12주차 1차시 문자열 관련 함수

【학습목표】

1. 스트림과 데이터의 이동을 구분할 수 있다.
2. 표준 입출력과 버퍼를 구분할 수 있다.

학습내용1 : 스트림과 데이터의 이동

• stdin	표준 입력 스트림	키보드 대상으로 입력
• stdout	표준 출력 스트림	모니터 대상으로 출력
• stderr	표준 에러 스트림	모니터 대상으로 출력

✓ stdin과 stdout은 각각 표준 입력 스트림과 표준 출력 스트림을 의미하는 이름들이다.
✓ stderr은 표준 에러 스트림이라 하며, 출력의 대상은 stdout과 마찬가지로 모니터이다.
✓ 출력 리다이렉션이라는 것을 통해서 stdout과 stderr이 향하는 데이터 젂송의 방향을 각각 달리 할 수 있다.
✓ stdin, stdout, stderr은 모두 프로그램 시작과 동시에 자동으로 형성되고 프로그램 종료시 자동으로 소멸된다.
✓ 이외의 스트림들은 프로그래머가 직접 형성해야 한다. 예를 들어 파일 입출력을 위한 스트림은 직접 형성해야 한다.
스트림이라 불리는 이유는 데이터의 이동을 한 방향으로만 형성하기 때문이다.
물이 한 방향으로 흐르듯 스트림도(스트림은 물의 흐름을 의미함)
한 방향으로만 데이터가 이동한다.

학습내용2 : 입출력 함수

1. 문자 입출력 함수

■ 하나의 문자를 출력하는 두 함수

```
#include <stdio.h>
int putchar(int c);
int fputc(int c, FILE * stream);
```

putchar함수는 인자로 전달된 문자를 모니터에 출력한다.
fputc 함수의 두번째 인자를 통해서 출력의 대상을지정한다.

➜ 함수호출 성공 시 쓰여진 문자정보가, 실패 시 EOF 반환

fputc의 두 번째 인자로 stdout이 전달되면 이 putchar 함수와 동일한 결과를 보인다.

■ 하나의 문자를 입력 받는 두 함수

```
#include <stdio.h>
int getchar(void); 키보드로 입력된 문자의 정보를 반환한다.
int fgetc(FILE * stream);  문자를 입력 받을 대상정보를 인자로 전달한다.
    ➜ 파일의 끝에 도달하거나 함수호출 실패 시 EOF 반환
```

getchar 함수와 fgetc 함수의 관계는 putchar 함수와 fputc 함수의 관계와 같다.

2. 문자 입출력 관련 예제

```
int main(void)
{
    int ch1, ch2;

    ch1=getchar();   // 문자 입력
    ch2=fgetc(stdin);   // 엔터 키 입력

    putchar(ch1);   // 문자 출력
    fputc(ch2, stdout);   // 엔터 키 출력
    return 0;
}
```

```
p
p
```

첫번째 P는 입력이 된 P, 두번째 P는 출력된 P

문자의 입력을 완성하는 엔터키의 입력도 하나의 문자로 인식이 된다.
따라서 이 역시도 입출력이 가능하다.
문자를 int형 변수에 저장하는 이유는 EOF를 설명하면서 함께 설명한다.

3. 문자열 입출력에서의 EOF

√ EOF의 의미
- ▶ EOF는 End Of File의 약자로서, 파일의 끝을 표현하기 위해서 정의해 놓은 상수이다.
- ▶ 파일을 대상으로 fgetc 함수가 호출되었을 때 파일에 끝에 도달을 하면 EOF가 반환된다.

√ 콘솔 대상의 fgetc, getchar 함수호출로 EOF를 반환하는 경우
- ▶ 함수호출의 실패
- ▶ Windows에서 Ctrl+Z 키, Linux에서 Ctrl+D 키가 입력이 되는 경우

4. 반환형이 int이고, int형 변수에 문자를 담는 이유는?

```
int getchar(void);
int fgetc(FILE * stream);
```

✓ 반환형이 char형이 아닌 int형인 이유는?

- ▶ char형은 예외적으로 signed char가 아닌 unsiged char로 표현하는 컴파일러가 존재한다.
- ▶ 파일의 끝에 도달했을 때 반환하는 EOF는 -1로 정의되어 있다.
- ▶ char를 unsigend char로 표현하는 컴파일러는 EOF에 해당하는 -1을 반환하지 못한다.
- ▶ int는 모든 컴파일러가 signed int로 처리한다. 따라서 -1의 반환에 무리가 없다.

5. 문자열 출력 함수: puts, fputs

```
#include <stdio.h>
int puts(const char * s);
int fputs(const char * s, FILE * stream);
    ➡ 성공 시 0이 아닌 값을, 실패 시 EOF 반환
```

인자로 전달되는 문자열을 출력한다. 단 fputs 함수는 두 번째 인자를 통해서 출력의 대상을 지정할 수 있다.

```
int main(void)
{
    char * str="Simple String";

    printf("1. puts test ------ \n");
    puts(str);
    puts("So Simple String");

    printf("2. fputs test ----- \n");
    fputs(str, stdout); printf("\n");
    fputs("So Simple String", stdout); printf("\n");

    printf("3. end of main ----\n");
    return 0;
}
```

```
1. puts test ------
Simple String
So Simple String
2. fputs test -----
Simple String
So Simple String
3. end of main ----
```

puts 함수가 호출되면 문자열 출력 후 자동으로 개행이 이뤄지지만, fputs 함수가 호출되면 문자열 출력 후 자동으로 개행이 이뤄지지 않는다는 사실에 주목!

5. 문자열 입력함수: gets, fgets

```
#include <stdio.h>
char * gets(char * s);
char * fgets(char * s, int n, FILE * stream);
```

➡ 파일의 끝에 도달하거나 함수호출 실패 시 NULL 포인터 반환

```
int main(void)
{
    char str[7];   // 7바이트의 메모리 공간 할당
    gets(str);   // 입력 받은 문자열을 배열 str에 저장
    . . . .
}
```

이 경우 입력되는 문자열의 길이가 배열을 넘어설 경우 할당 받지 않은 메모리를 참조하는 오류가 발생한다.

```
int main(void)
{
    char str[7];
    fgets(str, sizeof(str), stdin);
    . . . .  // stdin으로부터 문자열 입력 받아서 str에 저장
}
```

stdin으로부터 문자열을 입력 받아서 str에 저장하되 널 문자를 포함하여 sizeof(str)의 크기 만큼 저장을 해라.

6. fgets 함수 호출의 예

```
int main(void)
{
    char str[7];
    int i;

    for(i=0; i<3; i++)
    {
        fgets(str, sizeof(str), stdin);
        printf("Read %d: %s \n", i+1, str);
    }
    return 0;
}
```

실행 1

```
12345678901234567890
Read 1: 123456
Read 2: 789012
Read 3: 345678
```

6개의 문자씩 끊어서 읽히고 있다. 즉, 한번의 fgets 함수호출당 최대 6개의 문자만 읽혀진다.

실행 2

```
We
Read 1: We


like
Read 2: like


you
Read 3: you
```

엔터키의 입력도 문자열의 일부로 받아들임을 보임.

실행 3

```
Y & I
Read 1: Y & I

ha ha
Read 2: ha ha

^^ --
Read 3: ^^ --
```

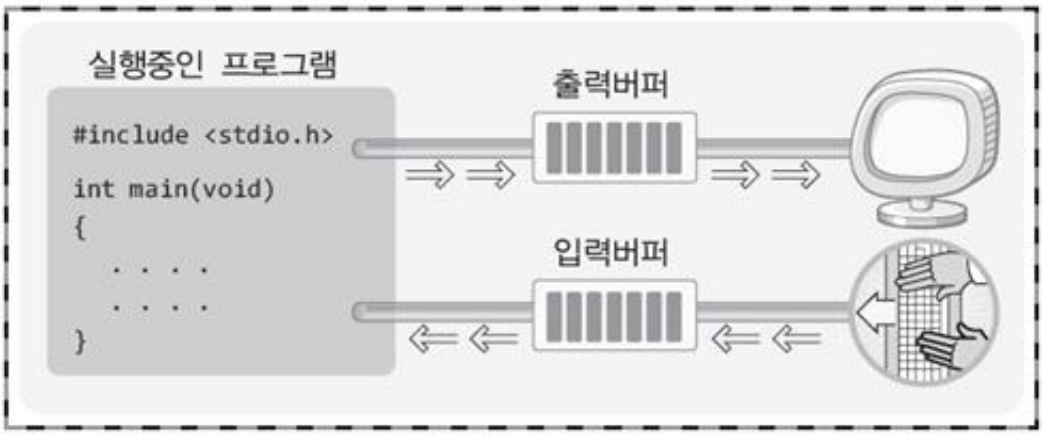
공백을 포함하는 문자열을 읽어 들임을 보임

학습내용3 : 표준 입출력과 버퍼

1. 표준 입출력 기반의 버퍼와 버퍼링의 이유

√ 입출력 버퍼
- ▶ 버퍼는 특정 크기의 메모리 공간을 의미한다.
- ▶ 운영체제는 입력과 출력을 돕는 입출력 버퍼를 생성하여 제공한다.
- ▶ 표준 입출력 함수를 기반으로 데이터 입출력 시 입출력 버퍼를 거친다.

√ 입출력 버퍼에 데이터가 전송되는 시점
- ▶ 호출된 출력함수가 반환이 되는 시점이 출력버퍼로 데이터가 완전히 전송된 시점이다.
- ▶ 엔터를 입력하는 시점이 키보드로 입력된 데이터가 입력버퍼로 전달되는 시점이다.



버퍼링을 하는 이유는 데이터 이동의 효율과 관련이 있다. 데이터를 모아서 전송하면, 하나씩 전송하는 것보다 효율적이다.

2. 출력버퍼를 비우는 fflush 함수

```
#include <stdio.h>
int fflush(FILE * stream);
```

➜ 함수호출 성공 시 0, 실패 시 EOF 반환

▶ 인자에 해당하는 출력버퍼를 비운다.
출력버퍼를 비운다는 것은 출력버퍼에 저장된 데이터를 지우는 것이 아니라, 출력버퍼에 저장된 데이터를 목적지로 최종 전송함을 뜻한다.

▶ fflush(stdout) → 출력버퍼를 지워라!
출력버퍼의 경우와 달리 입력버퍼의 비움은 입력버퍼에 저장된 데이터의 소멸을 뜻한다. 그리고 fflush 함수는 출력버퍼를 대상으로 정의된 함수이다. 따라서 fflush(stdin)과 같은 형태의 함수호출은 그 결과를 보장받지 못한다. 그렇다면 입력버퍼는 어떻게 비워야 할까?

3. 입력버퍼는 어떻게 비워야 하나요?

```
int main(void)
{
    char perID[7];
    char name[10];
    fputs("주민번호 앞 6자리 입력: ", stdout);
    fgets(perID, sizeof(perID), stdin);
    fputs("이름 입력: ", stdout);
    fgets(name, sizeof(name), stdin);
    printf("주민번호: %s \n", perID);
    printf("이름: %s \n", name);
    return 0;
}
```

주민번호 앞 6자리 입력: 950915 실행결과1
이름 입력: 주민번호: 950915
이름:

엔터 키가 남아서 문제가 되는 상황

주민번호 앞 6자리 입력: 950709-1122345 실행결과2
이름 입력: 주민번호: 950709
이름: -1122345

말 안 듣는 사람들 때문에 문제되는 상황

입력버퍼를 비우는 함수

```
void ClearLineFromReadBuffer(void)
{
    while(getchar()!='\n');
}
int main(void)
{
    char perID[7];
    char name[10];

    fputs("주민번호 앞 6자리 입력: ", stdout);
    fgets(perID, sizeof(perID), stdin);
    ClearLineFromReadBuffer();   // 입력버퍼 비우기

    fputs("이름 입력: ", stdout);
    fgets(name, sizeof(name), stdin);

    printf("주민번호: %s\n", perID);
    printf("이름: %s\n", name);
    return 0;
}
```

어떠한 경우에도 주민번호 6자리만 입력 받도록 재 구현된 예제

【학습정리】

1. 입출력 버퍼에 데이터가 전송되는 시점은 호출된 출력함수가 반환이 되는 시점이 출력버퍼로 데이터가 완전히 전송된 시점이다. 그리고 엔터를 입력하는 시점이 키보드로 입력된 데이터가 입력버퍼로 전달되는 시점이다.

2. 출력버퍼를 비운다는 것은 출력버퍼에 저장된 데이터를 지우는 것이 아니라, 출력버퍼에 저장된 데이터를 목적지로 최종 전송함을 뜻한다.

3. 출력버퍼의 경우와 달리 입력버퍼의 비움은 입력버퍼에 저장된 데이터의 소멸을 뜻한다. 그리고 fflush 함수는 출력버퍼를 대상으로 정의된 함수이다. 따라서 fflush(stdin) 과 같은 형태의 함수호출은 그 결과를 보장받지 못한다.