프로그래밍 기초

12장. 문자와 문자열 Part 1

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(character): 'A'**
  - 작은 따옴표(single quotation marks)에 의해 표기된 문자 1개
    - 한 글자를 작은 따옴표로 둘러싸서 'A'와 같이 표기
    - C 언어에서 크기 1바이트인 자료형 char로 지원

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

- **문자열(string): "A", "Hello"**
  - 큰 따옴표(double quotation marks)로 묶인 하나의 문자 또는 여러 문자의 모임
    - 일련의 문자 앞 뒤로 큰 따옴표로 둘러싸서 "Hello"로 표기
    - "A" 처럼 문자 하나도 큰 따옴표로 둘러싸면 문자열
    - ~~'ABC'~~처럼 작은 따옴표로 둘러싸도 문자가 될 수 없으며 오류가 발생

```
char str1[] = "Programming";
```
==
```
char str1[] = {'P','r','o','g','r','a','m','m','i','n','g','\0'};
```

```
char str2[] = "C";
```
==
```
char str2[] = {'C', '\0'};
```

문자와 문자열의 선언

- **문자열 선언 및 값 할당**
  - C언어는 문자열을 저장하기 위한 자료형을 따로 제공하지 않음
  - 문자 배열을 선언하여 문자열을 처리: char str[10];
    - 각각의 원소에 문자를 저장
    - 문자열의 마지막을 의미하는 NULL 문자('\0')를 마지막에 저장해야 됨
  - 문자열을 저장하는 배열 크기: char array[문자열길이+1]
    - 반드시 저장될 문자 수보다 1 크도록 선언 후 NULL 문자를 마지막에 추가(또는 자동 추가)
    - 문자열의 마지막에 NULL 문자가 없으면 문자열의 끝을 찾지 못함
  - 배열 csharp의 크기를 3으로 선언한 후 배열 csharp에 문자열 "C#"을 저장
    - 마지막 원소인 csharp[2]에 NULL 문자('\0')를 저장

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

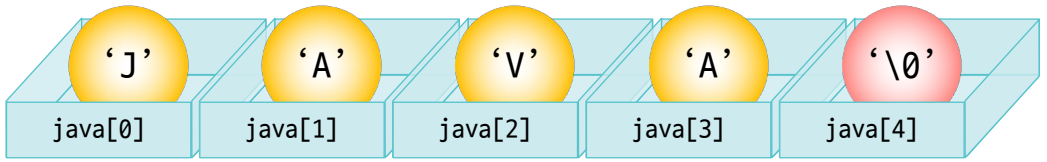
마지막에 NULL 추가

문자열을 선언하는 방법 #1

- **char형 배열에 문자를 하나씩 대입**
  - `char array[] = {'a', 'b', 'c', '\0'}`
    - 중괄호를 사용하여 배열 원소로 문자를 하나씩 입력
    - 마지막 문자로 NULL 문자('\0')를 개발자가 추가
  - 배열 초기화에서 배열 크기를 지정하지 않음
    - 자동으로 크기가 결정됨

```
char java[] = {'J', 'A', 'V', 'A', '\0'};
```

마지막에 '\0'을 빼면, 대입에는 문제가 없으나 출력 등에서 문제가 발생



문자열을 선언하는 방법 #2

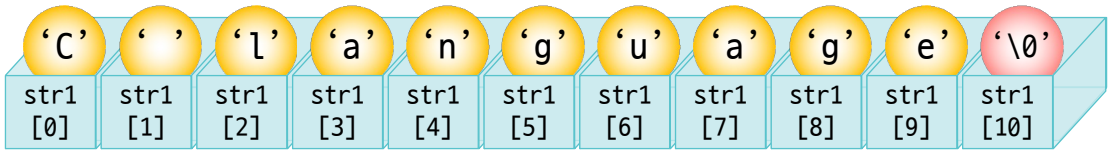
- **배열에 문자열을 직접 대입하는 방법**
  - 배열 초기화 시 배열 크기를 지정하지 않는 것이 편리
    - 배열 크기가 자동으로 결정됨
    - 배열의 마지막에 NULL 문자가 자동으로 추가됨

```
char str1[] = "C language";
```

  - 배열의 크기를 지정하는 경우: 배열[문자수 + 1]로 선언
    - NULL 문자를 저장할 추가 공간이 필요

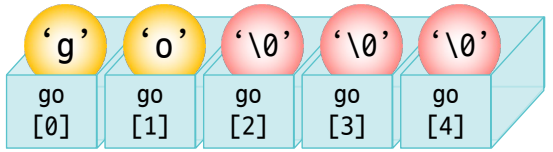
```
char str1[] = "C language";
char str1[11] = "C language";
```

배열 크기를 지정하지 않은 경우:
- 자동으로 마지막에 NULL 문자 추가됨



  - 문자열의 길이(2)가 배열의 크기(5) 보다 작은 경우
    - 빈 공간은 자동으로 NULL 문자로 채워짐

```
char go[5]  = "go";
```

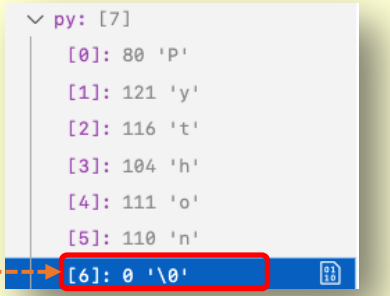


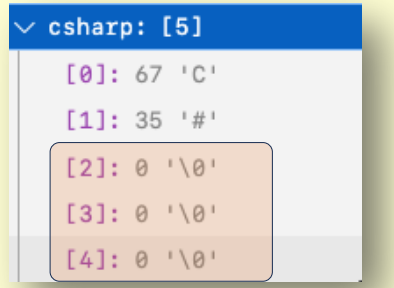
문자와 문자열 출력 (실습)

<01chstrprt.c>

- **문자 출력**
  - 형식제어문자 %c
- **문자열 출력**
  - 형식제어문자 %s
  - 배열 이름: 문자열의 첫 주소
  - printf("%s\n", 배열이름);
- **함수 puts()로 문자열 출력**
  - puts(csharp)
    - 문자열 출력 후 한 줄 띄움
    - printf("%s\n", csharp)와 동일

```
#include <stdio.h>

int main(void)
{
    // 문자 선언과 출력
    char ch = 'A';
    printf("%c %d\n", ch, ch);

    // 문자열 선언 방법1
    char java[] = {'J', 'A', 'V', 'A', '\0'};
    printf("%s\n", java);

    // 문자열 선언 방법2
    char py[] = "Python";  // 배열크기를 생략: NULL('\0') 자동 추가
    printf("%s\n", py);

    // 문자열 선언 방법3
    char csharp[5] = "C#";
    // printf("%s\n", csharp);
    puts(csharp);

    // 문자 배열에서 문자 출력
    printf("%c %c\n", csharp[0], csharp[1]);

    return 0;
}
```





실행 결과

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

문자열 구성하는 문자 참조

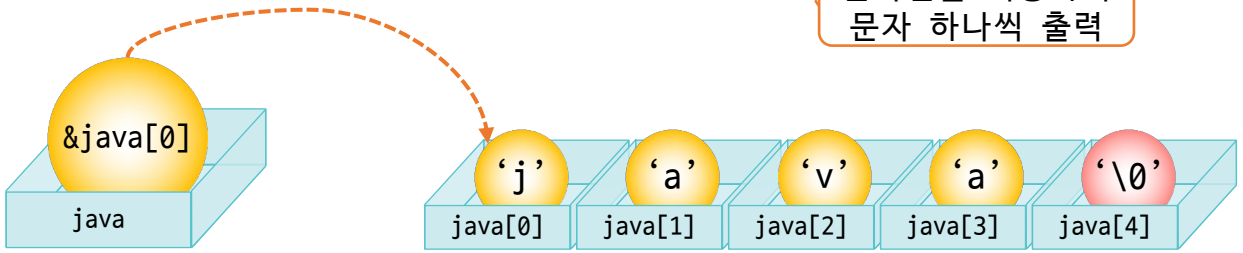
- **문자열 상수를 문자 포인터 변수에 저장**
  - 문자 포인터 변수에 문자열의 주소를 저장
    - 문자열을 구성하는 문자 하나 하나의 수정은 불가능: 문자열 상수는 text(code) segment에 저장
    - 문자열 “java”의 주소를 포인터 변수에 저장

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

  - 문자열 출력: printf()
    - %s: 문자열을 한 번에 모두 출력
    - %c: 문자 하나씩 출력

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

문자열을 이동하며
문자 하나씩 출력



문자열 배열의 시작 주소

문자 포인터로 문자열 저장과 출력

<02charprt.c>

- **변수 char *java = “java”;**
  - 포인터 변수에 문자열의 시작 주소를 저장
    - ➢ 문자열 상수: 마지막에 NULL 자동 추가됨
  - 포인터 변수 java가 가리키는 문자열 수정 불가능

- **문자열 상수(string constant)**
  - 프로그램 소스 코드 내부에 직접 작성된 문자열
    - ➢ 변수의 메모리 할당이 없이 문자열만 선언
  - 문자열 상수는 text(code) segment 영역에 저장
    - ➢ 문자열의 내용은 읽기는 가능하지만 변경 불가능

- **반복문을 이용**
  - 문자가 ‘\0’이 아니면 문자를 출력
    - ➢ 출력할 문자열의 끝을 ‘\0’ 문자로 검사

```c
#include <stdio.h>

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

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

    i = 0;
    while (*(java + i) != '\0') // java[i] 와 같음
    {
        printf("%c", *(java + i));
        i++;
    }
    // 수정 불가능, 실행 결과에 문제 발생
    java[1] = 'J';
    printf("%c", java[1]);

    return 0;
}
```

문자 포인터: 읽기만 가능

읽기만 가능하고 수정 불가능

02charprt.c 실행 후 오류 메시지

- 수정할 수 없는 메모리 접근

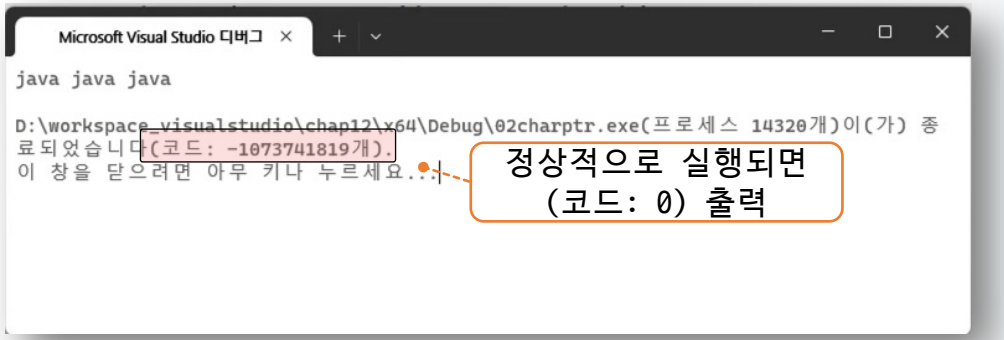
02charptr.c

chap12 > 02charptr.c > main(void)

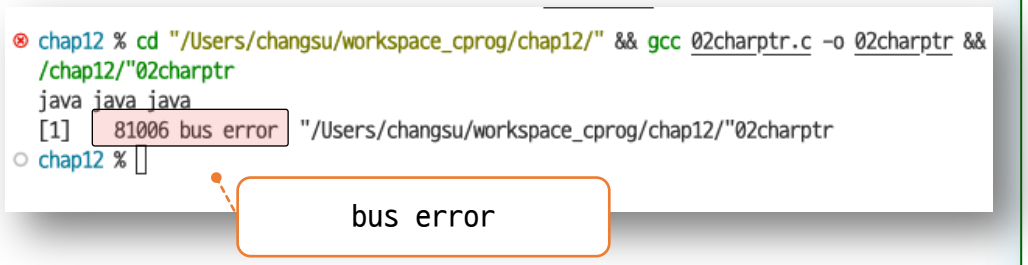
```
3   int main(void)
12          printf("%c", java[i++]);
13      printf("\n");
14
15      i = 0;
16      while (*(java + i) != '\0') // java[i]는 *(java + i)와 같음
17      {
18          printf("%c", *(java + i));
19          i++;
20      }
21      printf("\n");
22
23      // 수정 불가능, 실행 결과에 문제 발생
24      java[1] = 'J';
```

Exception has occurred. ×
EXC_BAD_ACCESS (code=2, address=0x100007fa9)

```
25      printf("%c", java[1]);
26
27      return 0;
28  }
29
```



Widnows11 Visual Studio 2022



Mac: Visual Studio Code

https://ko.wikipedia.org/wiki/버스_오류

문자열 상수와 문자열 변수

- **문자열 포인터(String Pointer)**
  - 문자열 상수가 저장된 공간은 읽기 전용 메모리 영역: text(code) segment
  - char *s1은 해당 문자열 상수의 주소만 저장
    - 수정이 불가능함: code 영역의 주소만 가지고 있음
    - 메모리 공간 할당 없이 문자열의 주소만 저장

- **문자열 배열(String Array)**
  - 문자열 배열 변수: stack segment에 저장
  - char s2[8] = "abcdefg";
    - 문자열 상수값을 자신의 메모리로 복사해서 사용
    - 수정이 가능

```c
#include <stdio.h>
int main()
{
    char *s1 = "abcdefg";   // 문자열 포인터
    char s2[8] = "abcdefg"; // 문자열 배열

    printf("s1: %p\n", s1);
    printf("s2: %p\n", s2);
    return 0;
}
```

NULL('\0') 문자에 의한 문자열 분리

- **함수 printf("%s", str)**
  - 문자열에서 NULL 문자까지 하나의 문자열로 인식

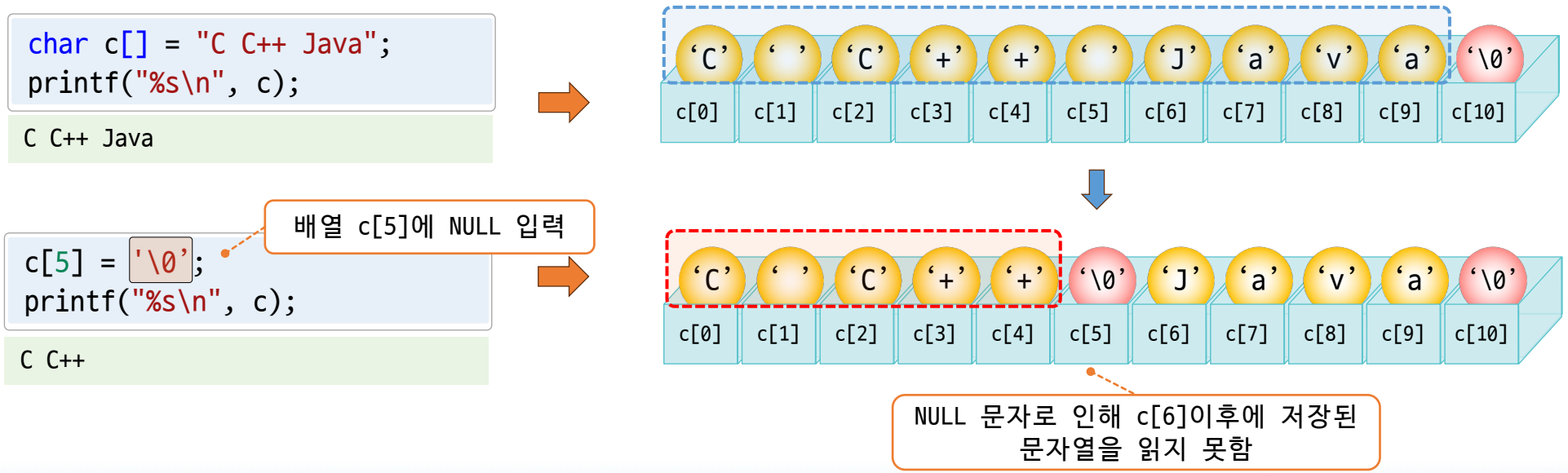
- **만일 배열 c에 c[5] = '\0'; NULL 문자로 변경**
  - c[5]에 저장된 '\0' 문자에 의해 c가 가리키는 문자열은 "C C++"까지만 인식
    - ➢ 문자열은 시작 문자부터 '\0' 문자가 나올 때까지 하나의 문자열로 처리
  - (c+6)로 문자열을 출력하면 "Java"가 출력

```
char c[] = "C C++ Java";
printf("%s\n", c);
```

```
C C++ Java
```

```
c[5] = '\0';
printf("%s\n", c);
```

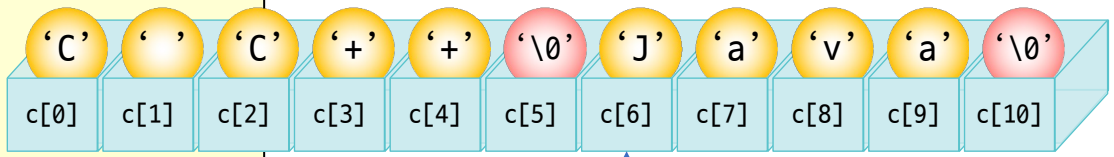
```
C C++
```

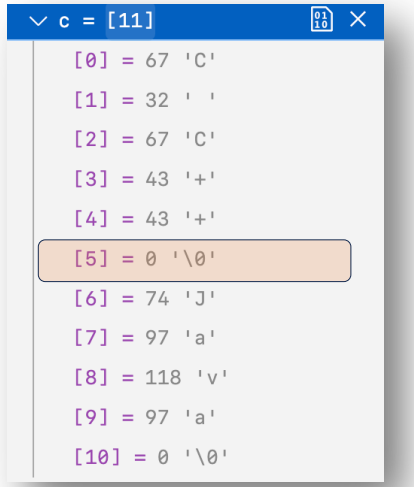


문자열 중간에 '\0'을 삽입해 문자열 분리

<03strsplit.c>

```c
#include <stdio.h>

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

    // 문자 배열의 각 원소를 하나 하나 출력하는 방법
    c[5] = ' '; // NULL 문자를 빈 문자로 바꾸어 문자열 복원
    char *p = c;

    while (*p != '\0') // while (*p) 도 가능
        printf("%c", *p++);
    printf("\n");

    return 0;
}
```



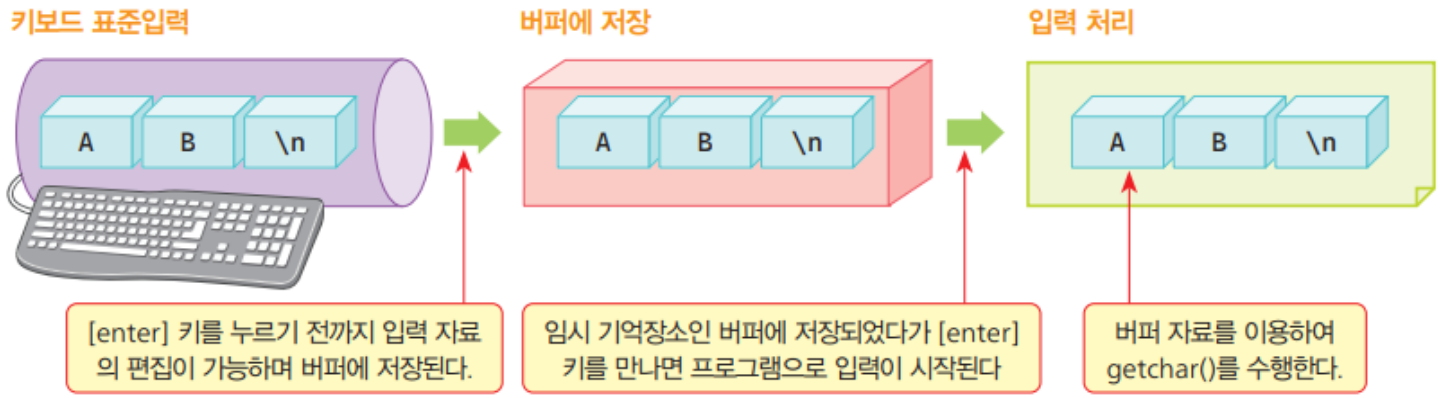



```
C C++ Java
C C++
Java
C C++ Java
```

다양한 문자 입출력

- **함수 getchar(), putchar()**
  - 문자의 입력과 출력에 사용: GCC 표준 함수

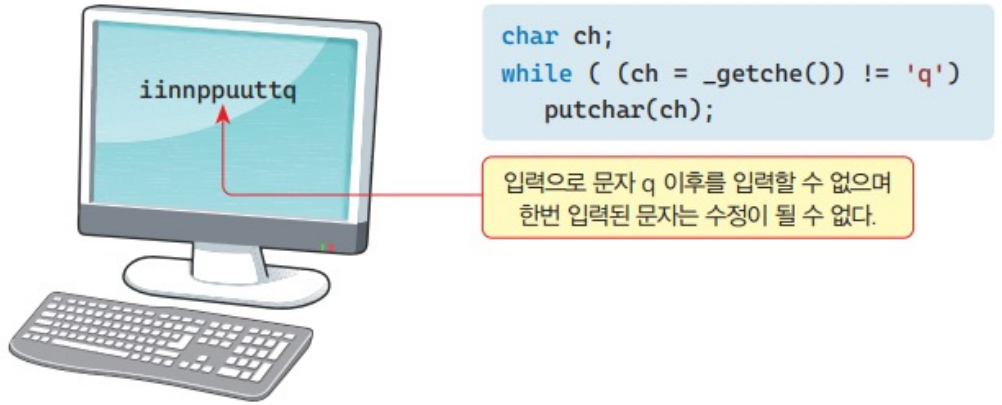
- **라인 버퍼링(line buffering) 방식**
  - 함수 getchar()에서 문자 하나를 입력해도 반응을 보이지 않음
  - [Enter] 키를 누르면 이전에 입력한 문자마다 입력이 실행
    - 입력한 문자는 임시 저장소인 버퍼(buffer)에 저장
    - [Enter] 키를 만나면 getchar() 함수는 버퍼에서 문자를 읽기 시작
  - 즉각적(interactive)인 입력을 요구하는 시스템에서는 사용이 불가능



버퍼를 사용하지 않고 문자를 입력하는 함수 _getche(): Windows

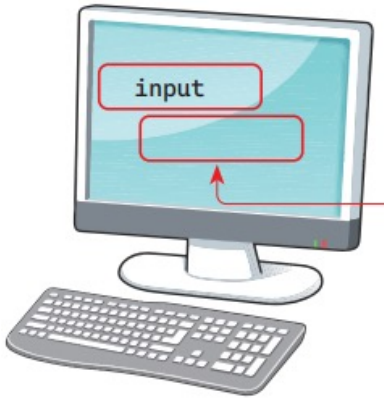
- **함수 _getche(): Windows 전용 함수**
  - 버퍼를 사용하지 않으므로 문자 하나를 입력하면 바로 함수 _getche()가 실행
  - 함수 _getche()에서 입력된 문자는 바로 모니터에 표시
    - 마지막 e는 입력 문자를 표시하다는 echo를 의미
  - 헤더파일 conio.h를 삽입 필요: Windows에서만 사용

- **입력 문자가 'q'가 아니면**
  - 함수 putchar()에 의하여 문자가 바로 출력
  - 입력된 문자도 보이고, 바로 putchar()에 의하여 출력
    - 입력 문자가 "inputq"
    - 화면에는 "iinnppuuttq"가 표시



입력한 문자가 화면에 보이지 않는 함수 _getch(): Windows

- **함수 _getch(): Windows 전용 함수**
  - 입력된 문자를 출력 함수로 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	

```
int getchar();   // GCC
int _getche();
int _getch();

int putchar(int c);  // GCC
int _putchar(int c);
```

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

<04getch.c>

```c
#include <stdio.h>
#include <conio.h> // Windows

int main(void)
{
    char ch;

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

    printf("\n문자를 누른 것이 보이고, _putch에 의해 입력문자
           출력 >>\n");
    while ((ch = _getche()) != 'q')
        _putch(ch); // conio.h: Windows 만 사용

    printf("\n문자를 누른 것이 안 보이고, _putch에 의해 입력문자
           출력 >>\n");
    while ((ch = _getch()) != 'q')
        _putch(ch); // conio.h
    printf("\n");

    return 0;
}
```

GCC 표준

Windows 전용

마지막에 q를 입력해야 종료되며, q는 출력 안됨

실행 결과

```
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
```

문자열의 입출력 (실습)

- **함수 scanf("%s", buf)는 공백으로 구분되는 하나의 문자열을 입력**
  - 입력 받은 문자열이 저장될 충분한 공간인 문자 배열(char buf[])을 선언해야 됨
    - 단순히 문자 포인터(char *)로는 문자열 저장 불가능: 메모리 공간이 없기 때문
  - 함수 scanf("%s", buf): "%s"를 사용하여 문자열 입력
    - 공백 이전까지 하나의 문자열로 인식해서 buf에 저장
  - 함수 printf("%s", buf): "%s"를 사용하여 문자열을 출력

<05scanf_string.c>

```c
#include <stdio.h>

int main()
{
    char buf[100] = {0};

    printf("Type a string: ");
    scanf("%s", buf);

    printf("buf: %s\n", buf);
    return 0;
}
```

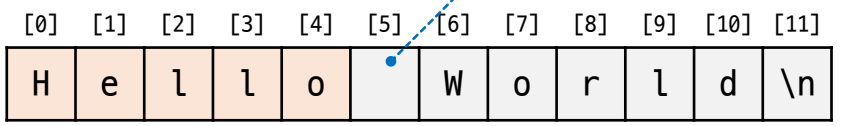
문자열을 저장할 char형 배열 선언

배열 이름이 배열의 시작 주소: buf만 전달

실행 결과 #1

```
Type a string: Hello Enter↵
buf: Hello
```

scanf("%s", buf)는 공백 이전까지 하나의 문자열로 인식

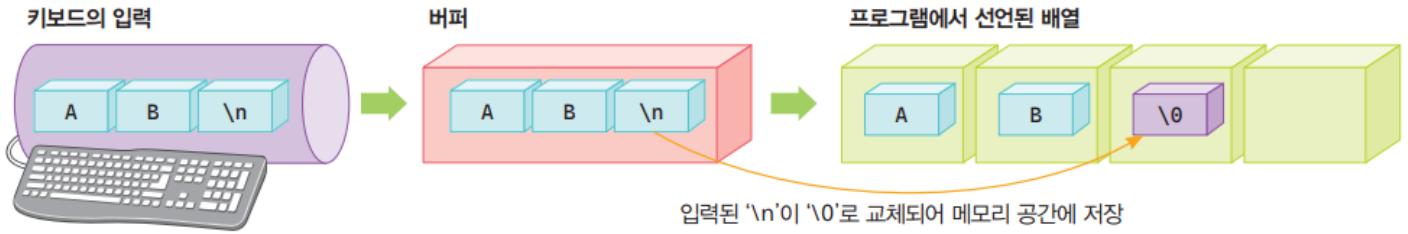


실행 결과 #2

```
Type a string: Hello World Enter↵
buf: Hello
```

한 줄의 문자열 입력 함수: gets()/fgets()

- **gets()/fgets(): [Enter] 키를 누를 때까지 한 행을 버퍼에 저장한 후 입력 처리**
  - 함수 gets()는 마지막에 입력된 '\n'이 '\0'로 교체되어 배열에 저장
    - '\n'은 buffer에 저장되지 않음, 프로그램에서 한 행을 하나의 문자열로 간주



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);
```

Visual Studio 에서만 사용 가능

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

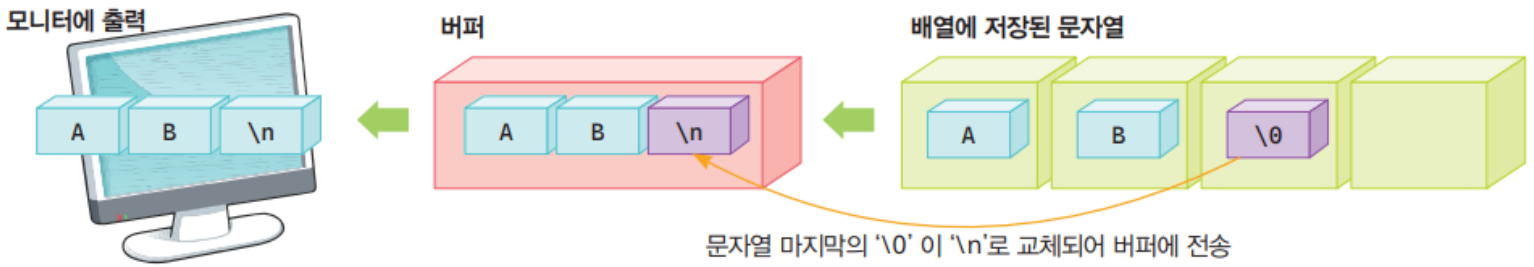
```
int puts(const char * str);
```

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

문자열 입출력 함수: gets(), puts()

- **함수 gets()의 인자**
  - 입력된 문자열을 저장할 수 있는 충분한 공간의 문자 배열을 사용
    - gets(char *str): 마지막에 '\n'은 '\0' 로 교체되어 저장
    - fgets(char *str, int size, FILE* stream): 마지막에 '\n' 및 '\0' 추가

- **문자열 출력함수 puts()**
  - 한 줄의 문자열 출력 함수
  - gets()와 반대로 문자열의 마지막에 저장된 '\0'를 '\n'로 교체하여 버퍼에 전송
    - 문자열 출력 후 자동으로 한 줄 띄움('\n')



화면 문자열 입력 함수: gets()/ fgets()

- **gets()/fgets() 함수 비교**
  - 공백을 포함한 한 줄의 문자열 입력 함수
  - Enter키('\n')를 누를 때까지 한 줄 전체를 입력 받음
  - gets_s(): Windows Visual Studio에서 사용 가능

	char *gets(char *buffer)	fgets(char *buffer, int size, FILE *stream)
개행문자('\n') 저장 여부	• 개행 문자('\n')는 저장 안함 • 마지막에 NULL 자동 추가	• 개행 문자('\n') 및 NULL('\0')이 자동 추가됨
버퍼 크기 제한	• 버퍼의 크기 제한이 없음(안전하지 않음)	• buffer의 크기 제한(int size)
사용 권장 여부	• 더 이상 권장하지 않음	• 권장
사용 예제	`char buf[100];` `gets(buf);  // 입력 문자열의 크기 제한 없음`	`char buf[100];` `fgets(buf, sizeof(buf), stdin); // stdin: 화면 입력`

gets()/fgets() 예제 (실습)

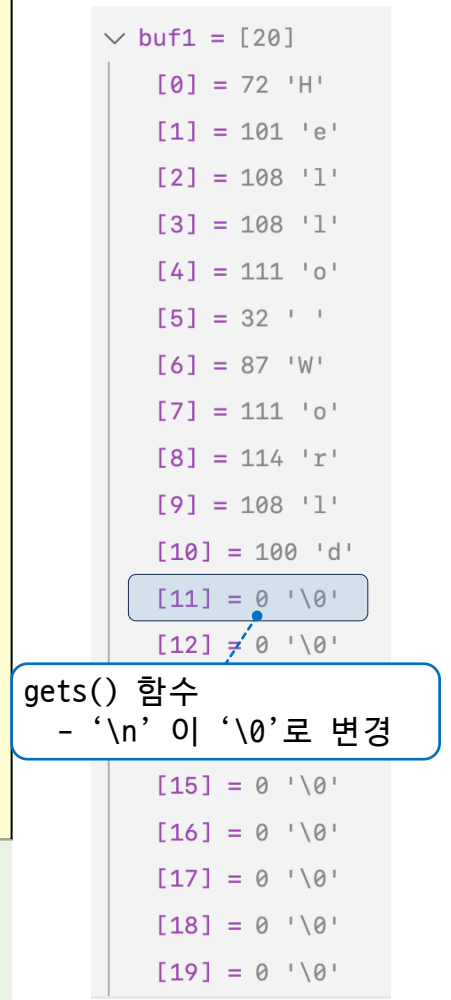
<05gets_fgets.c>

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char buf1[20] = {0};
    char buf2[20] = {0};

    printf("Type a sentence(gets): ");
    gets(buf1);
    puts(buf1);

    printf("Type a sentence(fgets): ");
    fgets(buf2, sizeof(buf2), stdin); // stdin: 화면 입력
    printf("%s, len: %d\n", buf2, strlen(buf2));

    buf2[strlen(buf2) - 1] = '\0'; // '\n' 제거
    printf("%s", buf2);
    return 0;
}
```

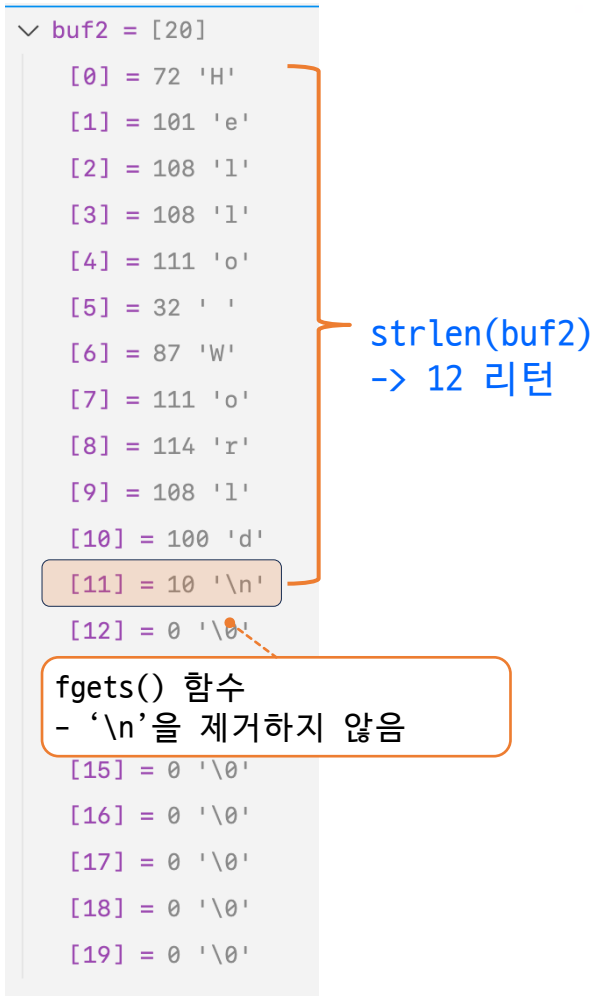
gets() 사용 권장하지 않음

strlen(문자열)
- 문자열의 길이 리턴

```
warning: this program uses gets(), which is unsafe.
Type a sentence(gets): Hello World Enter
Hello World
Type a sentence(fgets): Hello World Enter
Hello World'\n'
, len: 12
Hello World%
```



gets() 함수
- '\n' 이 '\0'로 변경



strlen(buf2)
-> 12 리턴

fgets() 함수
- '\n'을 제거하지 않음

기호 상수 EOF(End of File)

- **기호 상수 EOF(End Of File)**
  - #define EOF (-1)
    - 파일의 끝이라는 의미로 stdio.h 헤더파일에 정수 -1로 정의
  - gets(), getchar(), scanf() 등 입력 함수가 더 이상 읽을 데이터가 없을 때 EOF 반환

- **EOF 강제 생성 방법**
  - 사용자가 실행 중인 프로그램에 EOF를 강제로 생성할 때 사용
  - Ctrl + Z
    - Windows 운영체제:
  - Ctrl + D
    - Linux/Unix 운영체제

gets(), puts()

<05strinput.c>

- **함수 scanf()에서 제어문자 %s**
  - 문자열을 입력: 공백으로 분리됨
- **함수 fgets()와 gets_s()를 사용**
  - 공백을 포함한 한 라인의 문자열 입력
- **while문을 사용**
  - 한 줄의 문자열을 입력
    - ➢ 입력 받은 문자열 출력
  - 반복을 종료: EOF 강제 생성
    - ➢ Windows: Ctrl + Z 입력
    - ➢ Mac: Ctrl + D 입력
- **함수 printf()와 scanf()**
  - 다양한 입출력에 적합
- **함수 puts()와 fgets()**
  - 한 줄 단위 입출력에 편리

```c
#include <stdio.h>

int main(void)
{
    char name[20]={0}, dept[30]={0}; // 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\n", snum, dept, name);
    getchar(); // 개행 문자 제거

    char line[101]={0}; // char *line 으로는 오류발생
    printf("한 행에 학번 이름 학과 입력한 후 [Enter]\n");
    printf("새로운 행에서 (Ctrl + Z, Ctrl+D)를 누르십시요.\n");

    // while (gets_s(line, 101)) // Visual studio
    while (fgets(line, 101, stdin)) // 표준 GCC
        puts(line);

    printf("Exit program\n");
    return 0;
}
```

다양한 형태의 입출력에 적합

Line 단위 입출력에 편리

gets(), puts() 실행 결과

```
학번 이름 학과 입력 >>
1 Hong Computer      Enter↵
출력: 1 Hong Computer

한 행에 학번 이름 학과 입력한 후 [enter]를 누르고 새로운 행에서 (ctrl + Z)를 누르십시요.
2 Kim Computer Engineering      Enter↵
2 Kim Computer Engineering

3 Lee Math
3 Lee Math
^Z   Ctrl + Z
Exit program
```

Lab 한 행을 표준입력으로 받아 문자 하나씩 출력 (실습)

- **함수 fgets() 사용**
  - 한 행의 표준 입력
  - 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

<lab1lineprt.c>

```c
#include <stdio.h>

int main()
{
    char s[100];
    // 문자배열 s에 표준 입력한 한 행을 저장
    printf("Input a sentence: ");
    fgets(s, 100, stdin);

    // 문자배열에 저장된 한 행을 출력
    char *p = s;
    while (*p != '\0')
    {
        printf("%c", *p);
        p++;
    }
    printf("\n");

    return 0;
}
```

```
Input a sentence: C programming language
C programming language
```

Questions?