# 제 12 장 문자와 문자열

01 문자와 문자열
02 문자열 관련 함수
03 여러 문자열 처리

# 학습목표

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

# 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 12-1 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";
```

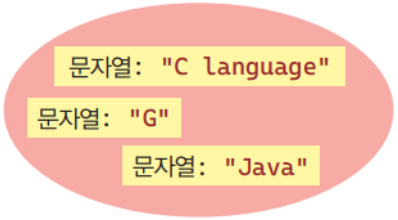



그림 12-2 문자열 상수와 선언

# 문자와 문자열의 선언

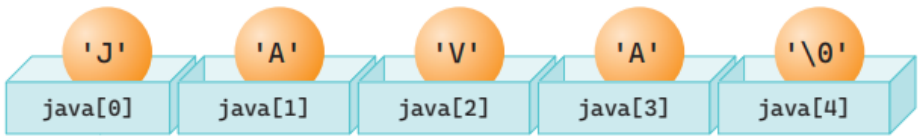
- **문자**
  - char형 변수에 문자를 저장
- **문자열**
  - 문자열을 저장하기 위한 자료형을 따로 제공하지 않음
    - **문자 배열을 선언하여 각각의 원소에 문자를 저장**
    - **문자열의 마지막을 의미하는 NULL 문자 '₩0'가 마지막에 저장**
  - 문자열이 저장되는 배열크기
    - **반드시 저장될 문자 수보다 1이 커야 널(NULL) 문자를 문자열의 마지막으로 인식**
    - **문자열의 마지막에 널(NULL) 문자가 없다면 출력과 같은 문자열 처리에 문제가 발생**
  - 배열 csharp의 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] = '\0';
```

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

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

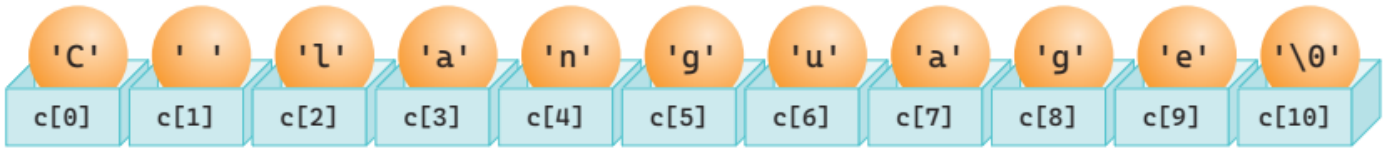
마지막에 '\0'을 빼면, 대입 시에는 문제가 없으나 출력 등에서 문제가 발생한다.



# 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**

```
char c[] = "C language";        //크기를 생략하는 것이 간편
char c[11] = "C language";      //크기 지정 시 (문자수+1)
```




```
char go[5] = "go";              //크기가 (문자+1)보다 크면 나머지는 모두 '\0'로 채워짐
```

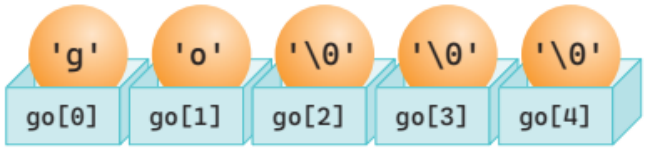



그림 12-5 문자열 상수로 문자배열 초기화

# 문자와 문자열 출력

- **문자 출력**
  - 형식제어문자 %c
- **문자열을 출력**
  - 형식제어문자 %s
    - **배열이름**
    - **문자열의 첫 주소**
- **함수 puts()로 문자 출력**
  - puts(csharp)

Prj01 01chstrprt.c 문자와 문자열 출력 난이도: ★

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

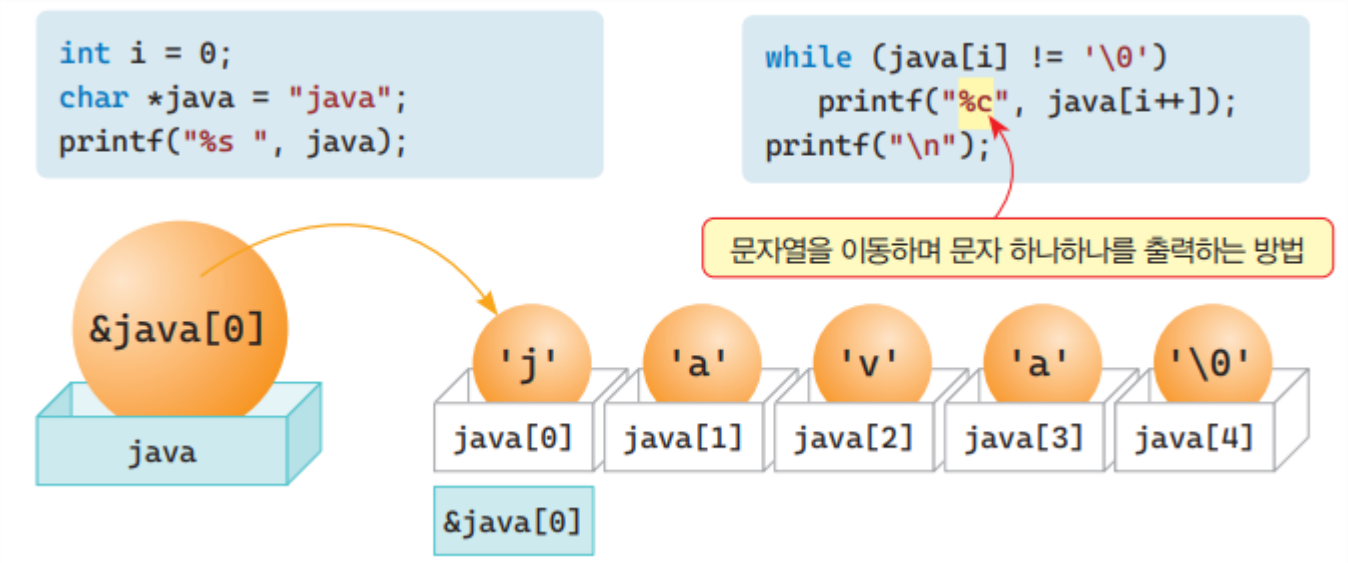
%c로 문자 변수 ch 출력, %d를 사용하면 문자 코드 값 출력

문자열 출력을 위해 배열이름과 형식제어문자 %s를 이용

```
A 65
JAVA
Python
C#
C #
```

# 문자열 구성하는 문자 참조

- **문자열을 처리하는 다른 방법**
  - 문자열 상수를 문자 포인터에 저장하는 방식
    - **문자열을 구성하는 문자 하나 하나의 수정은 불가능**
  - 문자열 출력
    - **함수 printf()에서 포인터 변수와 형식제어문자 %s로 간단히 처리**



# 문자 포인터로 문자열 저장과 출력

- **변수 java**
  - 문자열 상수를 저장하는 문자 포인터
  - 계속 java를 사용해 문자를 참조
  - 사용 상 주의가 필요
    - **변수 java가 가리키는 문자열은 상수이므로 수정 불가능**
- **반복문을 이용**
  - 문자가 '₩0'이 아니면 문자를 출력하도록 하는 방식
    - **출력할 문자열의 끝을 '₩0' 문자로 검사하면 편리**

Prj02 02charptr.c 문자 포인터로 문자열 저장과 출력 난이도: ★

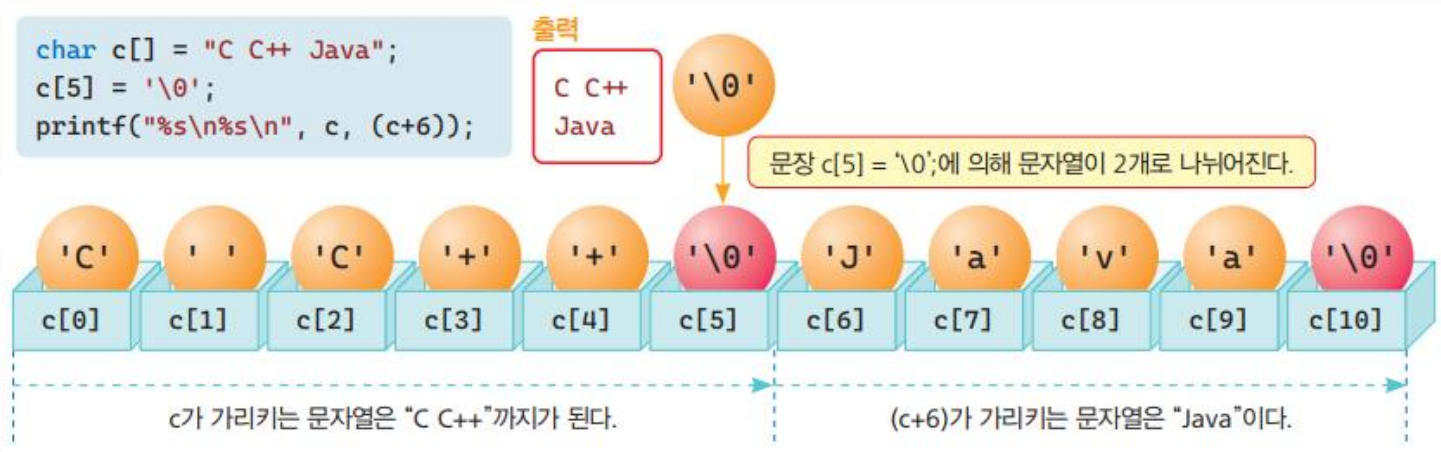
```c
#include <stdio.h>

int main(void)
{
   char *java = "java";
   printf("%s ", java);

   //문자 포인터가 가리키는 문자 이후를 하나씩 출력
   int i = 0;
   while (java[i]) //while (java[i] != '\0')
      printf("%c", java[i++]);
   printf(" ");

   i = 0;
   while (*(java + i) != '\0') //java[i]는 *(java + i)와 같음
      printf("%c", *(java + i++));
   printf("\n");

   //수정 불가능, 실행 결과에 문제 발생
   java[1] = 'A';
   printf("%c", java[1]);

   return 0;
}
```

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

```
java java java
```

# '□0' 문자에 의한 문자열 분리

- **함수 printf()에서 %s**
  - 문자 포인터가 가리키는 위치에서 NULL 문자까지를 하나의 문자열로 인식
- **만일 배열 c에 문장 c[5] = '₩0';을 실행**
  - c를 출력하면 무엇이 출력될까?
  - c[5]에 저장된 '₩0' 문자에 의해 c가 가리키는 문자열은 "C C++"까지
    - **즉 문자열은 시작 문자부터 '₩0' 문자가 나올 때까지 하나의 문자열로 처리**
  - (c+6)로 문자열을 출력하면 "Java"가 출력



# 문자열 중간에 '□0'을 삽입해 문자열 분리

실습예제 12-3 | Prj03 | 03strsplit.c | 문자열 중간에 '\0'을 삽입해 문자열 분리 | 난이도: ★

```c
#include <stdio.h>

int main(void)
{
    char c[] = "C C++ Java";
    printf("%s\n", c);
    c[5] = '\0'; // NULL 문자에 의해 문자열 분리
    printf("%s\n%s\n", c, (c + 6));

    //문자 배열의 각 원소를 하나하나 출력하는 방법
    c[5] = ' '; //널 문자를 빈 문자로 바꾸어 문자열 복원
    char *p = c;
    while (*p) //(*p != '\0')도 가능
       printf("%c", *p++);
    printf("\n");

    return 0;
}
```

c[5]에 '\0'를 저장하면 "C C++\0Java\0"이 저장되어 두 개의 문자열 "C C++"와 "Java"로 나뉨

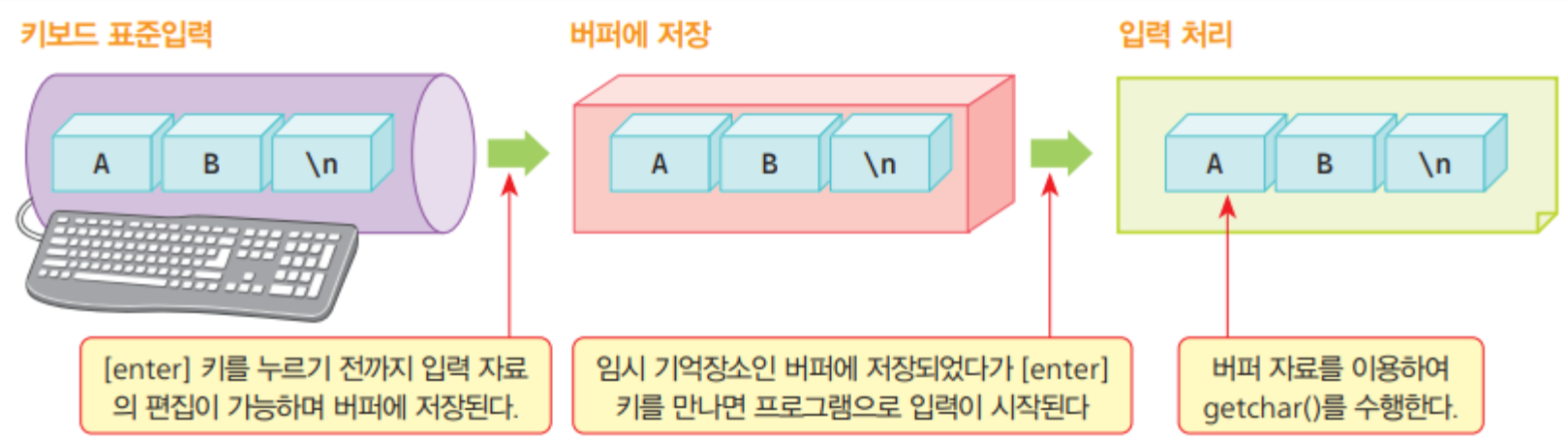
```c
int i = 0;
while  (c[i])
   printf("%c", c[i++]);
printf("\n");

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

결과

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

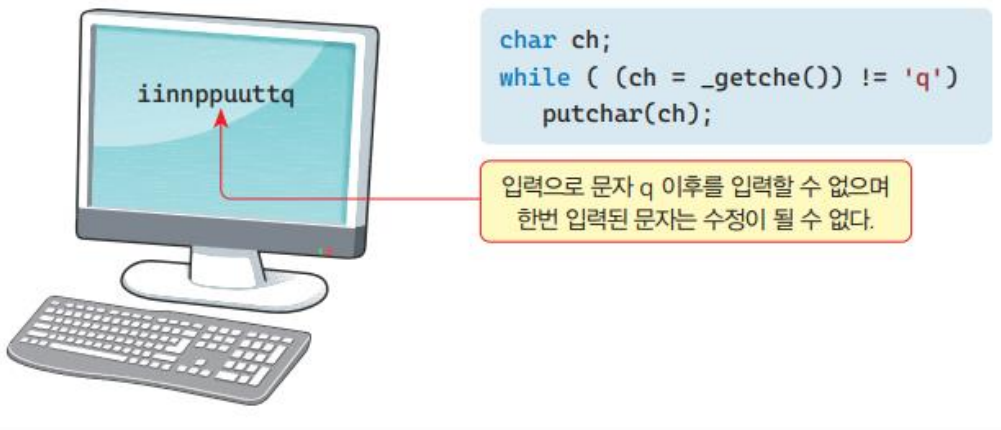
# 다양한 문자 입출력

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



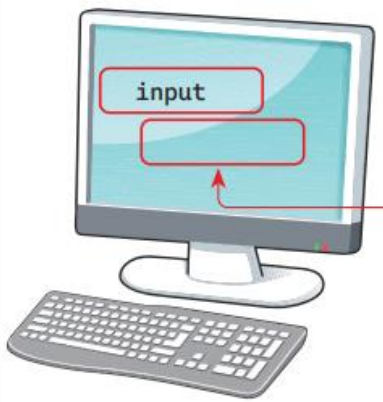
# 버퍼를 사용하지 않고 문자를 입력하는 함수 _getche()

- **함수 _getche()**
  - 버퍼를 사용하지 않으므로 문자 하나를 입력하면 바로 함수 getche()가 실행
  - 함수 getche()에서 입력된 문자는 바로 모니터에 표시
    - **마지막 e는 입력문자를 표시하다는 echo를 의미**
  - 헤더파일 conio.h를 삽입 필요
- **입력 문자가 'q'가 아니면**
  - 함수 putchar()에 의하여 문자가 바로 출력
  - 입력된 문자도 보이고, 바로 putchar()에 의하여 출력
    - **입력문자가 "inputq"**
    - **화면에는 "iinnppuuttq"가 표시**



# 입력한 문자가 화면에 보이지 않는 함수 _getch()

- **함수 _getch()**
  - 입력된 문자를 출력함수로 따로 출력하지 않으면
    - **입력 문자가 화면에 보이지(echo) 않음**
    - **버퍼를 사용하지 않는 문자 입력 함수**
  - conio.h 파일 삽입 필요
  - 문자 "inputq"를 입력으로 실행하면
    - **"input"이 출력**
    - **함수 putch(ch)는 인자를 출력하는 함수**




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

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

**표 12-1** 문자입력 함수 scanf(), getchar(), _getche(), _getch()의 비교

| 함수 | scanf("%c", &ch) | getchar() | _getche() | _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) | 입력하면 바로 표시 | | 입력하면 바로 표시 | 표시 안됨 |
| 입력문자 수정 | 가능 | | 불가능 | |

# 함수 getchar(), _getche(), _getch()의 차이

- **세 개의 while 문의 입력**
  - 모두 "pythonq"를 입력
    - **마지막 문자 q가 루프를 종료하는 조건 문자**
- **문자 출력**
  - 헤더파일 conio.h가 필요
  - _putch(char)
    - **putchar(char)처럼 문자 char을 표준 출력**

Prj04 04getch.c 함수 getchar(), _getche(), _getch()의 차이를 알아보는 예제 난이도: ★

```
01 #include <stdio.h>
02 #include <conio.h>
03
04 int main(void)
05 {
06    char ch;
07
08    printf("문자를 계속 입력하고 Enter를 누르면 >>\n");
09    while ((ch = getchar()) != 'q')
10       putchar(ch); // stdio.h
11
12    printf("\n문자를 누른 것이 보이고, _putch에 의해 입력문자 출력 >>\n");
13    while ((ch = _getche()) != 'q')
14       _putch(ch); // conio.h
15
16    printf("\n문자를 누른 것이 안 보이고, _putch에 의해 입력문자 출력 >>\n");
17    while ((ch = _getch()) != 'q')
18       _putch(ch); // conio.h
19    printf("\n");
20
21    return 0;
22 }
```

함수 getchar()로 입력한 문자를 변수 ch에 저장하여 'q'가 아니면 반복몸체인 putchar(ch)를 실행, 변수 ch에 'q'이면 반복이 종료되므로 출력이 안됨

함수 _getche()로 입력한 문자를 변수 ch에 저장하여 'q'가 아니면 반복몸체인 _putch(ch)를 실행, 변수 ch에 'q'이면 반복 종료이므로 출력이 안됨

함수 _getch()로 입력한 문자를 변수 ch에 저장하여 'q'가 아니면 반복몸체인 _putch(ch)를 실행, 변수 ch에 'q'이면 반복 종료이므로 출력이 안됨. 함수 _getch()는 입력한 문자마다 바로 처리하며, 입력한 문자도 보이지 않음

```
문자를 계속 입력하고 Enter를 누르면 >>
pythonq
python
문자를 누른 것이 보이고, _putch에 의해 입력문자 출력 >>
ppyytthhoonnq
문자를 누른 것이 안 보이고, _putch에 의해 입력문자 출력 >>
python
```

마지막에 입력한 문자 q가 입력 종료를 처리

마지막에 입력한 문자 q가 입력 종료를 처리

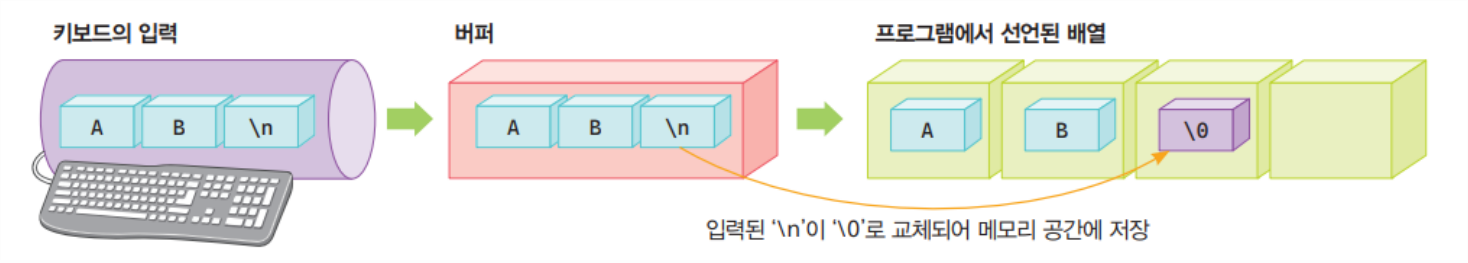
입력을 pythonq와 같이 마지막에 q를 입력해야 종료되며, 마지막에 입력한 q는 출력되지 않음

# 문자열의 입출력

- **함수 scanf()는 공백으로 구분되는 하나의 문자열을 입력**
  - 입력 받은 문자열이 저장될 충분한 공간인 문자 배열 str을 선언
  - 함수 scanf("%s", str)
    - **형식제어문자 %s를 사용하여 문자열 입력**
  - 함수 printf("%s", str)
    - **%s를 사용하여 문자열을 출력**
  - 단순히 문자 포인터로는 문자열 저장 불가능
- **gets()와 puts()**
  - 함수 gets()는 한 행의 문자열 입력에 유용한 함수
  - 함수 puts()는 한 행에 문자열을 출력하는 함수
  - 함수 gets(), puts(), gets_s()를 사용하려면 헤더파일 stdio.h 를 삽입
- **Visual C++**
  - 함수 gets_s()는 현재 함수 gets()의 대체함수로 사용을 권장

# 함수 gets()

- **[enter] 키를 누를 때까지 한 행을 버퍼에 저장한 후 입력처리**
  - 함수 gets()는 마지막에 입력된 '₩n'이 '₩0'로 교체되어 인자인 배열에 저장
    - **그러므로 프로그램에서 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);
```

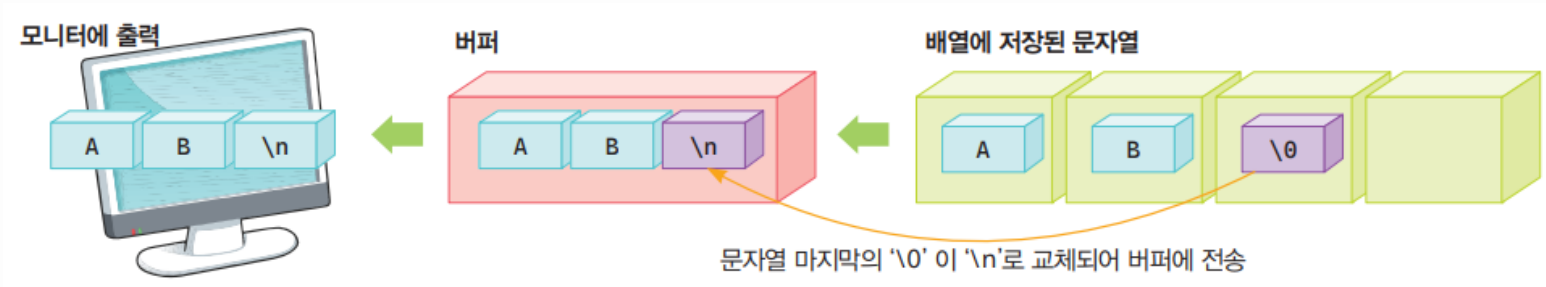
- 두 번째 인자인 sizebuffer는 정수형으로 buffer의 크기를 입력한다.
- Visual C++에서는 앞으로 gets() 대신 함수 gets_s()의 사용을 권장한다.

```
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.

# 함수 gets()와 puts()

- **함수 gets()의 인자**
  - 입력된 문자열을 저장할 수 있는 충분한 공간의 문자 배열을 사용
    - **그렇지 않으면 실행오류가 발생**
- **문자열 출력함수 puts()**
  - 문자열을 한 줄에 출력하는데 유용하게 사용
  - gets()와 반대로 문자열의 마지막에 저장된 '₩0'를 '₩n'로 교체하여 버퍼에 전송
    - **버퍼의 내용이 모니터에 출력**
- **기호 상수 EOF(End Of File)**
  - #define EOF (-1)
    - **파일의 끝이라는 의미로 stdio.h 헤더파일에 정수 -1로 정의**



# gets(), puts()

- 함수 scanf()에서 제어문자 %s
  - 문자열을 입력
- 함수 gets()와 gets_s()를 사용
  - 여러 줄을 입력
- while문을 사용
  - 연속된 여러 행을 입력 받아 바로 행 별로 출력
  - 다음 반복을 종료
    - **새로운 행 처음에 (ctrl + Z)를 입력**
- 함수 printf()와 scanf()
  - 다양한 입출력에 적합
- 함수 puts()와 gets()
  - 처리 속도가 빠르다는 장점

Prj05 05strinput.c 함수 scanf()와 printf(), gets()와 puts()의 문자열 입출력 난이도: ★

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
   char name[20], dept[30]; //char *name, *dept; 실행 오류 발생
   int snum;
   printf("학번 이름 학과 입력 >>\n");
   scanf("%d %s %s", &snum, dept, name);
   printf("출력: %d %s %s\n", snum, dept, name);

   char line[101]; //char *line 으로는 오류발생
   printf("한 행에 학번 이름 학과 입력한 후 [enter]를 누르고 ");
   printf("새로운 행에서 (ctrl + Z)를 누르십시오.\n");
   while (gets(line))
      puts(line);
   printf("\n");

   while (gets_s(line, 101))
      puts(line);
   printf("\n");

   return 0;
}
```

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

한 줄에 입력되는 모든 문자열이 입력되도록 충분한 크기의 문자배열 line[101] 선언

함수 gets(line) 호출로 한 줄 전체를 입력 받음, 새로운 행에서 ctrl + Z 입력하면 while 반복 종료

함수 gets_s(line, 101) 호출로 한 줄 전체를 입력 받음, 새로운 행에서 ctrl + Z 입력하면 while 반복 종료

```
학번 이름 학과 입력 >>
20222007 컴퓨터정보공학과 나윤희
출력: 20222007 컴퓨터정보공학과 나윤희
한 행에 학번 이름 학과 입력한 후 [enter]를 누르고 새로운 행에서 (ctrl + Z)를 누르십시오.

20222007 컴퓨터정보공학과 나윤희
20222007 컴퓨터정보공학과 나윤희
^Z

20222007 컴퓨터정보공학과 나윤희
20222007 컴퓨터정보공학과 나윤희
^Z
```

공백으로 구분해 snum, dept, name에 저장

한 행 모두가 배열 line에 저장

새로운 행에서 ctrl + Z 입력하면 while 반복 종료

# Lab 한 행을 표준입력으로 받아 문자 하나 하나를 그대로 출력

- **함수 gets_s()를 사용**
  - 한 행의 표준입력
  - 배열 s에 저장한 후,
  - 문자 포인터 p를 사용
    - **배열 s에서 문자 하나 하나를 이동하면서 출력**
- **char 포인터 변수 p**
  - 선언하면서 배열 s의 첫 원소를 가리키도록 저장
- **포인터 변수 p**
  - p는 주소 값
  - *p는 p가 가리키는 곳의 문자

Lab 12-1 lab1lineprt.c

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

정답

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

# 다양한 문자열 라이브러리 함수

- **헤더파일 string.h에 함수원형으로 선언된 라이브러리 함수로 제공**
  - 문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리 함수
- **함수에서 사용되는 자료형: 64비트 윈도우 시스템인 경우**
  - size_t
    - **비부호 정수 long long형(unsigned __int64)**
  - void *
    - **아직 정해지지 않은 다양한 포인터를 의미**

**표 12-2** 문자열 배열에 관한 다양한 함수

| 함수원형 | 설명 |
|---|---|
| size_t strlen(const char *str) | 포인터 src 위치에서부터 널 문자를 제외한 문자열의 길이 반환 |
| void *memcpy(void *dest, const void *src, size_t n) | 포인터 src 위치에서 dest에 n 바이트를 복사한 후 dest 위치 반환 |
| 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 *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 |

# 함수 strcmp()와 strncmp()

- **문자열 관련 함수는 대부분 strOOO()로 명명**
  - 대표적인 문자열 처리 함수, 두 문자열을 비교하는 함수

문자열 비교 함수: 헤더파일 string.h 삽입

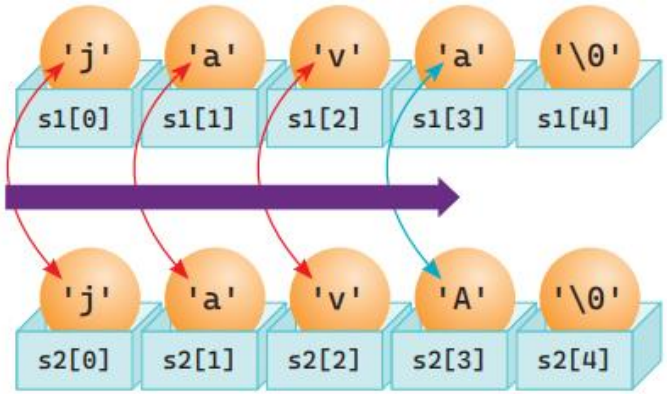
```
int strcmp(const char * s1, const char * s2);
```

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

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

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

  - 인자인 두 문자열을 사전(lexicographically) 상의 순서로 비교하는 함수
  - strncmp()
    - **두 문자를 비교할 문자의 최대 수를 지정하는 함수**




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

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

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

# memcpy(), strcmp()

- **strlen()**
  - 문자열의 길이를 반환
- **memcpy()**
  - 문자배열의 복사를 위한 함수

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

  - 포인터 src 위치에서 dest에 n 바이트를 복사한 후 dest 위치 반환

- **strcmp()와 strncmp()**
  - 문자 열을 비교

Prj06 06strfun.c 문자열 함수 strlen()과 memcpy(), strcmp()와 strncmp()의 활용 난이도: ★

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

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

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

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

   return 0;
}
```

문자열 src에서 포인터 위치 dst에 strlen(src) + 1 수만큼 복사하여 문자열의 마지막이 널이 되도록

포인터 위치 dst에 문자열 상수 "안녕하세요!"를 복사하기 위해서는 (strlen("안녕하세요!") + 1)만큼 복사해야 문자열의 마지막이 널이 됨

비교 세 번째 문자가 각각 'l'과 '0'이므로 strcmp(s1, s2)는 양수임

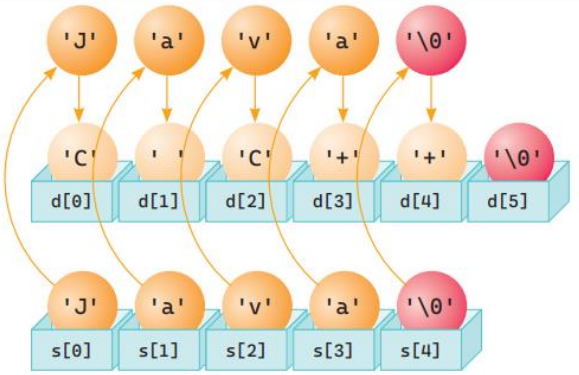
비교 세 번째 문자가 각각 '0'과 'l'이므로 strcmp(s1, s2)는 음수임

```
C Python
8
C Python
안녕하세요!

strcmp(C lang, C lang) = 0
strcmp(C lang, C ) = 1
strcmp(C , C lang) = -1
strncmp(C lang, C , 2) = 0
```

문자열 "C Python"의 길이는 8

# 함수 strcpy()

- **함수 strcpy()**
  - 문자열을 복사하는 함수
    - **앞 인자 문자열 dest에 뒤 인자 문자열 source를 복사**
    - **항상 문자열은 마지막 널 문자까지 포함하므로 NULL 문자 까지 복사**
    - **첫 번째 인자 dest는 복사 결과가 저장될 수 있도록 충분한 공간을 확보**




결과는 d에도 "java"가 저장된다.

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

- **함수 strncpy()**
  - 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 source를 복사하여 그 복사된 문자열을 반환한다.
- 앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.

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

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

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

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

# 함수 strcat()

- **하나의 문자열 뒤에 다른 하나의 문자열을 연이어 추가해 연결**
  - 앞 문자열에 뒤 문자열의 null 문자까지 연결
    - **앞의 문자열 주소를 반환하는 함수**
  - 인자 dest의 저장공간이 연결된 문자열의 길이를 모두 수용할 수 있는 공간보다 부족
    - **문제가 발생**
- **문제를 예방하기 위한 함수가 strncat() 함수**
  - 전달 인자의 마지막에 연결되는 문자의 수를 지정하여 그 이상은 연결되지 않도록
    - **여기서 지정하는 문자 수는 널 문자를 제외한 수**

문자열 연결 함수

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

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

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

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

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

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

# strcpy()와 strcat()활용

Prj07 07strcpycat.c 문자열 복사와 연결 함수 strcpy()와 strcat()활용 난이도: ★

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

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

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

    printf("%s\n\n", strncpy(dest, "C#", 3));
    //printf("%d\n", strncpy_s(dest, 80, "C#", 3));
    //printf("%s\n", dest);

    char data[80] = "C";

    printf("%s\n", strcat(data, " is "));
    //printf("%d\n", strcat_s(data, 80, " is "));
    //printf("%s\n", data);
    printf("%s\n", strncat(data, "a java", 2));
    //printf("%d\n", strncat_s(data, 80, "a proce", 2));
    //printf("%s\n", data);
    printf("%s\n", strcat(data, "procedural "));
    printf("%s\n", strcat(data, "language."));

    return 0;
}
```

비주얼 스튜디오에서 추천하는 strcpy_s() 사용 방법

"C#"을 dest에 2바이트인 "C#"까지만 복사

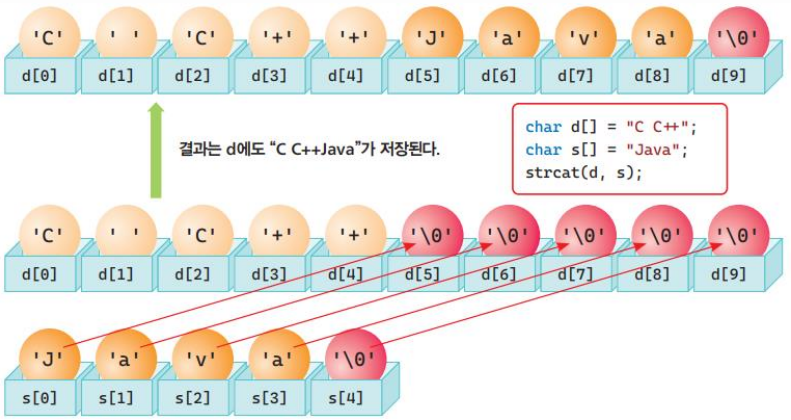
"C#"을 dest에 3바이트인 "C#\0"까지 복사

비주얼 스튜디오에서 추천하는 strncpy_s() 사용 방법

2개 문자인 "a "까지만 뒤에 연결되어 문자열은 "C is a"가 됨

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

\0 이전인 C#만 출력




**TIP 함수 strcpy()와 strcat()를 이용할 시 주의점**

함수 strcpy()와 strcat()를 이용할 시 첫 번째 인자인 dest는 복사 또는 연결 결과가 저장될 수 있도록 충분한 공간을 확보해야 한다. 또한 문자열 관련 함수에서 단순히 문자열 포인터를 수정이 가능한 문자열의 인자로 사용할 수 없다. 즉 함수 strcpy()와 strcat()에서 첫 인자로 문자열 포인터변수는 사용할 수 없다. 그러므로 다음 소스는 모두 실행 시 바른 결과가 표시되지 않는다. 다양한 문자열 관련 함수에서 자료형이 (char *)인 인자에는 문자열 상수를 사용할 수 없으며, 자료형이 (const char *)인 인자에는 문자열 상수를 사용할 수 있다.

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

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

# 문자열 분리 함수 strtok()

- **문자열에서 구분자(delimiter)로 토큰을 추출하는 함수**
  - 첫 번째 인자인 str은 토큰을 추출할 대상인 문자열
    - **str은 문자배열에 저장된 문자열을 사용**
      - **str은 문자열 상수를 사용 불가능**
  - 두 번째 인자인 delim은 구분자로 문자의 모임인 문자열
    - **문자 여러 개를 지정한 문자열 상수를 구분자로 지정 가능**
- **문자열 "C and C++₩t languages are best!"**
  - 구분자를 공백문자 하나인 " "로 지정
    - **토큰을 분리하면 C, and, C++₩t, languages, are, best! 총 6개의 토큰이 추출**
      - **공백문자 분리자를 이용하여 토큰을 분리**
    - **구분자에 더 많은 문자를 삽입**
      - **분리된 토큰이 많아지거나 하나 하나의 토큰에서 구분자가 사라짐**

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

# 함수 strtok()의 사용방법

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

Prj12 08strtok.c 문자열에서 지정한 분리자를 사용해 문자열 토큰을 분리 난이도: ★★

```
01 #define _CRT_SECURE_NO_WARNINGS // strtok() 사용하기 위해
02 #include <stdio.h>
03 #include <string.h>
04
05 int main(void)
06 {
07    char str[] = "C and C++\t languages are best!";
08    char *delimiter = " !\t";
09    //char *next_token;
10
11    printf("문자열 \"%s\"을 >>\n", str);
12    printf("구분자[%s]를 이용하여 토큰을 추출 >>\n", delimiter);
13    char* ptoken = strtok(str, delimiter);
14    //char* ptoken = strtok_s(str, delimiter, &next_token);
15    while (ptoken) //(ptoken != NULL)
16    {
17       printf("%s\n", ptoken);
18       ptoken = strtok(NULL, delimiter); //다음 토큰을 반환
19       //ptoken = strtok_s(NULL, delimiter, &next_token); //다음 토큰을 반환
20    }
21
22    return 0;
23 }
```

구분자가 공백문자, 느낌표 !, 수평탭 모두 3개

9, 14, 19행을 사용해 strtok_S() 활용

두 번째 호출부터는 첫 인자를 NULL로 호출

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

Prj09 09strstr.c 다양한 문자열 관련 함수의 이해

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

int main(void)
{
   char str[] = "JAVA 2022 Python C";
   printf("%zu\n", strlen("python"));//python 길이: 6
   printf("%s, ", _strlwr(str));   //모두 소문자로 변환
   printf("%s\n", _strupr(str));   //모두 대문자로 변환

   //문자열 VA가 시작되는 포인터 반환: VA 2022 PYTHON C
   printf("%s, ", strstr(str, "VA"));
   //문자 A가 처음 나타나는 포인터 반환: AVA 2022 PYTHON C
   printf("%s\n", strchr(str, 'A'));

   return 0;
}
```

```
6
java 2022 python c, JAVA 2022 PYTHON C
VA 2022 PYTHON C, AVA 2022 PYTHON C
```

# Lab 문자열을 역순으로 저장하는 함수 reverse() 구현

- **함수 memcpy()를 사용**
  - 문자열 상수 str
    - **문자열 "Python C"를 저장**
  - char 일차원 배열 s[50]를 선언
    - **함수 memcpy()로**
    - **문자열 "Python C"이 저장된 str을 복사**
- **함수 reverse()를 호출**
  - 문자열을 역순으로 저장한 후 그 결과를 출력
  - 함수 reverse()
    - **문자열 배열을 역순으로 저장하는 함수**
- **원 문자열과 역순 문자열을 출력**
  - Python C
  - C nohtyp

lab2reversestr.c

```
01  #include <stdio.h>
02  #include <string.h>
03
04  void reverse(char []);
05
06  int main(void)
07  {
08     char s[50];
09     char* str = "Python C";
10     memcpy(s, [    ], strlen(str) + 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 = (int) strlen(str) - 1; i < j; i++, j--)
22     {
23        char c = str[i];
24        [                ]
25        str[j] = c;
26     }
27  }
```

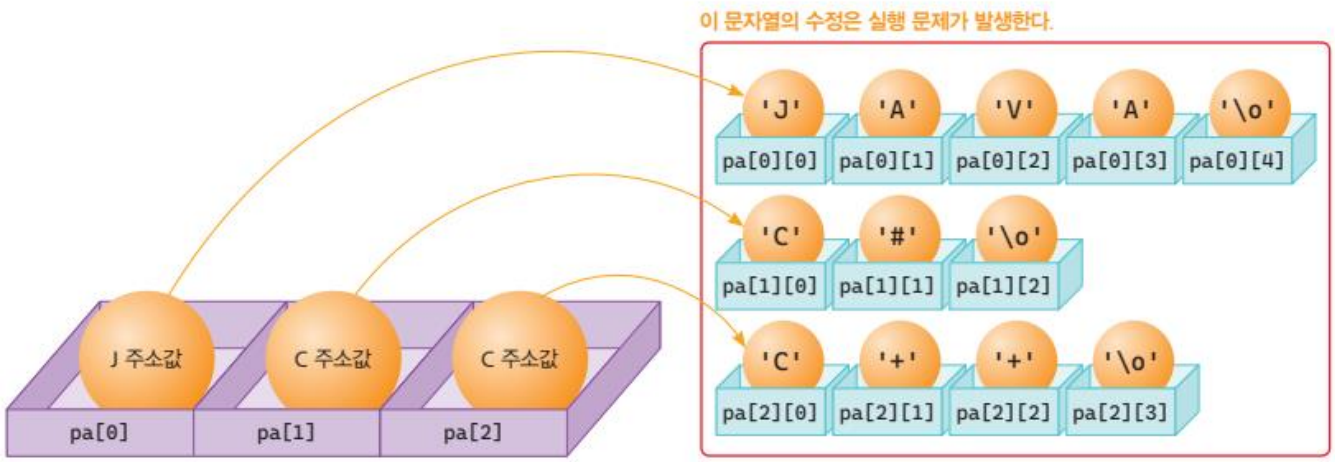
```
10     memcpy(s, str, strlen(str) + 1);
24        str[i] = str[j];
```

# 문자 포인터 배열

- **여러 개의 문자열을 처리하는 하나의 방법**
  - 하나의 문자 포인터가 하나의 문자열을 참조 가능
    - **문자 포인터 배열은 여러 개의 문자열을 참조 가능**
- **장 단점**
  - 문자 포인터를 사용해서는 문자열 상수의 수정은 불가능
    - **문장 pa[0][2] = 'v';와 같이 문자열의 수정은 실행오류가 발생**
    - **문자 포인터 배열 pa를 이용**
      - **각 문자열을 출력하려면 pa[i]로 형식제어문자 %s를 이용**

```
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.



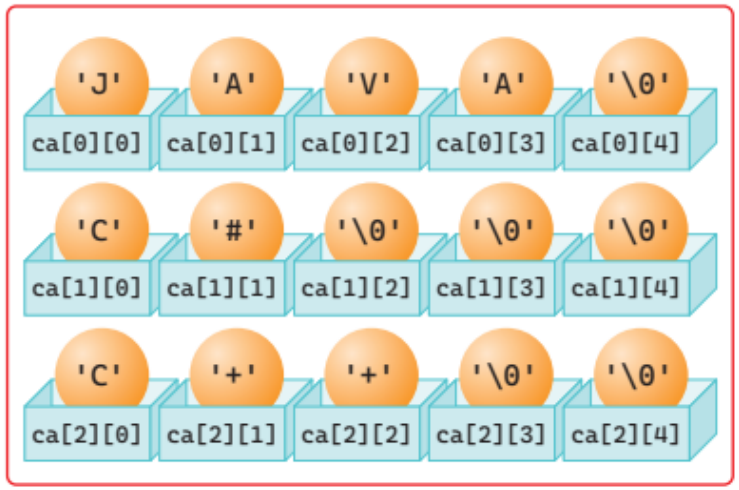
# 문자의 이차원 배열을 이용하는 방법

- **문자 이차원 배열**
  - 이차원 배열의 열 크기
    - **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 갯수
    - **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 개수를 지정하거나 빈 공백으로 두며, 두 번째(열) 크기는 문자열 중에서 가장 긴 문자열의 길이보다 1크게 지정한다.

이 문자열의 수정될 수 있다.




- **장 단점**
  - 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**

# 여러 개의 문자열을 선언과 동시에 저장하고 처리하는 방법

실습예제 12-10 | Prj10 | 10strary.c | 여러 개의 문자열을 선언과 동시에 저장하고 처리하는 방법 | 난이도: ★★

```c
#include <stdio.h>

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

   //pa[0][2] = 'v';   //실행 문제 발생
   //ca[0][2] = 'v';   //수정 가능
   //각각의 3개 문자열 출력
   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;
}
```

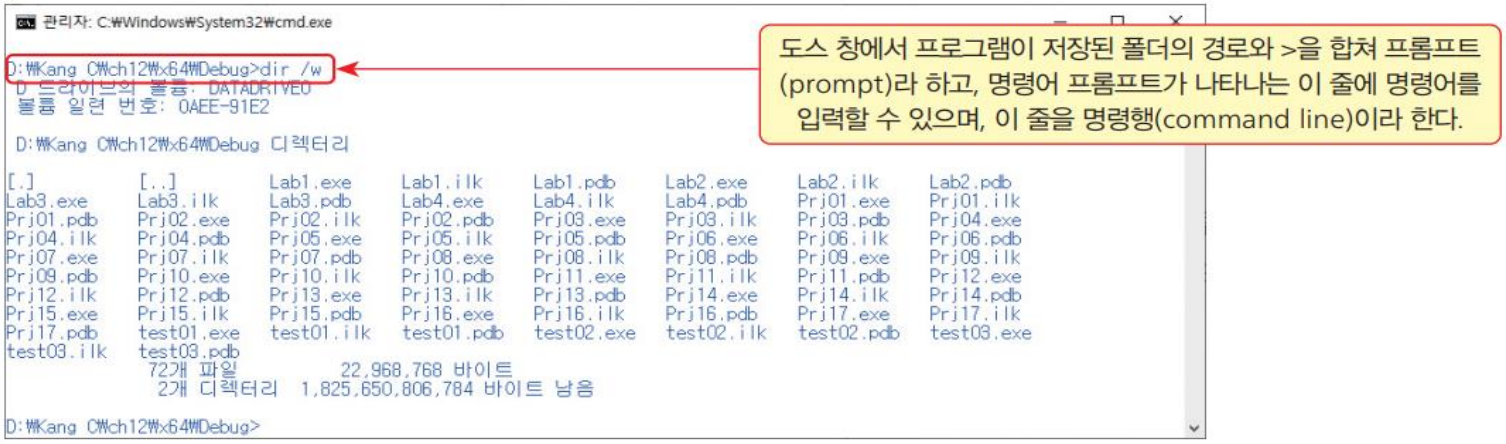
문자열을 구성하는 각각의 문자를 출력하려면 pa[i][j]와 ca[i][j]을 형식제어문자 %c로 출력

결과

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

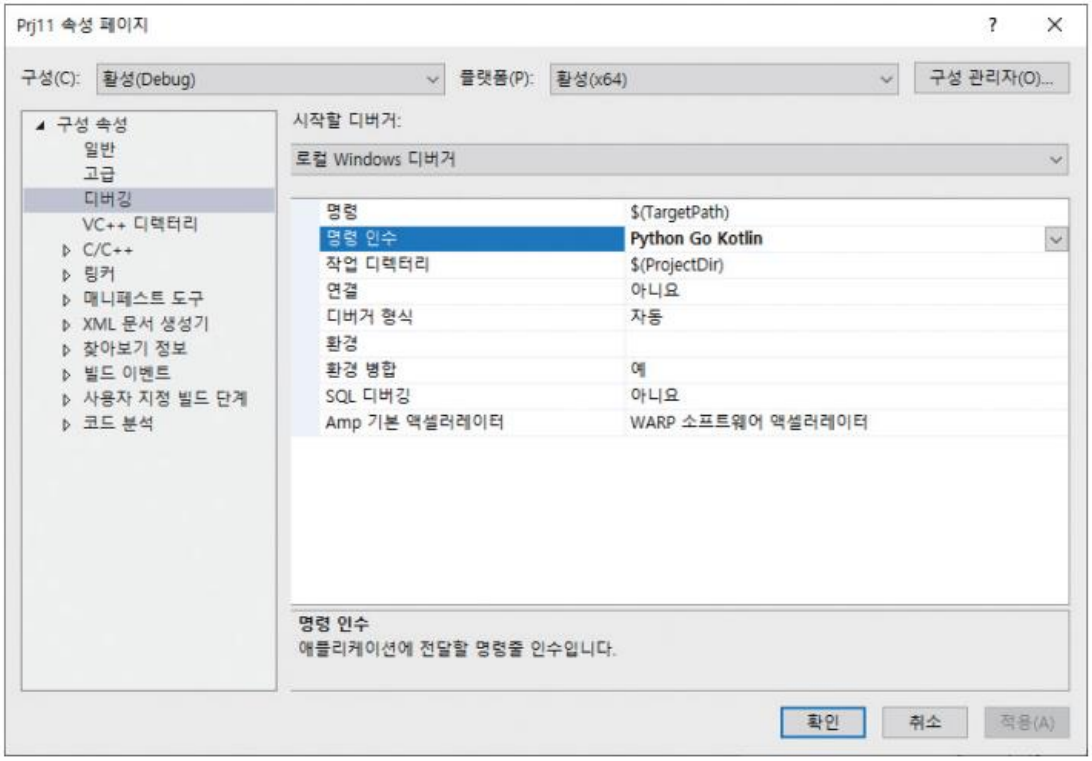
# 명령행 인자

- **main(int argc, char *argv[])**
  - 도스명령어 dir를 프로그램으로 개발한다면 옵션 "/w" 등은 어떻게 인식할까?
    - **명령행 인자(command line arguments)를 사용하는 방법**
      - **명령행에서 입력하는 문자열을 프로그램으로 전달하는 방법**
- **프로그램에서 명령행 인자를 받으려면**
  - 두 개의 인자 argc와 argv를 (int argc, char * argv[])로 기술
    - **매개변수 argc**
      - **명령행에서 입력한 문자열의 수**
    - **argv[]**
      - **명령행에서 입력한 문자열을 전달 받는 문자 포인터 배열**
    - **실행 프로그램 이름도 하나의 명령행 인자에 포함**



# 비주얼 스튜디오에서 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**



# 명령행 인자 출력

실습예제 12-11 | Prj11 | 11cmdarg.c | 명령행 인자 출력 | 난이도: ★★

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

argc(argument count)에 인자의 수가, argv(argument variables)에는 인자인 여러 개의 문자열의 포인터가 저장된 배열이 전달

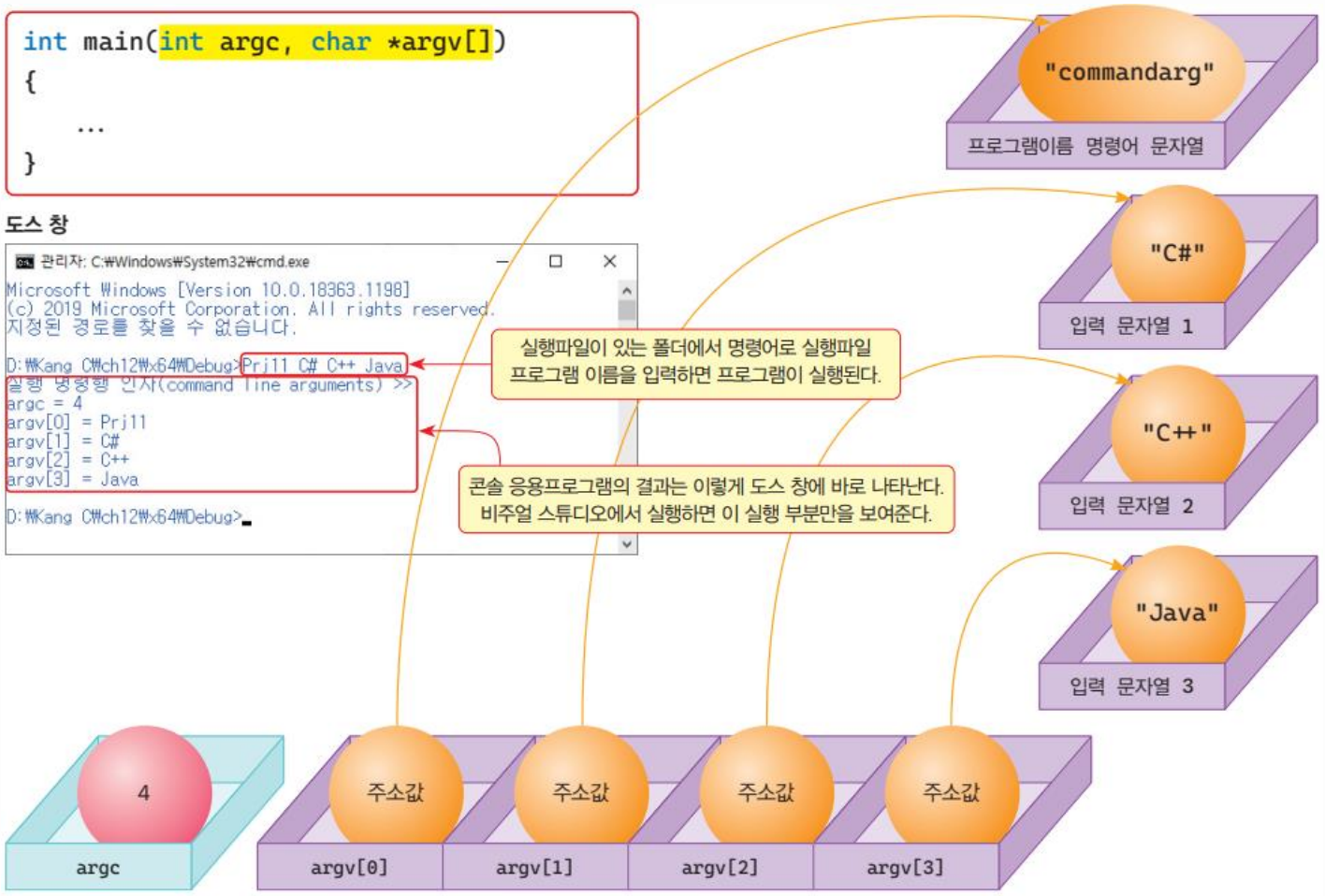
결과

```
실행 명령행 인자(command line arguments) >>
argc = 4
argv[0] = C:\Kang C\ch12\x64\Debug\Prj11.exe
argv[1] = Python
argv[2] = Go
argv[3] = Kotlin
```

비주얼 스튜디오에서 실행하면 전체 경로를 포함한 실행파일의 이름이 첫 번째 인자로 표시된다.

# 명령행 인자 실행과 명령행 인자 전달

- **₩ch12₩x64₩Debug 폴더**
  - 비주얼 스튜디오의 실행파일이 생성된 폴더에서 명령 프롬프트를 실행



# LAB 여러 문자열 처리

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

Lab 12-3 lab3strprocess.c 난이도: ★

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05     char str1[] = "Python";
06     char str2[] = "Kotlin";
07     char str3[] = "Tensorflow";
08
09     char *pstr[] = { ____________________ };
10
11     //각각의 3개 문자열 출력
12     printf("%s ", pstr[0]);
13     printf("%s ", ____________________);
14     printf("%s\n", pstr[2]);
15
16     //문자 출력
17     printf("%c %c %c\n", str1[0], str2[1], str3[2]);
18     printf("%c %c %c\n", pstr____________,
19                                  pstr____________, pstr____________);
20
21     return 0;
22  }
```

정답

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

감사합니다.