문자로 채워짐
  - 만일 배열크기가 작으면
    - 문자열 상수가 아닌 단순한 문자 배열이 되므로 문자열 출력 등에서 문제가 발생

그림 11-5 문자열 상수로 문자배열 초기화

문자와 문자열 출력

예제 chararray.c

- 함수 printf()를 사용한 문자와 문자열 출력

제어변수

- 함수 printf()
  - 형식제어문자 %c로 문자를 출력
  - 배열이름 또는 문자 포인터를 사용하여 형식제어문자 %s로 문자열을 출력
- 함수 puts(csharp)
  - 한줄에 문자열을 출력한 후 다음 줄에서 출력을 준비
- 함수 printf(c)
  - 배열이름을 인자로 사용해도 문자열 출력

실습예제 11-1 chararray.c

문자열 저장을 위한 문자열 배열 처리와 문자열 출력

```
01  // file: chararray.c
02  #include <stdio.h>
03
04  int main(void)
05  {
06     //문자 선언과 출력
07     char ch = 'A';
08     printf("%c %d\n", ch, ch);
09
10     //문자열 선언 방법1
11     char java[] = { 'J', 'A', 'V', 'A', '\0' };
12     printf("%s\n", java);
13     //문자열 선언 방법2
14     char c[] = "C language";    //크기를 생략하는 것이 간편
15     printf("%s\n", c);
16     //문자열 선언 방법3
17     char csharp[5] = "C#";
18     printf("%s\n", csharp);   <- 문자열 출력을 위해 배열이름과 형식제어문자 %s를 이용한다.
19
20     //문자 배열에서 문자 출력
21     printf("%c%c\n", csharp[0], csharp[1]);
22
23     return 0;
24  }
```

설명

```
07  char형 변수 ch 선언하면서 문자 'A' 저장
08  함수 printf()에서 %c로 문자 변수 ch 출력, %d를 사용하면 문자 코드값 출력
11  문자 배열을 선언하면서 문자 하나 하나를 초기화할 경우, 마지막에 '\0'을 삽입
12  문자열을 출력하려면 함수 printf()에서 %s로 문자배열 변수 이름 java를 기술
14  문자 배열을 선언하면서 문자열 상수를 초기화할 경우, 배열 크기를 생략하면 간편
15  문자열을 출력하려면 함수 printf()에서 %s로 문자배열 변수 이름 c를 기술
17  문자 배열을 선언하면서 문자열 상수를 초기화할 경우, 배열 크기를 지정한다면 문자열 상수의 문자
    수보다 1이 더 크게 지정하며, 문자 배열에서 지정된 이후 공간은 나머지는 모두 '\0'가 자동으로 저장
18  문자열을 출력하려면 함수 printf()에서 %s로 문자배열 변수 이름 csharp를 기술
21  문자열이 저장된 문자배열에서 %c로 문자배열 원소를 지정하면 문자 하나를 출력 가능
```

실행결과

```
A 65
JAVA
C language
C#
C#
```

문자열 구성하는 문자 참조

- **문자열을 처리하는 다른 방법**
  - 문자열 상수를 문자 포인터에 저장하는 방식
  - 문자 포인터 변수에 문자열 상수를 저장
  - 문자열 출력도 함수 printf()에서 포인터 변수와 형식제어문자 %s
  - 문자 포인터에 의한 선언으로는 문자 하나 하나의 수정은 불가능

```
int i = 0;
char *java = "java";
printf("%s ", java);
```

```
while (java[i] != '\0')
   printf("%c", java[i++]);
printf("\n");
```

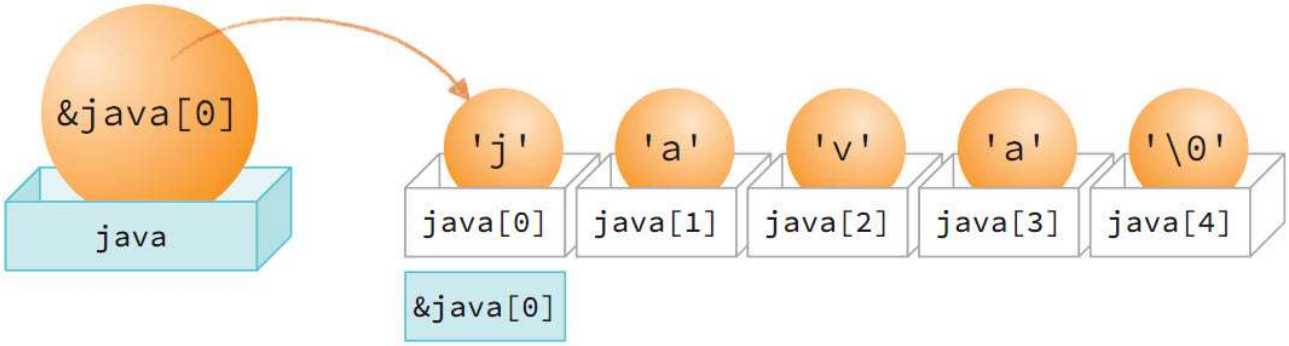


그림 11-6 문자 포인터를 사용한 문자열 처리

문자 포인터

예제 charpointer.c

- 변수의 값과 주소 값의 대입과 활용

문자 포인터 변수

- 문자열을 구성하는 하나 하나의 문자를 배열형식으로 직접 참조하여 출력
- 변수 java를 사용하여 문자를 수정하거나 수정될 수 있는 함수의 인자로 사용
  - 실행 오류가 발생
  - 변수 java가 가리키는 문자열은 상수이므로 수정은 불가능
- 출력할 문자열의 끝을 '₩0' 문자로 검사하면 편리
- 반복문을 이용하여 문자가 '₩0'이 아니면 문자를 출력

```
10    int i = 0;
11    while (java[i]) //while (java[i] != '\0')
12       printf("%c", java[i++]);
13    printf(" ");
14
15    i = 0;
16    while (*(java + i) != '\0') //java[i]는 *(java + i)와 같음
17       printf("%c", *(java + i++));
18    printf("\n");
19
20    //수정 불가능, 실행오류 발생
21    java[0] = 'J';
22
23    return 0;
24 }
```

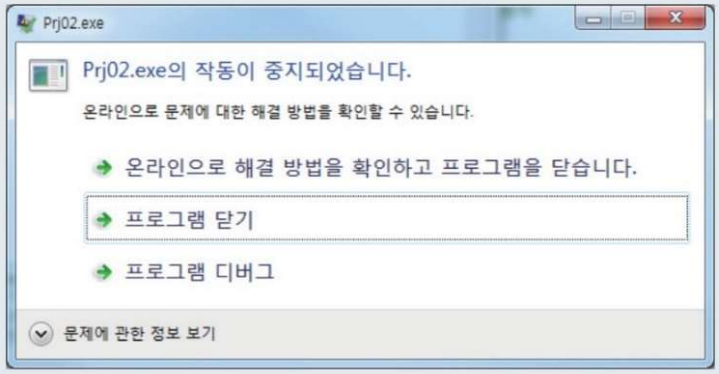
java[i]는 *(java + i)와 동일한 표현 방식이므로 java[i++]도 *(java + i++)와 같다.

설명

```
06    char 형 포인터 변수 java를 선언하면서 문자열 'java'의 첫 주소를 저장하므로,
      변수 java는 문자열 상수의 첫 주소를 가리키는 상황
10    반복문의 첨자로 사용될 변수 i를 선언하면서 0으로 지정
11    포인터 변수 java의 위치를 하나 하나 증가시키면서 널(NULL)이 아니면 반복을 실시
12    java가 가리키는 문자를 출력하고 i를 하나 증가시킴
11~12 이 반복문으로 java가 가리키던 문자열을 모두 출력
15    다시 반복문의 첨자로 사용될 변수 i에 0을 지정
16    포인터 변수 java의 위치를 하나 하나 증가시키면서 널(NULL)이 아니면 반복을 실시,
      *(java + i)는 java[i]와 같은 식임
17    java가 가리키는 문자를 출력하고 i를 하나 증가시킴
16~17 이 반복문으로 java가 가리키던 문자열을 모두 출력
21    java[i]는 수정할 수 없음
```

실행결과

```
java java java
```



실습예제 11-2 charpointer.c

문자 포인터로 문자열 처리

```
01 // file: charpointer.c
02 #include <stdio.h>
03
04 int main(void)
05 {
06    char *java = "java";
07    printf("%s ", java);
08
09    //문자 포인터가 가리키는 문자 이후를 하나 하나 출력
```

'₩0' 문자에 의한 문자열 분리

예제 string.c

- 문자배열로 문자열을 처리

문자 포인터 변수

- 함수 printf()
  - %s는 문자 포인터가 가리키는 위치에서 NULL 문자까지를 하나의 문자열로 인식
- 배열 c[]
  - 처음에 문자열 "C C++ Java"가 저장되고 마지막에 NULL 문자가 저장
- 만일 배열 c에 문장 c[5] = '₩0';을 실행하고 c를 출력하면 무엇이 출력될까?
  - c[5]에 저장된 '₩0' 문자에 의해 c가 가리키는 문자열은 "C C++"까지
  - 즉 문자열은 시작문자부터 '₩0' 문자가 나올 때까지 하나의 문자열로 처리
- (c+6)로 문자열을 출력
  - "Java" 출력

실습예제 11-3 string.c

문자 포인터로 문자열 처리

```
01  // file: string.c
02  #include <stdio.h>
03
04  int main(void)
05  {
06     char c[] = "C C++ Java";
07     printf("%s\n", c);
08     c[5] = '\0'; // NULL 문자에 의해 문자열 분리
09     printf("%s\n%s\n", c, (c + 6));
10
11     //문자 배열의 각 원소를 하나 하나 출력하는 방법
12     c[5] = ' '; //널 문자를 빈 문자로 바꾸어 문자열 복원
13     char *p = c;
14     while (*p) //(*p != '\0')도 가능
15        printf("%c", *p++);
16     printf("\n");
17
18     return 0;
19  }
```

```
int i = 0;
while  (c[i])
   printf("%c", c[i++]);
printf("\n");

i = 0;
while  (*(c+i))
   printf("%c", *(c + i++));
printf("\n");
```

설명

```
06  char 형 배열변수 c 선언하면서 문자열 "C C++ Java" 저장
07  함수 printf()에서 %s로 문자배열 c 출력, 문자열 "C C++ Java"이 출력
08  c[5]에 '\0'를 저장하면 "C C++\0Java"이 저장되어 두 개의 문자열 "C C++"와 "Java"로 나뉨
09  문자 포인터 c, (c + 6)는 각각 두 개의 문자열 "C C++"와 "Java"이 시작하는 위치이므로
    각각의 문자열이 출력
12  다시 c[5]에 ' '를 저장하면 "C C++ Java"이 저장되어 한 개의 문자열 "C C++ Java"이 복원됨
13  char 포인터 변수 p를 선언하면서 c의 첫 주소를 저장
14  문자 포인터 p가 가리키는 값이 널이 아니면 while 반복 실행
15  문자 포인터 p가 가리키는 값의 문자를 출력한 후 p의 주소값을 1 증가하여 다음 문자를 가리키도록
14~15 while 반복으로 p가 가리키는 문자열을 출력
```

실행결과

```
C C++ Java
C C++
Java
C C++ Java
```

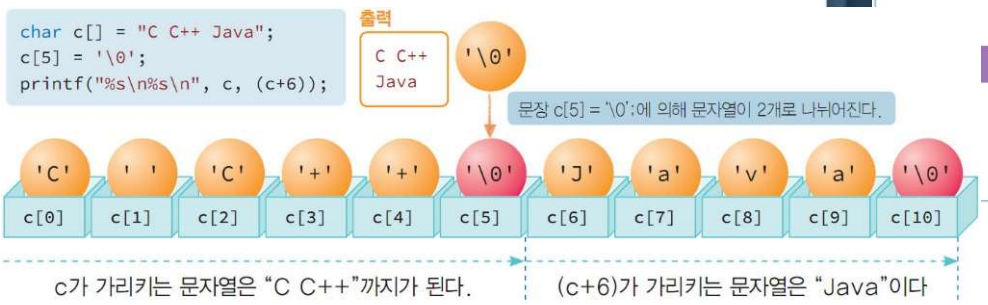


그림 11-7 문자열은 문자열이 시작되는 부분부터 '\0'문자까지

버퍼처리 함수 getchar()

- **함수 getchar()**
  - 문자의 입력에 사용
  - 라인 버퍼링(line buffering) 방식을 사용
    - 문자 하나를 입력해도 반응을 보이지 않다가
    - [enter] 키를 누르면 그제서야 이전에 입력한 문자마다 입력이 실행
    - 입력한 문자는 임시 저장소인 버퍼(buffer)에 저장되었다가
    - [enter] 키를 만나면 함수는 버퍼에서 문자를 읽기 시작
  - 즉각적(interactive)인 입력을 요구하는 시스템에서는 사용이 불가능

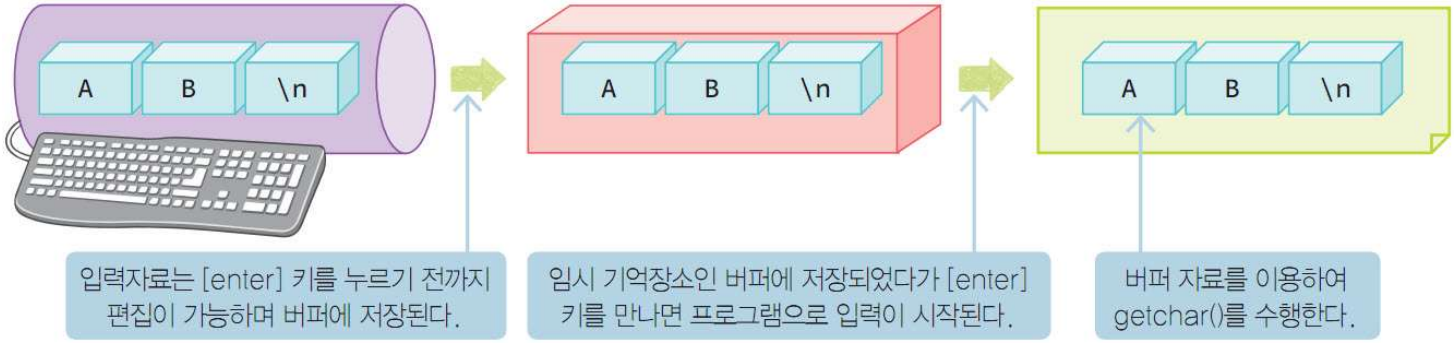


그림 11-8 함수 getchar()의 처리 과정

함수 getche()

- **버퍼를 사용하지 않고 문자를 입력하는 함수 getche()**
  - getche()는 버퍼를 사용하지 않으므로
    - 문자 하나를 입력하면 바로 함수 getche()가 실행
  - 함수 getche()에서 입력된 문자는 바로 모니터에 표시
  - 함수를 이용하려면 헤더파일 conio.h를 삽입
  - 입력 문자가 'q'가 아니면 함수 putchar()에 의하여 문자가 바로 출력
    - 함수 getche()에 의하여 입력된 문자도 보이고
    - 바로 putchar()에 의하여 출력
  - 입력문자가 "inputq"
    - 화면에는 "iinnppuuttq" 표시
    - 화면에 보이는 행이 표준입력과 표준출력이 번갈아 가면서 나오게 되므로 한 문자가 두 번씩 표시

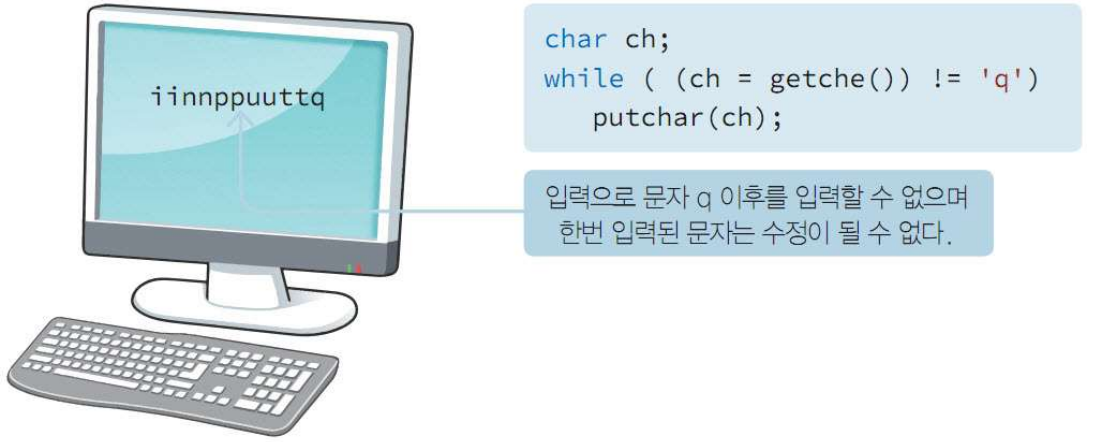
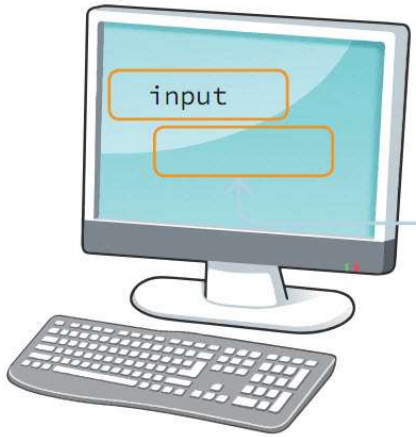


그림 11-9 함수 getche()와 putchar()의 이용

함수 getch()

- **문자 입력을 위한 함수 getch()**
  - 입력한 문자가 화면에 보이지 않는 특성
  - 입력된 문자를 출력함수로 따로 출력하지 않으면 입력 문자가 화면에 보이지(echo) 않음
  - 버퍼를 사용하지 않는 문자 입력 함수
  - conio.h를 삽입
- **위 소스에서 문자 "inputq"를 입력으로 실행**
  - "input"이 출력
  - 함수 putch(ch)
    - 인자를 출력하는 함수




```
char ch;
while ( (ch = getche()) != 'q')
   putchar(ch);
```

마지막에 입력한 문자 q 이전까지 문자 하나씩 출력되며, 입력한 문자는 수정할 수 없다. 마지막에 입력한 문자 'q'는 함수 putch()를 실행하지 않으므로 출력되지 않는다.

그림 11-10 함수 getch()와 putch()의 이용

표 11-1 문자입력 함수 scanf(),getchar(), getche(), getch()의 비교

함수	scanf("%c", &ch)	getchar()	getche() _getche()	getch() _getch()
헤더파일	stdio.h		conio.h	
버퍼 이용	버퍼 이용함		버퍼 이용 안함	
반응	[enter] 키를 눌러야 작동		문자 입력마다 반응	
입력 문자의 표시(echo)	누르면 바로 표시		누르면 바로 표시	표시 안됨
입력문자 수정	가능		불가능	

다양한 문자 입력

예제 getche.c

- 세 개의 while 문의 입력으로 각각 "getchar()q", "getche()q", "getch()q"를 입력

실습예제 11-4 getche.c

함수getchar(), _getche(), _getch()의 차이를 알아보는 예제

```
01  // file: getche.c
    #include <stdio.h>
    #include <conio.h>

    int main(void)
    {
       char ch;

       printf("문자를 계속 입력하고 Enter를 누르면 >>\n");
       while ((ch = getchar()) != 'q')
          putchar(ch);

       printf("\n문자를 누를 때마다 두 번 출력 >>\n");
       while ((ch = _getche()) != 'q')
          putchar(ch);

       printf("\n문자를 누르면 한 번 출력 >>\n");
       while ((ch = _getch()) != 'q')
          _putch(ch);
       printf("\n");

       return 0;
23  }
```

설명

```
07  입력한 문자를 저장할 공간으로 char형 변수 ch 선언
09  입력 프롬프트 출력
10  함수 getchar()로 입력한 문자를 변수 ch에 저장하여 'q'가 아니면 반복몸체인 putchar(ch)를
    실행, 변수 ch에 'q'이면 반복 종료라 출력이 안됨
13  입력 프롬프트 출력
14  함수 _getche()로 입력한 문자를 변수 ch에 저장하여 'q'가 아니면 반복몸체인 putchar(ch)를
    실행, 변수 ch에 'q'이면 반복 종료라 출력이 안됨. 함수 _getche()는 입력한 문자마다 바로 처리
17  입력 프롬프트 출력
18  함수 _getch()로 입력한 문자를 변수 ch에 저장하여 'q'가 아니면 반복몸체인 _putch(ch)를
    실행, 변수 ch에 'q'이면 반복 종료라 출력이 안됨. 함수 _getch()는 입력한 문자마다
    바로 처리하며, 입력한 문자도 보이지 않음
```

실행결과

```
문자를 계속 입력하고 Enter를 누르면 >>
java
java
python
python
q                    마지막에 입력한 문자 q가 입력 종료를 처리

문자를 누를 때마다 두 번 출력 >>
jjaavvaaq
문자를 누르면 한 번 출력 >>
java
```

문자배열 변수로 scanf()에서 입력

예제 stringput.c

- 문자열 입력

문자 포인터 변수

- 함수 scanf()는 공백으로 구분되는 하나의 문자열을 입력
  - 함수 scanf("%s", str)에서 형식제어문자 %s를 사용
  - 문자열 입력은 충분한 공간의 문자배열이 있어야 가능
  - 단순히 문자 포인터로는 문자열 저장이 불가능
- 가장 먼저 입력 받은 문자열이 저장될 충분한 공간인 문자 배열 str을 선언
- 함수 printf("%s", str)에서 %s를 사용하여 문자열을 출력
- 이름과 성을 분리하여 입력한다면 성만 name[]에 저장
- 함수 printf()에서 %10s는 폭이 10, 우측정렬로 문자열을 출력

실습예제 11-5 strinput.c

함수scanf()와 printf()에서 문자열 입출력

```
01  // file: strinput.c
02  #define _CRT_SECURE_NO_WARNINGS
03  #include <stdio.h>
04
05  int main(void)
06  {
07     char name[20], dept[30]; //char *name, *dept; 실행 오류 발생
08
09     printf("%s", "학과 입력 >> ");
10     scanf("%s", dept);
11     printf("%s", "이름 입력 >> ");
12     scanf("%s", name);
13     printf("출력: %10s %10s\n", dept, name);
14
15     return 0;
16  }
```

설명

```
07  이름과 학과가 저장될 공간인 문자배열 name[20]과 dept[30]을 선언, 배열크기 20, 30은
    이름과 학과가 저장될 충분한 크기여야 하며, 특히 한글은 크기 2개에 하나의 글자가 입력됨
09  학과 입력 프롬프트 출력
10  함수 scanf("%s", dept)와 같이 배열 dept에 표준입력 문자열 저장
11  이름 입력 프롬프트 출력
10  함수 scanf("%s", name)과 같이 배열 name에 표준입력 문자열 저장
13  입력된 학과와 이름을 출력
```

실행결과

```
학과 입력 >> 컴퓨터정보공학과
이름 입력 >> 김미림
출력:  컴퓨터정보공학과       김미림
```

gets()

- **함수 gets(): 한 행의 문자열 입력**
  - 헤더파일 stdio.h 를 삽입
  - 함수 gets()는 [enter] 키를 누를 때까지 한 행을 버퍼에 저장 한 후 입력처리
    - 마지막에 입력된 '₩n'가 '₩0'로 교체되어 인자인 배열에 저장
    - 한 행을 하나의 문자열로 간주하고 프로그래밍할 수 있도록

문자열 입출력 함수: 헤더파일 stdio.h 삽입

```
char * gets(char * buffer);
```

- 함수 gets()는 문자열을 입력 받아 buffer에 저장하고 입력 받은 첫 문자의 주소값을 반환한다.
- 함수 gets()는 표준입력으로 [enter] 키를 누를 때까지 공백을 포함한 한 행의 모든 문자열을 입력 받는다.
- 입력된 문자열에서 마지막 [enter] 키를 '\0' 문자로 대체하여 저장한다.

```
char * gets_s(char * buffer, size_t sizebuffer);
```

- 두 번째 인자인 sizebuffer는 정수형으로 buffer의 크기를 입력한다.
- Visual C++에서는 앞으로 gets() 대신 함수 gets_s()의 사용을 권장한다.

```
int puts(const char * str);
```

- 인자인 문자열 str에서 마지막 '\0' 문자를 개행 문자인 '\n'로 대체하여 출력한다.
- 함수 puts()는 일반적으로 0인 정수를 반환하는데, 오류가 발생하면 EOF를 반환한다.

그림 11-11 문자열 입출력 함수

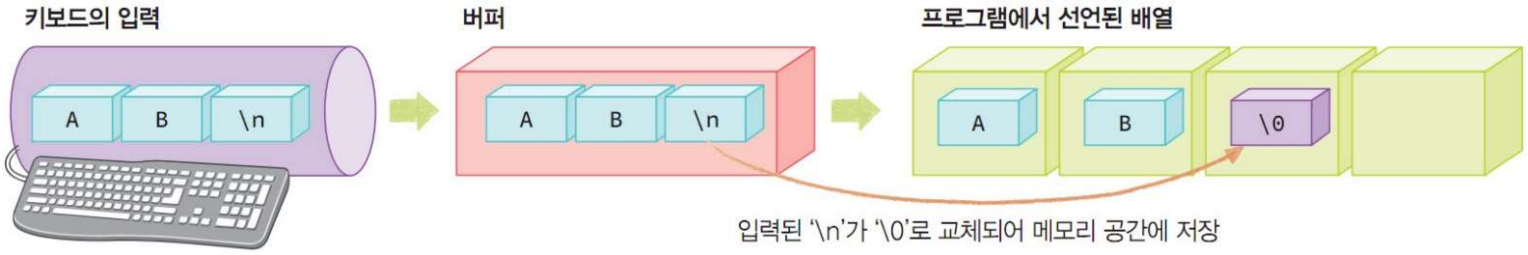


그림 11-12 함수 gets()의 처리 과정

puts()

- **함수 puts(): 한 행에 문자열을 출력**
  - 헤더파일 stdio.h 를 삽입
  - 오류가 발생하면 EOF를 반환
    - 기호 상수 EOF(End Of File)
    - 파일의 끝이라는 의미로 stdio.h 헤더파일에 정수 -1로 정의
      - #define EOF (-1)
  - 함수 gets()와 반대로 문자열의 마지막에 저장된 '₩0'를 '₩n'로 교체하여 버퍼에 전송
  - 버퍼의 내용이 모니터에 출력되면 문자열이 한 행에 출력

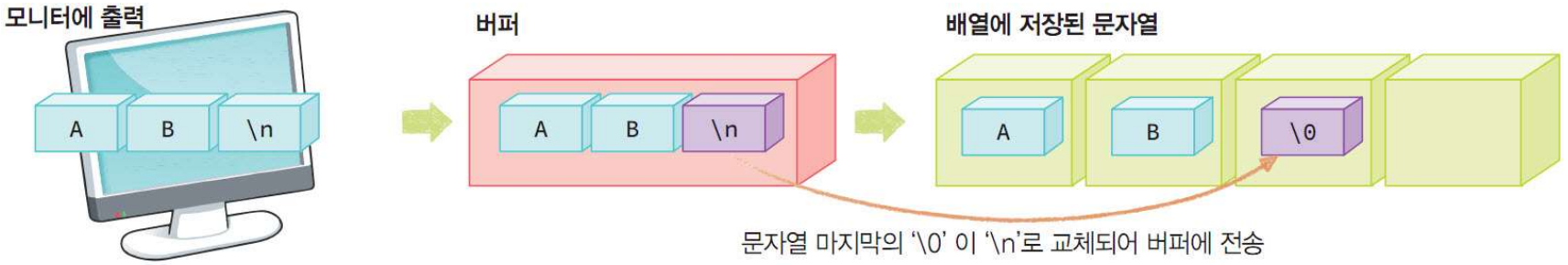


그림 11-13 함수 puts()의 처리 과정

한 행 입력과 출력

예제 gets.c

- 함수 gets()와 gets_s()를 사용하여 여러 줄을 입력 받아 출력

문자 포인터 변수

- while문을 사용하면 연속된 여러 행을 입력 받아 바로 행별로 출력
  - 다음 반복을 종료하려면 새로운 행 처음에 (ctrl + Z)를 입력
- 함수 printf()와 scanf()
  - 다양한 입출력에 적합
- 함수 puts()와 gets()
  - 처리 속도가 빠르다는 장점

실습예제 11-6 gets.c

함수gets()와 puts() 기능을 알아보는 예제

```
01  // file: gets.c
02  #define _CRT_SECURE_NO_WARNINGS
03  #include <stdio.h>
04
05  int main(void)
06  {
07     char line[101]; //char *line 으로는 오류발생
08
09     printf("입력을 종료하려면 새로운 행에서 (ctrl + Z)를 누르십시오.\n");
10     while (gets(line))
11        puts(line);
12     printf("\n");
13
14     while (gets_s(line, 101))
15        puts(line);
16     printf("\n");
17
18     return 0;
19  }
```

설명

```
07  한 줄에 입력되는 모든 문자열이 입력되도록 충분한 크기의 문자배열 line[101] 선언
09  입력 프롬프트 출력
10  함수 gets(line) 호출로 한 줄 전체를 입력 받음, ctrl + Z 입력하면 종료
11  함수 puts(line) 호출로 한 줄 전체를 출력
14  함수 gets_s(line, 101) 호출로 한 줄 전체를 입력 받음, ctrl + Z 입력하면 종료
15  함수 puts(line) 호출로 한 줄 전체를 출력
```

실행결과

```
입력을 종료하려면 새로운 행에서 (ctrl + Z)를 누르십시오.
문자열 처리를 배우고 있습니다.
문자열 처리를 배우고 있습니다.
^Z

gets()의 사용도 마찬가지입니다.
gets()의 사용도 마찬가지입니다.
^Z
```

LAB 한 행을 표준입력으로 받아 문자 하나 하나를 그대로 출력

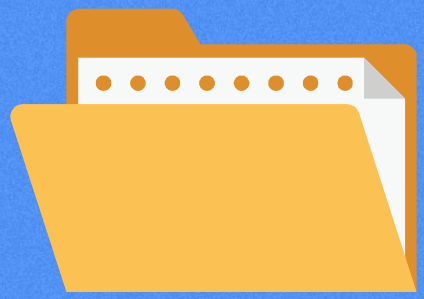
- **함수 gets()를 사용하여 한 행의 표준입력을 받아 배열 s에 저장한 후,**
  - 문자 포인터를 사용해서 이배열 s에서 문자 하나 하나를 이동하면서 출력
  - char 변수 p를 선언하면서 배열 s의 첫 원소를 가리키도록 저장
  - 포인터 변수 p는 주소값이며 *p는 p가 가리키는 곳의 문자
- **결과**
  - int main(void)
  - int main(void)

Lab 11-1 lineprint.c

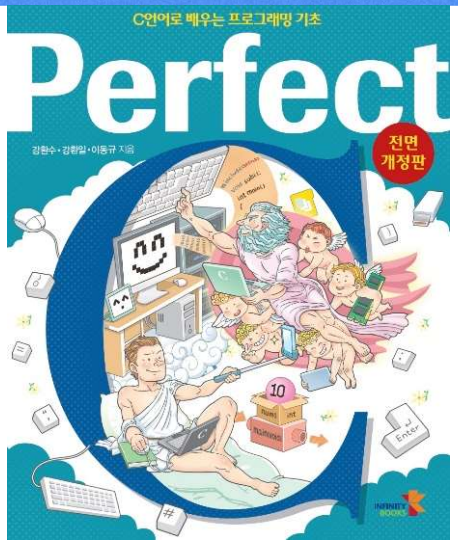
```
01  // lineprint.c:
02  #define _CRT_SECURE_NO_WARNINGS
03  #include <stdio.h>
04
05  int main()
06  {
07     char s[100];
08     //문자배열 s에 표준입력한 한 행을 저장
09     gets(s);
10
11     //문자배열에 저장된 한 행을 출력
12     char *p = ______________;
13     while (_________________)
14        printf("%c", ______________________);
15     printf("\n");
16
17     return 0;
18  }
```

정답

```
12  char *p = s;
13  while (*p)
14     printf("%c", *p++);
```



02. 문자열 관련 함수



다양한 문자열 라이브러리 함수

- **헤더파일 string.h에 함수원형으로 선언된 라이브러리 함수로 제공**
  - 문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리
  - 문자의 배열 관련 함수
  - 자료형 size_t
    - 비부호 정수형(unsigned int type)
  - 자료형 void *
    - 아직 정해지지 않은 다양한 포인터를 의미

표 11-1 문자열 배열에 관한 다양한 함수

함수원형	설명
`void *memchr(const void *str, int c, size_t n)`	메모리 str에서 n 바이트까지 문자 c를 찾아 그 위치를 반환
`int memcmp(const void *str1, const void *str2, size_t n)`	메모리 str1과 str2를 첫 n 바이트를 비교 검색하여 같으면 0, 다르면 음수 또는 양수 반환
`void *memcpy(void *dest, const void *src, size_t n)`	포인터 src 위치에서 dest에 n 바이트를 복사한 후 dest 위치 반환
`void *memmove(void *dest, const void *src, size_t n)`	포인터 src 위치에서 dest에 n 바이트를 복사한 후 dest 위치 반환
`void *memset(void *str, int c, size_t n)`	포인터 src 위치에서부터 n 바이트까지 문자 c를 지정한 후 src 위치 반환
`size_t strlen(const char *str)`	포인터 src 위치에서부터 널 문자를 제외한 문자열의 길이 반환

문자열 관련 함수 활용

예제 memfun.c

- 문자열의 길이를 반환하는 함수 strlen()
- 문자배열의 복사를 위한 함수 memcpy(),
- 문자배열에서 문자 이후의 문자열을 찾는 함수 memchr()을 알아보는 예제

실습예제 11-7 memfun.c

문자배열에 관한 함수

```c
// file: memfun.c
#include <stdio.h>
#include <string.h>

int main(void)
{
    char src[50] = "https://www.visualstudio.com";
    char dst[50];

    printf("문자배열 src = %s\n", src);
    printf("문자열크기 strlen(src) = %d\n", strlen(src));
    memcpy(dst, src, strlen(src) + 1);
    printf("문자배열 dst = %s\n", dst);
    memcpy(src, "안녕하세요!", strlen("안녕하세요!")+1);
    printf("문자배열 src = %s\n", src);

    char ch = ':';
    char *ret;
    ret = memchr(dst, ch, strlen(dst));
    printf("문자 %c 뒤에는 문자열 %s  이 있다.\n", ch, ret);

    return 0;
}
```

설명

```
char형 배열 변수 src[50]을 선언하면서 문자열 https://www.visualstudio.com 저장
char형 배열 변수 dst[50]을 선언
함수 printf()에서 %s로 문자배열 src 출력
함수 printf()에서 strlen(src)으로 문자배열 src에 저장된 문자열 길이(28) 출력
문자열 src에서 포인터 위치 dst에 strlen(src) + 1 수만큼 복사하여 문자열의
마지막이 널이 되도록
함수 printf()에서 %s로 문자배열 dst 출력
포인터 위치 src에 문자열 상수 "안녕하세요!"를 복사하기 위해서는 (strlen("안녕하세요!") + 1)
만큼 복사해야 문자열의 마지막이 널이 됨
함수 printf()에서 %s로 문자배열 src 출력
구분자인 문자 ':'를 변수 ch에 저장
문자 포인터 변수 ret 선언
함수 memchr(dst, ch, strlen(dst)) 호출로 문자열 dst에서 문자 ':' 이후의 문자열을
반환하여 변수 ret에 저장
함수 printf()에서 %c와 %s로 문자 ch와 문자열 ret 출력
```

실행결과

```
문자배열 src = https://www.visualstudio.com
문자열크기 strlen(src) = 28
문자배열 dst = https://www.visualstudio.com
문자배열 src = 안녕하세요!
문자 : 뒤에는 문자열 ://www.visualstudio.com  이 있다.
```

함수 strcmp()

- **문자열 관련 함수는 대부분 strxxx()로 명명**
  - 문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리
  - 헤더파일 string.h에 함수원형으로 선언된 라이브러리 함수로 제공

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 0을 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 n까지만 비교하여 같으면 0을 반환하고, 앞이 크면 양수를, 뒤가 크면 음수를 반환한다.

그림 11-14 문자열 비교 함수

함수 strcmp()

예제 strcmp.c

- 함수 strcmp()
  - 인자인 두 문자열을 사전(lexicographically) 상의 순서로 비교하는 함수
- 함수 strncmp()
  - 두 문자를 비교할 문자의 최대 수를 지정하는 함수
  - 비교 기준은 아스키 코드값
  - 두 문자가 같다면 계속 다음 문자를 비교하여 문자가 다를 때까지 계속 비교
  - 비교 앞 문자가 작으면 음수, 뒤 문자가 작으면 양수, 같으면 0을 반환

실습예제 11-8 strcmp.c

문자열 비교함수 strcmp()와 strncmp() 사용

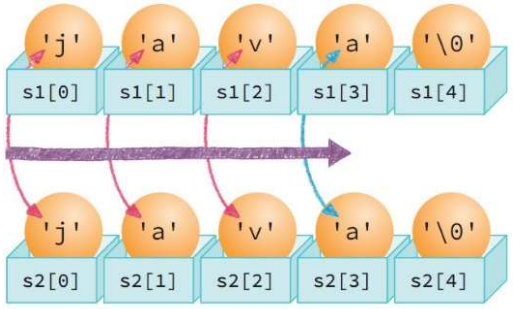
```
01  // file: strcmp.c
02  #include <stdio.h>
03  #include <string.h>
04
05  int main(void)
06  {
07     char *s1 = "java";
08     char *s2 = "java";
09     printf("strcmp(%s, %s) = %d\n", s1, s2, strcmp(s1, s2));
10
11     s1 = "java";
12     s2 = "jav";
13     printf("strcmp(%s, %s) = %d\n", s1, s2, strcmp(s1, s2));
14     s1 = "jav";
15     s2 = "java";
16     printf("strcmp(%s, %s) = %d\n", s1, s2, strcmp(s1, s2));
17     printf("strncmp(%s, %s, %d) = %d\n", s1, s2, 3, strncmp(s1, s2, 3));
18
19     return 0;
20  }
```

설명

```
09  s1과 s2가 모두 "java"로 같으므로 함수호출 strcmp(s1, s2)는 0임
13  s1은 "java", s2는 "jav"이므로 함수호출 strcmp(s1, s2)는 양수임
16  s1은 "jav", s2는 "java"이므로 함수호출 strcmp(s1, s2)는 음수임
17  s1은 "jav", s2는 "java"이므로 함수호출 strcmp(s1, s2, 3)는 "jav"까지 비교하므로 0임
```

실행결과

```
strcmp(java, java) = 0
strcmp(java, jav) = 1
strcmp(jav, java) = -1
strncmp(jav, java, 3) = 0
```



- 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

그림 11-15 함수 strcmp()와 strncmp()의 비교 방법

함수 strcpy()

- **함수 strcpy()와 strncpy()**
  - 문자열을 복사하는 함수
  - 함수 strcpy()는 앞 인자 문자열 dest에 뒤 인자 문자열 source를 복사
  - 첫 번째 인자인 dest는 복사 결과가 저장될 수 있도록 충분한 공간을 확보
  - 함수 strncpy()는 복사되는 최대 문자 수를 마지막 인자 maxn으로 지정하는 함수

문자열 복사 함수

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

- 앞 문자열 dest에 처음에 뒤 문자열 null 문자를 포함한 source 를 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 source에서 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 dset의 크기를 입력한다.
- 반환형 errno_t는 정수형이며 반환값은 오류번호로 성공하면 0을 반환한다.
- Visual C++에서는 앞으로 함수strcpy_s()와 strncpy_s()의 사용을 권장한다.

그림 11-16 문자열 복사 함수

함수 strcpy()

예제 strcpy.c

- 문자열 복사 사용
- 항상 문자열은 마지막 NULL 문자까지 포함하므로 다음 부분 소스에서 문자배열 d에는 NULL 문자까지 복사

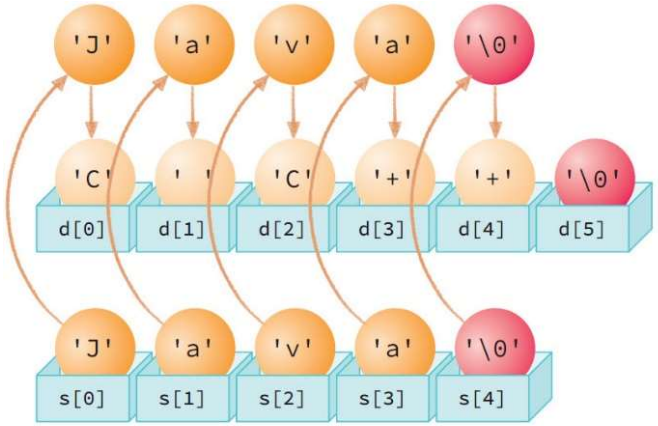


그림 11-17 함수 strcpy()의 기능

실습예제 11-9 strcpy.c

문자열 복사 함수 strcpy()와 strncpy() 사용

```
01  // file: strcpy.c
02  #define _CRT_SECURE_NO_WARNINGS
03  #include <stdio.h>
04  #include <string.h>
05
06  int main(void)
07  {
08     char dest[80] = "Java";
09     char source[80] = "C is a language.";
10
11     printf("%s\n", strcpy(dest, source));
12     //printf("%d\n", strcpy_s(dest, 80, source));   // 함수 strcpy_s()의 사용
13     //printf("%s\n", dest);
14     printf("%s\n", strncpy(dest, "C#", 2));
15
16     printf("%s\n", strncpy(dest, "C#", 3));
17     //printf("%d\n", strncpy_s(dest, 80, "C#", 3));   // 함수 strncpy_s()의 사용
18     //printf("%s\n", dest);
19
20     return 0;
21  }
```

11 함수호출 strcpy(dest, source)의 결과는 문자열 source 모두를 dest에 복사한 후, 반환값은 문자 포인터 dest 임

14 함수호출 strncpy(dest, "C#", 2)의 결과는 "C#"을 dest에 2바이트만 복사한 후, 반환값은 문자 포인터 dest 임

16 함수호출 strncpy(dest, "C#", 3)의 결과는 "C#"을 dest에 3바이트까지 복사하므로 마지막은 널 문자가 복사되며, 반환값은 문자 포인터 dest 임

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

"C#"까지 2개의 문자가 복사되므로 앞 2 문자가 문자열 "C#"으로 대체되고 나머지도 모두 출력된다.

"C#\0"까지 3개의 문자가 복사되므로 문자열 "C#"이 출력된다.

결과는 d에도 "java"가 저장된다.

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

함수 strcat()

- **하나의 문자열 뒤에 다른 하나의 문자열을 연이어 추가해 연결**
  - 앞 문자열에 뒤 문자열의 null 문자까지 연결
    - 앞의 문자열 주소를 반환
  - 앞 인자인 dest의 저장공간이 연결된 문자열의 길이보다 부족하면 문제가 발생
- **strncat() 함수**
  - 전달인자의 마지막에 연결되는 문자의 수를 지정
  - 마지막 수는 널 문자를 제외한 수

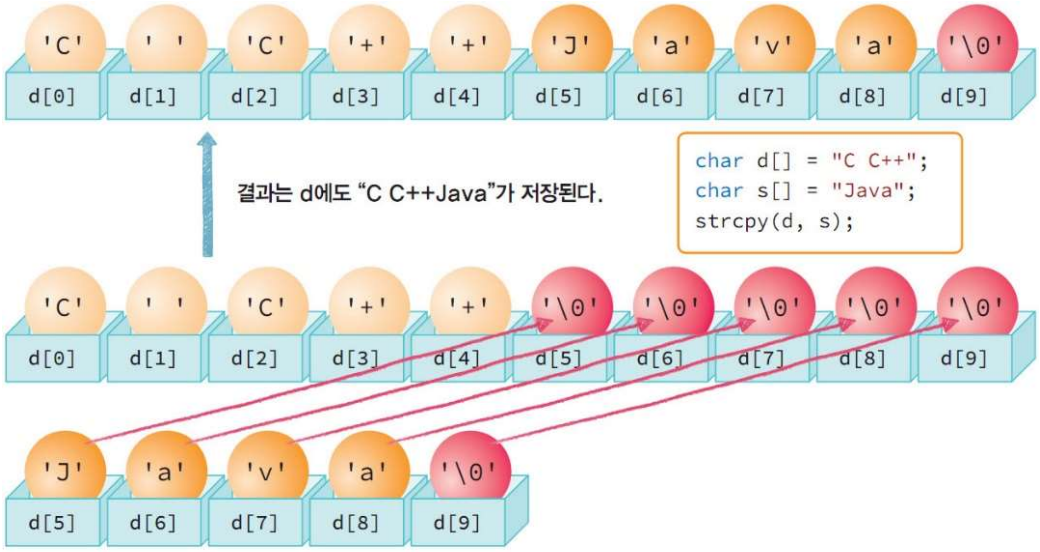


그림 11-19 함수 strcat()의 기능

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.

그림 11-18 문자열 연결 함수

함수 strcat()

예제 strcat.c

- 문자열 연결 함수 사용

문자 포인터 변수

- 함수 strcpy(), strcat()를 이용 시
  - 첫 인자로 문자열 포인터변수는 사용 불가능
  - 첫 번째 인자인 dest는 복사 또는 연결 결과가 저장될 수 있도록 충분한 공간을 확보
  - 문자열 관련 함수에서 단순히 문자열 포인터를 수정이 가능한 문자열의 인자로 사용 불가능
- 다음은 모두 오류 발생

```
char dest[5] = "C";
char *destc = "C";

strcpy(dest, "Java language");
strcpy(destc, " Java language");
strcat(dest, " is a language.");
strcat(destc, " is a language.");
```

실습예제 11-10 strcat.c

문자열 연결 함수 사용

```
01  // file: strcat.c
02  #define _CRT_SECURE_NO_WARNINGS
03  #include <stdio.h>
04  #include <string.h>
05
06  int main(void)
07  {
08     char dest[80] = "C";
09
10     printf("%s\n", strcat(dest, " is "));
11     //printf("%d\n", strcat_s(dest, 80, " is "));
12     //printf("%s\n", dest);
13     printf("%s\n", strncat(dest, "a java", 2));
14     //printf("%d\n", strncat_s(dest, 80, "a proce", 2));
15     //printf("%s\n", dest);
16     printf("%s\n", strcat(dest, "procedural "));
17     printf("%s\n", strcat(dest, "language."));
18
19     return 0;
20  }
```

함수 strcat_s()의 사용

함수 strncat_s()의 사용

2개 문자인 "a "까지만 뒤에 연결된다.

설명

```
10  함수호출 strcat(dest, " is ")로 문자열 dest는 "C is "가 됨
13  함수호출 strncat(dest, "a java", 2)로 문자열 dest는 "C is a "가 됨
16  함수호출 strcat(dest, "procedural ")로 문자열 dest는 "C is a procedural "이 됨
17  함수호출 strcat(dest, "language.")로 문자열 dest는 "C is a procedural language."
    가 됨
```

실행결과

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

함수 strtok()

- **문자열에서 구분자(delimiter)인 문자를 여러 개 지정하여 토큰을 추출하는 함수**
  - 첫 번째 인자인 str은 토큰을 추출할 대상인 문자열
  - 두 번째 인자인 delim은 구분자로 문자의 모임인 문자열
  - 첫 번째 인자인 str은 문자배열에 저장된 문자열을 사용
    - str은 문자열 상수를 사용 불가능

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.

그림 11-20 문자열 추출 함수 strtok()

```
문자열: "C and C++\t language are best!"
```

- 구분자 delim이 " "인 경우의 토큰: C, and, C++\t, language, are, best! 총 6개
- 구분자 delim이 " \t"인 경우의 토큰: C, and, C++, language, are, best! 총 6개
- 구분자 delim이 " \t!"인 경우의 토큰: C, and, C++, language, are, best 총 6개

그림 11-21 토큰 분리 예

함수 strtok()

예제 strtok.c

- 문자열 분리 함수 사용

함수 strtok()의 사용방법

- 문장 ptoken = strtok(str, delimiter);으로 첫 토큰을 추출
  - 결과를 저장한 ptoken이 NULL이면 더 이상 분리할 토큰이 없는 경우
  - 계속 토큰을 추출하려면 while 반복으로 추출된 토큰이 있는지를 (ptoken != NULL) 로 검사
  - NULL을 첫 번째 인자로 다시 strtok(NULL, delimiter)를 호출하면 그 다음 토큰을 반환

실습예제 11-11 strtok.c

문자열에서 지정한 분리자를 사용하여 토큰을 사용

```
01  // file: strtok.c
02  #define _CRT_SECURE_NO_WARNINGS
03  #include <stdio.h>
04  #include <string.h>
05
06  int main(void)
07  {
08     char str1[] = "C and C++\t language are best!";
09     char *delimiter = " ,\t!";   구분자가 공백문자, 쉼표, 수평탭, 느낌표 모두 4개이다.
10     //char *next_token;          함수 strtok_s()의 사용
11
12     printf("문자열 \"%s\"을 >>\n", str1);
13     printf("구분자[%s]를 이용하여 토큰을 추출 >>\n", delimiter);
14     char *ptoken = strtok(str1, delimiter);
15     //char *ptoken = strtok_s(str, delimiter, &next_token);   함수 strtok_s()의 사용
16     while ( ptoken != NULL )
17     {
18        printf("%s\n", ptoken);
19        ptoken = strtok(NULL, delimiter); //다음 토큰을 반환
20        //ptoken = strtok_s(NULL, delimiter, &next_token); //다음 토큰을 반환   함수 strtok_s()의 사용
21     }
                두 번째 호출부터는 첫 인자를 NULL로 호출한다.
22
23     return 0;
24  }
```

설명

```
08  문자열에서 \t는 탭 문자임
12  원 문자열 출력
13  구분자 문자열 출력
14  함수 strtok(str1, delimiter) 호출에 의해 분리되는 문자열 토큰이 저장될 문자 포인터 선언
    하여 strtok() 호출값을 저장
16  분리된 토큰이 NULL이 아니면 반복문 실행, 반복문의 몸체는 18행과 19행
18  분리된 토큰을 한 줄에 출력
19  두 번째 토큰 추출에서 strtok(NULL, delimiter)으로 호출하여 반환값을 ptoken에 저장
```

실행결과

```
문자열 "C and C++        language are best!"을 >>
구분자[ ,       !]를 이용하여 토큰을 추출 >>
C
and
C++
language
are
best
```

문자열의 길이와 위치 검색

- **함수 strlen()**
  - NULL 문자를 제외한 문자열 길이를 반환하는 함수
- **함수 strlwr()**
  - 인자를 모두 소문자로 변환하여 반환
- **함수 strupr()**
  - 인자를 모두 대소문자로 변환하여 반환

표 11-2 다양한 문자열 관련 함수

함수원형	설명
`char * strlwr(char * str);` `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);` `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

LAB 문자열을 역순으로 저장하는 함수 reverse() 구현

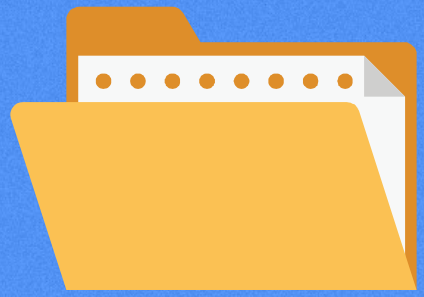
- **함수 memcpy()를 사용**
  - 문자열 상수를 배열에 저장하여 출력한 후,
  - 함수 reverse()를 호출
    - 문자열을 역순으로 저장한 후 그 결과를 출력
    - 문자열 배열을 역순으로 저장하는 함수
  - char 일차원 배열 s[50]를 선언하여 함수 memcpy()로 문자열 "C Programming!"을 저장
  - 함수 reverse(char str[])는 문자열 str을 역순으로 다시 저장
- **다음 결과와 같이 원 문자열과 역순 문자열을 출력**
  - C Programming!
  - !gnimmargorP C

Lab 11-2 strreverse.c

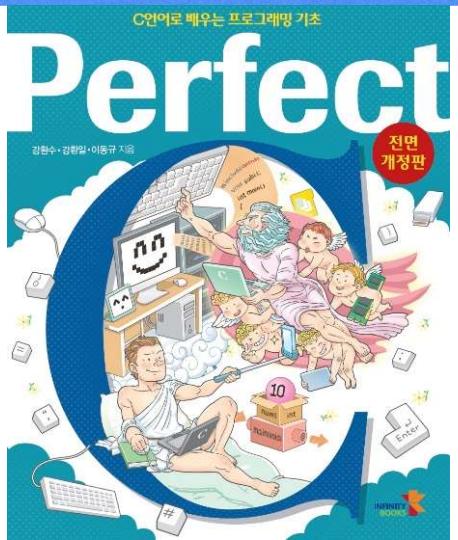
```
01  // strreverse.c:
02  #include <stdio.h>
03  #include ________________________
04
05  void reverse(char str[]);
06
07  int main(void)
08  {
09     char s[50];
10     ________________(s, "C Programming!", strlen("C Programming!")+1);
11     printf("%s\n", s);
12
13     reverse(s);
14     printf("%s\n", s);
15
16     return 0;
17  }
18
19  void reverse(char str[])
20  {
21     for (int i = 0, j = strlen(str) - 1; i < j; i++, j--)
22     {
23        char c = str[i];
24
25        ________________
26        str[j] = c;
27     }
28  }
```

정답

```
03  #include <string.h>
10  memcpy(s, "C Programming!", strlen("C Programming!")+1);
24          str[i] = str[j];
```



03. 여러 문자열 처리



문자 포인터 배열

- **문자 포인터 배열을 이용하는 방법**
  - 여러 개의 문자열을 처리하는 하나의 방법
  - 하나의 문자 포인터가 하나의 문자열을 참조 가능
  - 문자 포인터 배열은 여러 개의 문자열을 참조 가능
- **장 단점**
  - 문자 포인터 배열 이용 방법은 각각의 문자열 저장을 위한 최적의 공간을 사용
  - 문자 포인터를 사용해서는 문자열 상수의 수정은 불가능
    - 문장 pa[0][2] = 'v';와 같이 문자열의 수정은 실행오류가 발생

```
char *pa[] = {"JAVA", "C#", "C++"};
//각각의 3개 문자열 출력
printf("%s ", pa[0]); printf("%s ", pa[1]); printf("%s\n", pa[2]);
```

배열의 크기는 문자열 개수인 3을 지정하거나 빈 공백으로 한다.

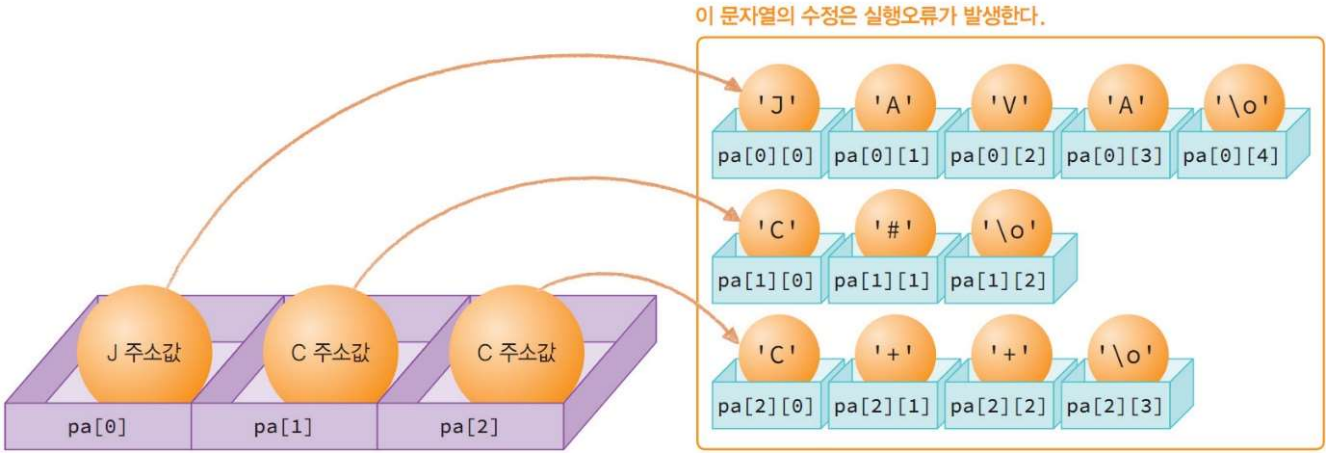


그림 11-22 문자 포인터 배열을 이용한 여러 개의 문자열 처리

이차원 문자 배열

- **여러 개의 문자열을 처리하는 다른 방법**
  - 문자의 이차원 배열을 이용하는 방법
  - 이차원 배열의 열 크기는 문자열 중에서 가장 긴 문자열의 길이보다 1 크게 지정
    - 가장 긴 문자열 "java"보다 1이 큰 5를 2차원 배열의 열 크기로 지정
  - 물론 이차원 배열의 행의 크기는 문자열 수
    - 3으로 지정하거나 공백으로 비워 둠

```
char ca[][5] = {"JAVA", "C#", "C++"};
//각각의 3개 문자열 출력
printf("%s ", ca[0]); printf("%s ", ca[1]); printf("%s\n", ca[2]);
```

첫 번째(행) 크기는 문자열 갯수를 지정하거나 빈 공백으로 두며, 두 번째(열) 크기는 문자열 중에서 가장 긴 문자열의 길이보다 1크게 지정한다.

- **장 단점**
  - 문자의 이차원 배열에서 모든 열 수가 동일하게 메모리에 할당
    - 열의 길이가 서로 다른 경우에는 '₩0' 문자가 들어가 낭비
  - 문자열을 수정 가능
    - ca[0][2] = 'v';와 같이 원하는 문자 수정이 가능

이 문자열의 수정은 실행오류가 발생한다.

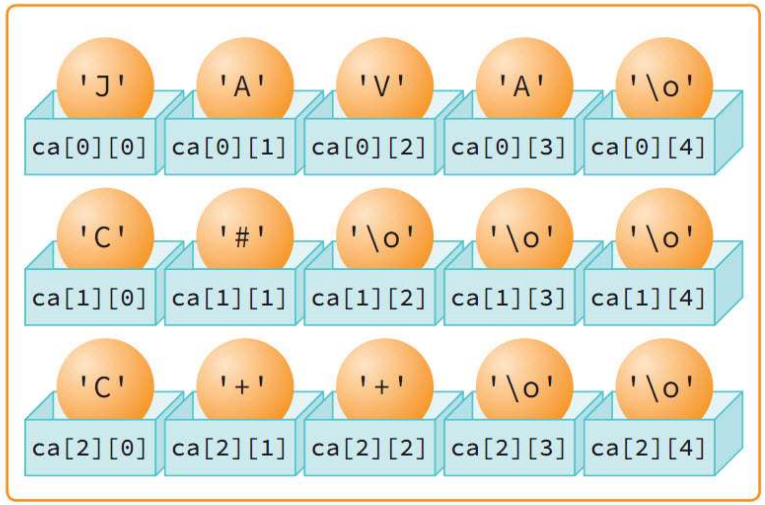


그림 11-23 이차원 문자배열을 이용한 여러 문자열 처리

여러 문자열 처리

예제 ptrarray.c

- 문자 포인터 배열과 이차원 문자 배열

실습예제 11-13 strarray.c

여러 개의 문자열을 선언과 동시에 저장하고 처리하는 방법

```c
// file: strarray.c
#include <stdio.h>

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

   //각각의 3개 문자열 출력
   //pa[0][2] = 'v';   //실행 오류 발생
   //ca[0][2] = 'v';   //수정 가능
   printf("%s ", pa[0]); printf("%s ", pa[1]); printf("%s\n", pa[2]);
   printf("%s ", ca[0]); printf("%s ", ca[1]); printf("%s\n", 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]);

   return 0;
}
```

문자열을 구성하는 각각의 문자를 출력하려면 pa[i][j]와 ca[i][j]을 형식제어문자 %c로 출력한다.

설명

```
문자 포인터 배열 pa를 선언하여 초기값으로 세 개의 문자열을 저장
열 크기가 3인 이차원 문자배열 ca를 선언하면서 초기값으로 "JAVA", "C#", "C++"을 저장
pa[0], pa[1], pa[2] 모두 문자열을 가리키는 포인터로 함수 printf()에서 %s로 문자열 출력 가능
ca[0], ca[1], ca[2] 모두 문자열을 가리키는 포인터로 함수 printf()에서 %s로 문자열 출력 가능
pa[0][1], pa[1][1], pa[2][1] 모두 세 문자열에서 모두 두 번째 문자가 저장된 변수로 함수 printf()에서 %c로 문자 출력 가능
ca[0][1], ca[1][1], ca[2][1] 모두 세 문자열에서 모두 두 번째 문자가 저장된 변수로 함수 printf()에서 %c로 문자 출력 가능
```

실행결과

```
JAVA C# C++
JAVA C# C++
A # +
A # +
```

명령행 인자

- **main(int argc, char *argv[])**
  - 프로그램 dir를 개발한다면 옵션에 해당하는 "/w"를 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 argc는 명령행에서 입력한 문자열의 수
    - argv[]는 명령행에서 입력한 문자열을 전달 받는 문자 포인터 배열
    - 실행 프로그램 이름도 하나의 명령행 인자에 포함

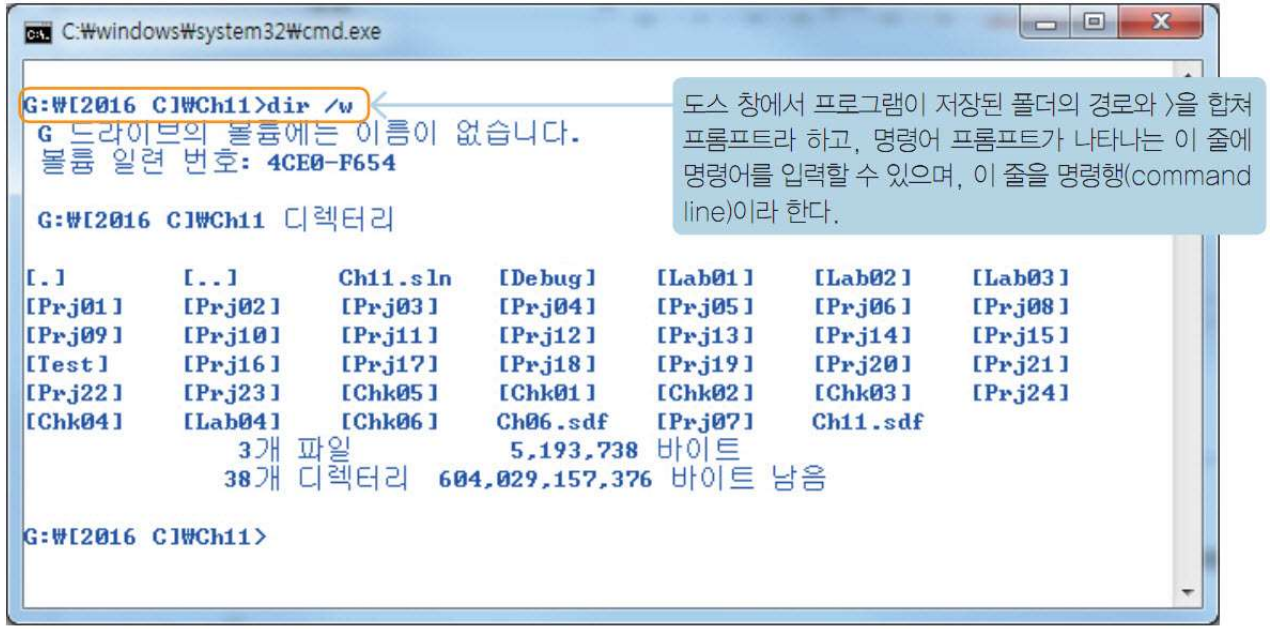


그림 11-24 도스 명령어의 옵션

명령행 인자 실행 샘플

- **명령행에서 실행파일의 이름이 commandarg**
  - 옵션으로 C# C++ Java: 프로그램을 실행한 결과
  - 명령행 인자로 프로그램을 실행하면 다음과 같은 구조의 문자열이 전달

```
int main(int argc, char *argv[])
{
    ...
}
```

도스 창

```
G:\.....\Ch11\Debug>commandarg C# C++ Java
실행 명령어 줄 인자(command line arguments) >>
argc = 4
argv[0] = commandarg
argv[1] = C#
argv[2] = C++
argv[3] = Java
```

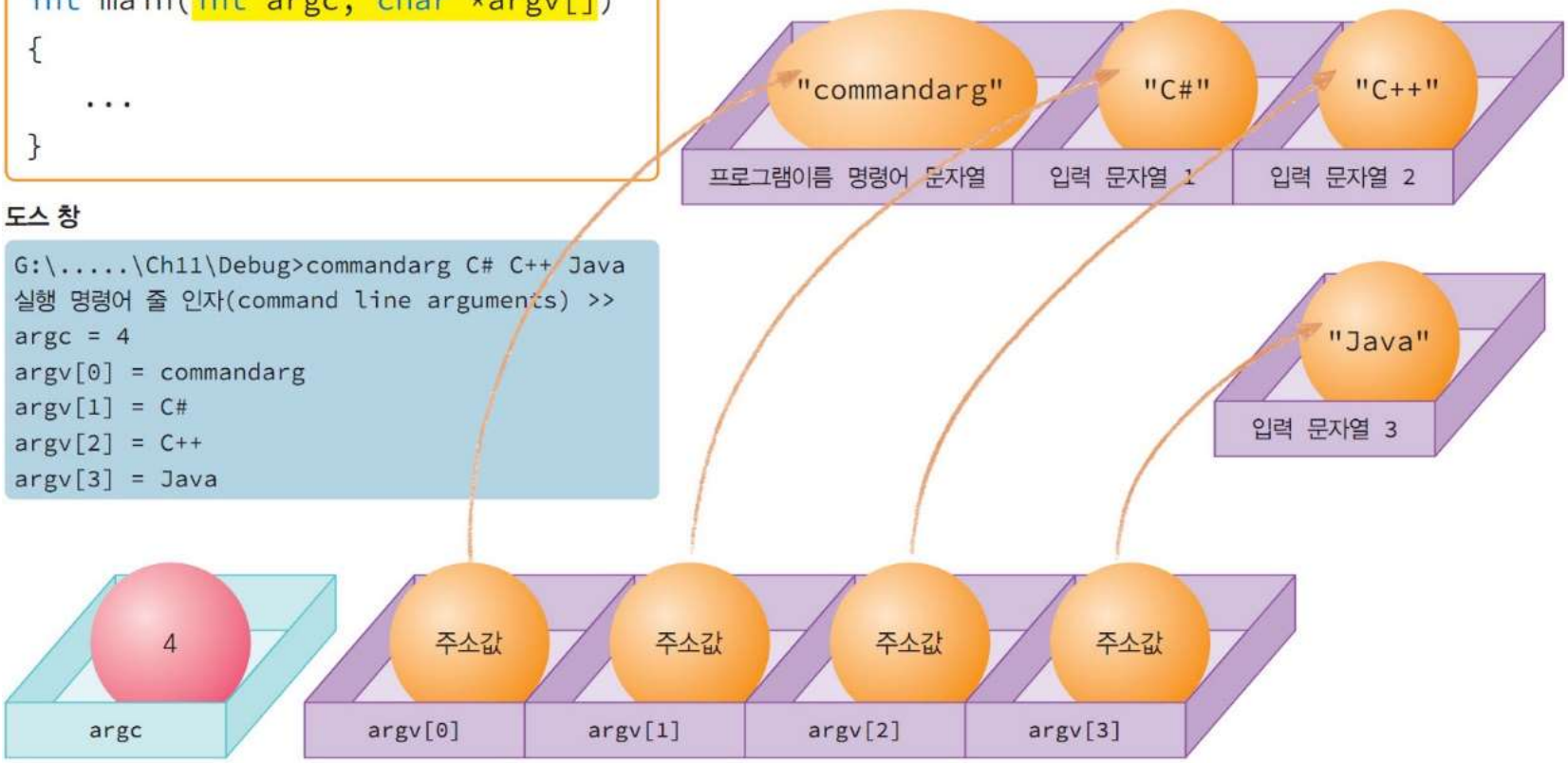


그림 11-25 명령행 인자와 매개변수

명령행 인자 설정

- **Visual C++에서 명령행 인자를 설정**
  - 메뉴 [프로젝트/{프로젝트이름} 속성...]를 누르거나
  - 단축 키 Alt+F7을 눌러 다음 대화상자에서 설정
    - 대화상자 [{프로젝트이름} 속성 페이지]의 항목 [디버깅]을 누르고
    - 중간의 [명령 인수] 의 입력 상자에 인자를 기술
      - 이 입력 상자에는 실행파일 이름 뒤의 옵션만을 기술

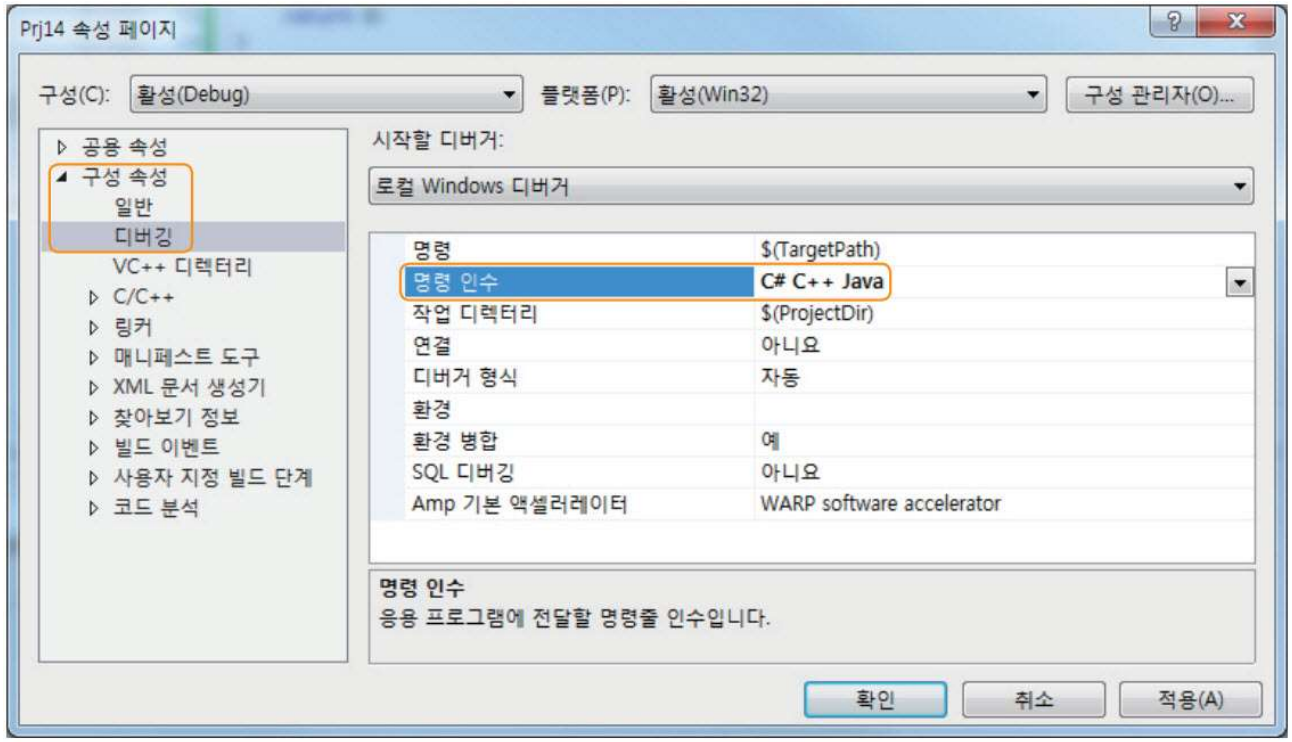


그림 11-26 Visual C++에서 명령행 인자 설정을 위한 대화상자

명령행 인자 예제

예제 commandarg.c

- 인자 argc, argv 활용

포인터 형변환

- 다음은 위 프로그램을 도스 창에서 실행한 결과
  - 도스 창에서 실행한 경우, 실행 결과의 첫 인자 값이 실행파일 이름

실습예제 11-14 commandarg.c

명령행 인자 출력

```
01  // file: commandarg.c
02  #include <stdio.h>
03
04  int main(int argc, char *argv[])
05  {
06     int i = 0;
07
08     printf("실행 명령행 인자(command line arguments) >>\n");
09     printf("argc = %d\n", argc);
10     for (i = 0; i < argc; i++)
11        printf("argv[%d] = %s\n", i, argv[i]);
12
13     return 0;
14  }
```

설명

04 명령행 인자를 처리하려면 main()의 매개변수를 "int argc, char *argv[]"로 기술하는데, argc(argument count)에 인자의 수가, argv(argument variables)에는 인자인 여러 개의 문자열의 포인터가 저장된 배열이 전달됨

09 인자의 수인 argc를 출력

10 제어변수인 i는 0부터 argc보다 작을 때까지 반복을 수행

11 제어변수인 i를 사용하고, 문자열 배열 argv[i]를 참조하여 명령행 인자인 각각의 문자열을 출력

실행결과

```
실행 명령행 인자(command line arguments) >>
argc = 4
argv[0] = G:\[2016 C]\Ch11\Debug\Prj14.exe
argv[1] = C#
argv[2] = C++
argv[3] = Java
```

개발도구에서 실행하면 첫 번째 인자는 폴더를 포함한 실행프로그램 이름이 지정된다.

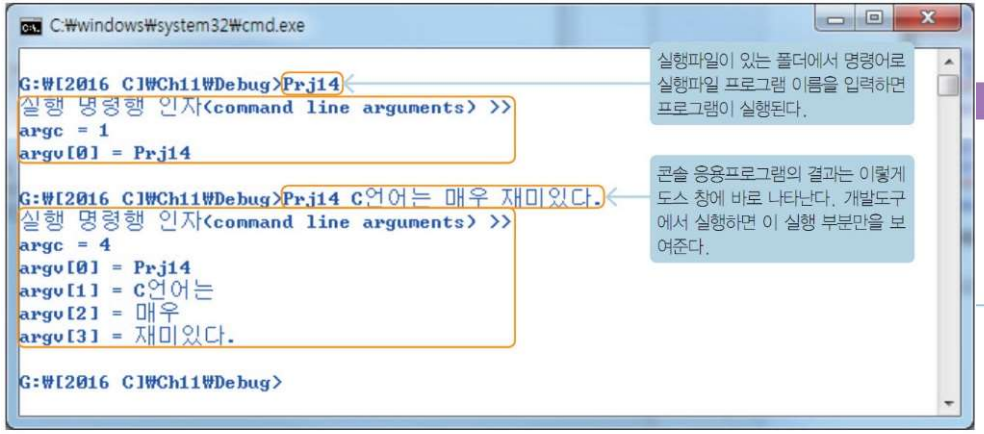


그림 11-27 도스 창에서 commandarg 실행

LAB 여러 문자열 처리

- **여러 문자열을 각각의 일차원 문자배열 str1, str2, str3에 저장한 후,**
  - 문자 포인터 배열 pstr을 선언
    - 문자배열 이름을 초기화로 저장
  - 변수 str1, str2, str3와 pstr을 사용
    - 저장된 문자열과 문자, 적절히 출력
    - str1, str2, str3을 선언하면서 문자열 "JAVA", "C#", "C++"를 저장
    - pstr을 선언하면서 문자열 포인터인 str1, str2, str3을 저장
- **다음과 같이 문자열과 문자를 출력**
  - JAVA C# C++
  - J # +
  - A # +

Lab 11-3 strprocess.c

```c
// file: strprocess.c
#include <stdio.h>

int main(void)
{
   char str1[] = "JAVA";
   char str2[] = "C#";
   char str3[] = "C++";

   char *pstr[] = { ______________________________ };

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

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

   return 0;
}
```

정답

```c
char *pstr[] = { str1, str2, str3 };
printf("%s ", pstr[1]);
   printf("%c %c %c\n", pstr[0][1], pstr[1][1], pstr[2][1]);
```

Thank you