chapter 11-1 참고자료.

파일 입출력

박 종 혁

Tel: 970-6702
Email: jhpark1@snut.ac.kr

출처: 뇌를 자극하는 C프로그래밍, 한빛미디어

❑ 파일 입출력의 개념

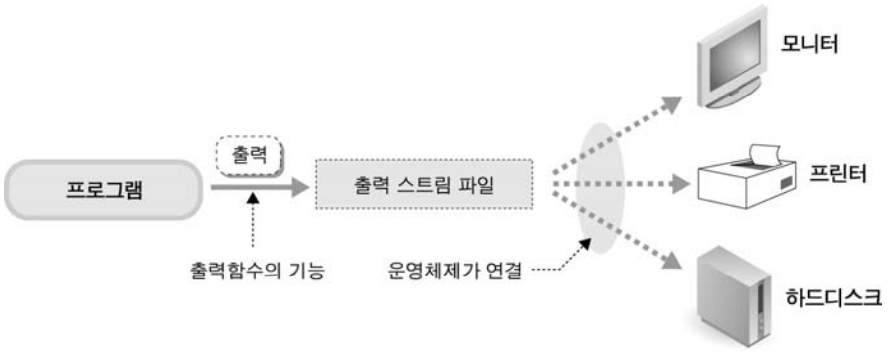
- 파일은 데이터를 입출력하는 모든 대상을 의미한다.

  - 키보드로부터 데이터를 입력하고 모니터로 출력하는 것은 키보드파일과 모니터파일로 데이터를 입출력하는 것이다.

  - 프로그램은 사실상 스트림 파일(stream file)이라고 하는 표준화된 형태의 파일로 입출력을 수행하고 이 파일이 다시 물리적인 장치와 연결되어 실제적인 입출력이 수행된다.

▶ 스트림 파일을 사용하는 이유

- 입출력 함수들이 다양한 입출력장치와 독립적으로 일관된 입출력 작업을 해야 한다(입출력 장치는 항상 변한다).



- 프로그램에서 데이터를 처리하는 속도와 입출력 장치에서 수행되는 입출력 속도의 차이를 줄이는 역할을 한다.
  - 하드디스크의 처리속도는 메모리의 전기적 처리속도를 따라갈 수 없다.
  - 스트림파일은 버퍼(buffer)를 사용하여 속도차이를 줄인다.

❑ 파일 개방

- 스트림 파일을 만드는 것을 파일 개방이라고 하며 fopen 함수를 사용하여 수행한다.

```
FILE *fopen(char *, char *);      // file open, 파일 개방
```

- 출력 전용으로 사용할 파일을 개방하는 예

```
fopen( "a.txt", "w" );   // 파일 개방 함수 호출
```

개방할 파일 이름 ····· "a.txt"

"w" ····· 출력 전용(write)으로 개방

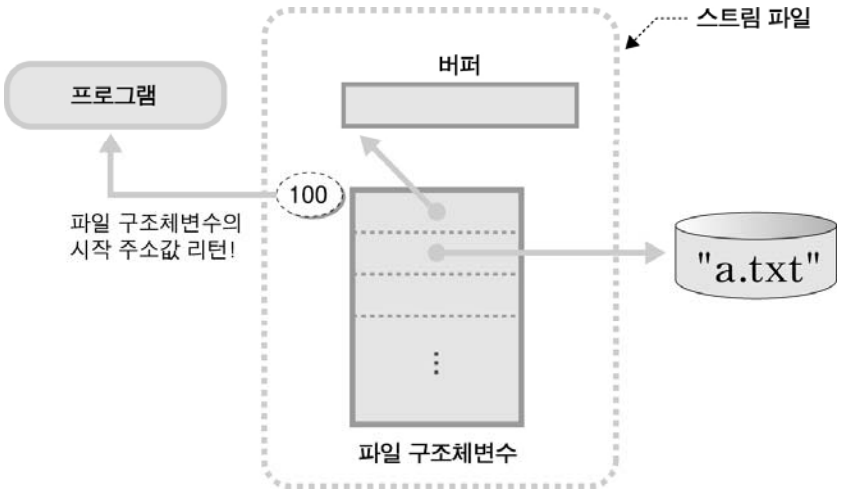
- 개방할 파일은 현재의 작업 디렉토리에서 찾으며 경로를 직접 지정할 수도 있다.

```
fopen("c:₩₩source₩₩a.txt", "w");
// 디렉토리를 표시하는 백슬래시는 문자열 안에 있으므로 두 번 사용한다.
```

▶ fopen함수의 리턴값은 무엇인가?

- fopen함수가 개방에 성공하면 스트림파일을 만들고 파일포인터를 리턴한다.

- 스트림파일은 데이터를 저장하는 버퍼와 버퍼를 관리하는 여러 정보를 파일 구조체변수에 저장하고 있는데 이 구조체변수의 포인터가 파일포이터이다.




```
struct _iobuf{
    char  *_ptr;
    int  _cnt;
    char  *_base;
    int  _flag;
    int  _file;
    int  _charbuf;
    int  _bufsize;
    char  *tmpfname;
};
```

typedef struct _iobuf FILE;

▶ fopen함수의 리턴값은 무엇인가?

- 파일포인터를 포인터변수에 저장하면 입출력 준비작업이 끝난다.

```
FILE *fp;                       // FILE구조체를 가리키는 포인터변수
fp = fopen("a.txt", "w");       // 파일포인터를 포인터변수에 저장한다.
```

- 파일 개방에 실패하면 fopen함수는 널 포인터를 리턴한다.

  – 널 포인터를 사용하면 실행할 때 에러가 발생하므로 반드시 개방에 성공했는지를 검사해야 한다.

```
fp = fopen("b.txt", "r");           // 읽기 전용으로 파일 개방
if(fp == NULL){                     // 파일이 개방되지 않았으면 조건식은 참
    printf("파일이 없습니다.");     // 안내 메시지를 출력하고
    return 1;                       // 프로그램을 종료한다.
}
```

▶ 입력과 출력으로 사용할 두 개의 파일을 개방하는 프로그램

```
#include <stdio.h>                          // FILE구조체에 대한 형 선언이 포함되어 있다.
int main()
{
    FILE *ifp, *ofp;                        // FILE구조체 포인터변수 선언
    ifp=fopen("a.txt", "r");                // a.txt 파일을 읽기 전용으로 개방
    if(ifp==NULL){                          // 파일이 없으면 조건식은 참
        printf("입력파일이 개방되지 않았습니다.₩n");      // 안내 메시지 출력
        return 1;                                       // 프로그램 종료
    }
    printf("입력파일이 개방되었습니다.₩n");
    ofp=fopen("b.txt", "w");                // b.txt 파일은 쓰기 전용으로 개방한다.
    if(ofp==NULL){
        printf("출력파일이 개방되지 않았습니다.₩n");
        return 1;
    }
    printf("출력파일이 개방되었습니다.₩n");
    return 0;
}
```

▶ 개방 모드

- 파일은 사용 용도에 맞게 적절한 모드로 개방한다.

개방 모드	파일이 있을 때	파일이 없을 때
r	읽기 위해 개방	널 포인터 리턴
w	파일의 내용을 삭제하고 쓰기 위해 개방	새로운 파일 생성
a	파일의 끝에 추가하기 위해 개방	새로운 파일 생성

- 출력전용 모드는 같은 이름의 파일이 있을 때 그 내용을 모두 삭제하고 개방하므로 주의해야 한다.

  – 일단 읽기전용 모드로 개방한 후에 파일 존재여부를 확인하고 다시 출력전용으로 개방한다.

```
ifp=fopen(“a.txt”, “r”);          // 일단 읽기 전용으로 개방한다.
if(ifp==NULL){                    // 파일이 없으면 조건식은 참
    ofp=fopen(“a.txt”, “w”);      // 이 때 다시 쓰기 전용으로 개방한다.
}
```

▶ 개방한 파일은 fclose함수로 닫는다.

- 사용이 끝난 파일은 파일을 닫아서 스트림파일을 제거한다.

```
int fclose(FILE *);     // file close, 파일 종결
```

- 성공적으로 닫으면 0을 리턴하며 오류가 발생하면 -1을 리턴한다.

```
FILE *fp;
int res;                        // fclose함수의 리턴값을 저장할 변수
fp=fopen("a.txt", "r");         // 파일 개방
...
res=fclose(fp);                 // 파일포인터변수 fp를 전달인자로 주고 파일을 닫는다.
if(res!=0){
    printf("파일이 닫히지 않았습니다.₩n");
    return 1;
}
```

- 개방된 파일은 프로그램이 종료되면 자동으로 닫히면서 메모리에서 제거 되지만 안정성을 위해서 명시적으로 닫는 것이 좋다.

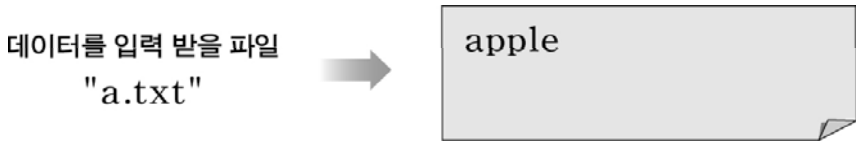
❑ 파일 입출력 과정

- 파일 개방이 끝나면 파일포인터로 해당 파일에 입출력을 할 수 있다.
- 파일로부터 하나의 문자를 입력하는 예(fgetc함수)

```
int fgetc(FILE *);     // 파일에서 하나의 문자를 가져온다.
```

```
#include <stdio.h>

int main()
{
    FILE *fp;
    char ch;
    fp=fopen("a.txt", "r");
    if(fp==NULL){
        printf("파일 개방 실패.₩n");
        return 1;
    }
    ch=fgetc(fp);
    printf("입력한 문자 : %c₩n", ch);
    fclose(fp);
    return 0;
}
```

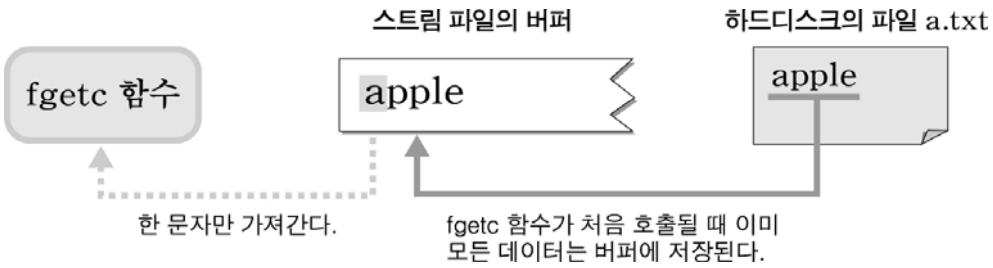


출력 결과

```
입력한 문자 : a
```

▶ 파일로부터 데이터가 입력되는 과정

- 입력함수가 처음 호출되면 파일로부터 버퍼 크기 만큼의 데이터를 한번에 버퍼로 읽어 들인다.

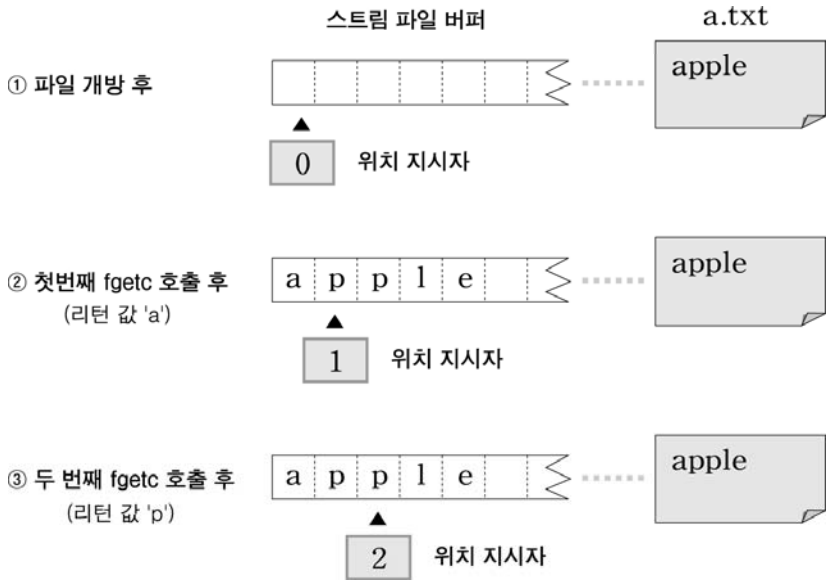


- 그 이후에 호출되는 입력함수는 버퍼에 데이터가 없을 때까지 버퍼로부터 데이터를 입력한다.

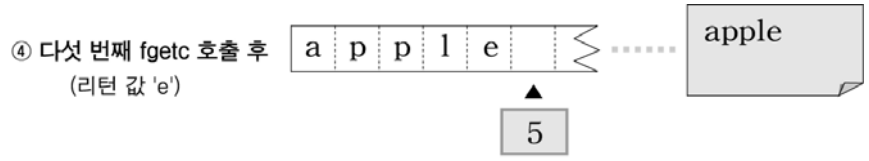
```
ch=fgetc(fp);
printf("입력 받은 문자 : %c₩n", ch);   // a 출력
ch=fgetc(fp);    // 두 번째 호출될 때는 버퍼로 부터 데이터를 입력한다.
printf("입력 받은 문자 : %c₩n", ch);   // p 출력
```

▶ 파일로부터 데이터가 입력되는 과정

- 버퍼로부터의 입력 위치는 FILE구조체의 멤버인 위치지시자로 확인한다.



- 위치지시자의 값이 파일의 크기와 같아지면 데이터를 모두 읽은 것이 되며 이 때 입력함수는 -1을 리턴한다(이 값은 보통 EOF로 기호화 한다).



⑤ 여섯 번째 fgetc 함수를 호출하면 -1을 리턴한다.

▶ 파일의 내용을 읽어서 화면에 출력하는 예

```
#include <stdio.h>
int main()
{
    FILE *fp;
    char ch;
    fp=fopen("a.txt", "r");
    if(fp==NULL){
        printf("파일 개방 실패.₩n");
        return 1;
    }
    while(1){                   // 무한 반복
        ch=fgetc(fp);           // 개방한 파일로부터 한 문자를 입력한다.
        if(ch==EOF) break;      // 리턴값이 -1(EOF)이면 파일의 끝이므로 반복 종료
        putchar(ch);            // 읽어 들인 문자를 화면에 출력한다.
    }
    fclose(fp);                 // 파일을 닫는다.
    return 0;
}
```

▶ 하나의 문자를 파일에 출력하자(fputc)

- 문자 하나를 파일에 출력할 때는 fputc함수를 사용한다.

```
int fputc(int, FILE *);     // 하나의 문자를 파일로 출력한다.
```

  - 첫 번째 전달인자로 주어지는 문자를 두 번째 전달인자의 파일로 출력한다.

- 키보드로부터 입력되는 데이터를 파일로 출력하는 예

```
#include <stdio.h>

int main()
{
    FILE *fp;
    char ch;
    fp=fopen("b.txt", "w");
    if(fp==NULL){
        printf("파일 개방 실패.₩n");
        return 1;
    }
    printf("데이터를 입력하세요.₩n");
    while(1){
        ch=getchar();          // 키보드 입력
        if(ch==EOF) break;
        fputc(ch, fp);         // 파일로 출력
    }
    fclose(fp);
    return 0;
}
```

```
데이터를 입력하세요.
banana (엔터)
apple (엔터)
^Z  (입력 종료)
```

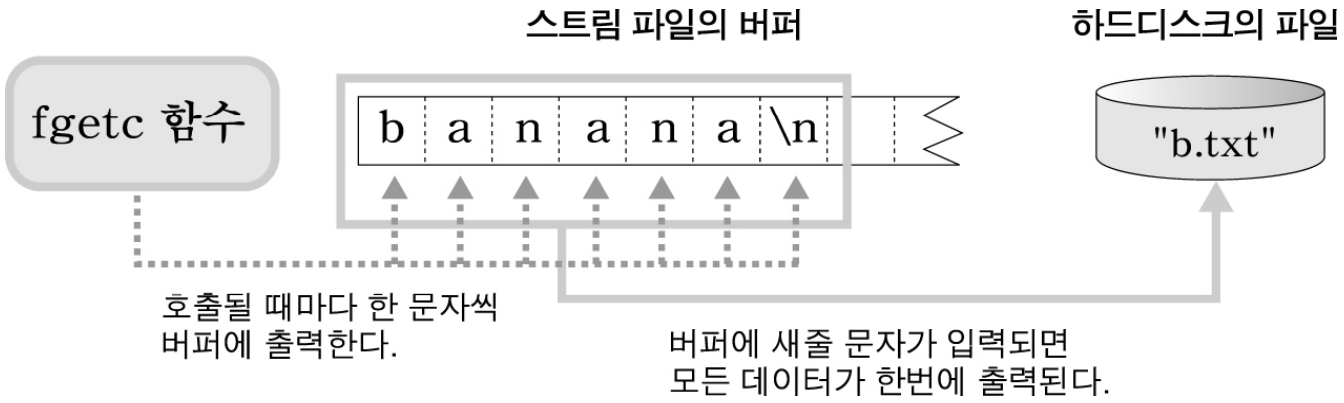
출력결과는 b.txt 파일을 메모장으로 열어 확인해 보도록 합시다.

하드디스크 파일 b.txt

```
banana
apple
```

▶ 출력 과정에서도 버퍼를 사용한다.

- fputc함수가 문자를 출력할 때도 fgetc함수와 마찬가지로 개방된 스트림 파일의 버퍼를 사용한다.
  - 한 문자를 출력할 때마다 일단 버퍼에 출력이 된 후에 새줄문자가 출력되면 하드디스크의 파일로 출력된다.

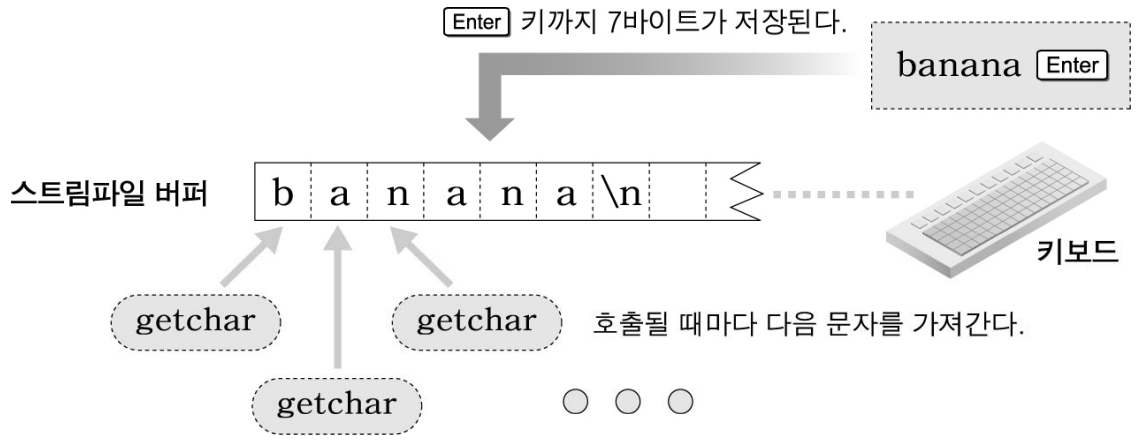


▶ 기본적으로 개방되는 표준 입출력 스트림 파일

- 프로그램이 실행될 때 기본적으로 개방되는 스트림파일 있다.

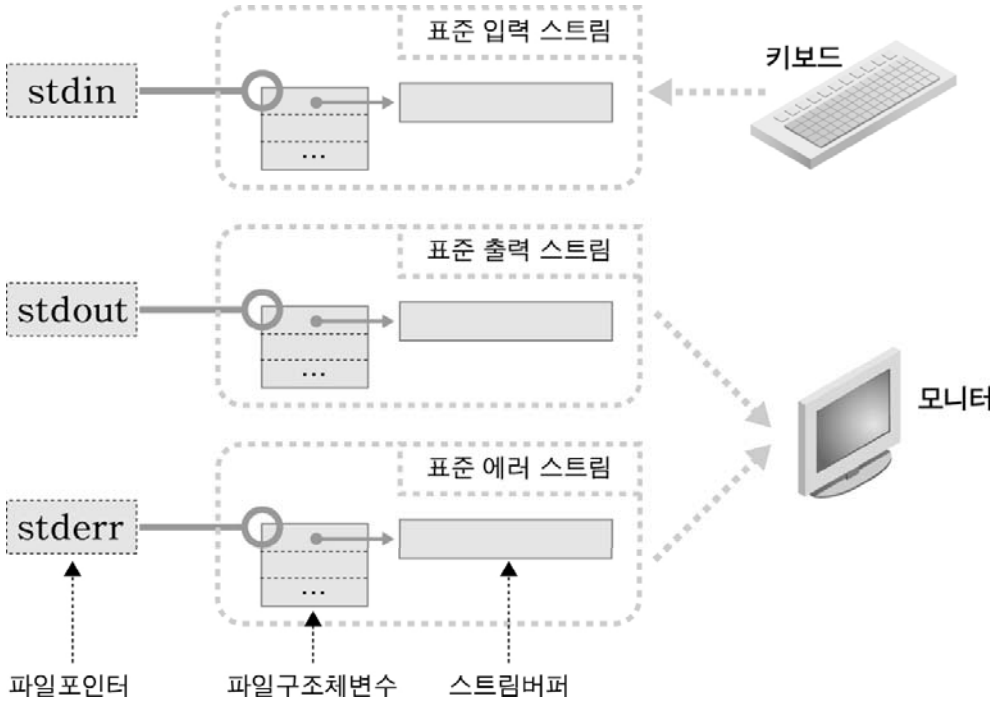
스트림 파일의 이름	스트림 파일의 용도	연결된 입출력 장치
stdin	표준 입력 스트림	키보드
stdout	표준 출력 스트림	모니터
stderr	표준 에러 스트림	모니터

– 지금까지 파일포인터의 사용 없이 입출력한 함수들은 기본개방 스트림파일을 이용하는 것이다(getchar함수가 키보드로부터 문자를 입력하는 예).



▶ 기본적으로 개방되는 표준 입출력 스트림 파일

- stdin, stdout, stderr은 파일포인터를 기호화한 것이다.



  - 이들 파일포인터를 사용하면 fgetc, fputc함수도 키보드나, 모니터로 입출력이 가능하다.

▶ 표준 입출력 스트림을 사용하는 프로그램 예

```
#include <stdio.h>

int main()
{
    char ch;

    while(1){
        ch=fgetc(stdin);        // stdin과 연결된 키보드로부터 데이터를 입력 받는다.
        if(ch==EOF) break;
        fputc(ch, stdout);      // stdout과 연결된 모니터로 데이터를 출력한다.
    }
    return 0;
}
```

출력 결과

```
stream (엔터)      // 키보드 입력
stream             // 화면 출력
^Z                 // 입력 종료
```

❑ 다양한 입출력 함수

- 파일 입출력 함수에는 필요에 따라 적절히 사용할 수 있는 다양한 함수가 있다.
- 문자열을 한번에 입출력 하는 함수 : fgets, fputs
- 다양한 자료형 맞게 입출력 하는 함수 : fscanf, fprintf
- 스트림 파일의 버퍼를 비워 주는 함수 : fflush

▶ 문자열을 한번에 입력하자(fgets)

- 문자열을 한번에 입력 할 때는 fgets함수를 사용한다.

```
char *fgets(char *, int, FILE *);  // 파일에서 문자열을 읽어 들인다.
```

- 파일포인터와 연결된 파일로부터 두 번째 전달인자로 주어진 바이트 수에 따라 데이터를 읽어와서 첫 번째 전달인자로 주어진 배열에 저장한다.

- 5바이트의 크기를 갖는 배열에 문자열을 입력 받는 경우

```
FILE *fp;
char str[5];
fp=fopen("a.txt", "r");
```

```
fgets( str, sizeof(str), fp );
```

입력 받을 배열의 배열명 / 입력 받을 바이트 수 / 파일포인터

널문자가 저장될 공간을 제외하고 4바이트만 입력된다. ⇨ str 배열 [m | o | n | k | \0]

▶ 문자열을 한번에 입력하자(fgets)

- fgets함수는 문자열 입력에 앞서 줄 단위로 입력 받는다.
  - 배열의 크기가 10바이트인 경우의 입력(새줄 문자도 입력 받는다).

```
FILE *fp;
char str[10];
fp=fopen("a.txt", "r");
```

fgets(str, sizeof(str), fp); ⇨ str 배열

m	o	n	k	e	y	\n	\0	?	?

  - 입력 받은 문자열에서 새줄 문자가 불필요할 때에는 제거한다.

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

str 배열에서 새줄 문자가 저장된 위치의 첨자가 구해진다.

strlen(str) ➡ 널문자 전까지의 바이트 수 ➡ 7

strlen(str) -1 ➡ 6 ➡ 새줄 문자가 저장된 위치의 첨자

str[strlen(str) -1] = '\0'; ➡ 새줄 문자가 널문자로 바뀐다.

▶ 문자열을 한번에 입력하자(fgets)

- 입력 받을 데이터의 수보다 파일의 크기가 작으면 파일 끝까지 읽어 들인다(물론 중간에 새줄 문자는 없어야 한다).

```
char str[80];
…
fgets(str, sizeof(str), fp);
```

입력파일의 데이터

```
monkey and tiger
```

⇨ 입력파일의 데이터가 80바이트가 안되므로 파일 끝까지 입력된다!

- fgets함수의 리턴값을 입력한 배열의 포인터이다. 따라서 입력이 끝난 후에 바로 이 포인터를 사용하여 문자열을 출력할 수 있다.

```
printf("%s₩n", fgets(str, sizeof(str), fp));
```

- 입력파일에서 더 이상 읽어 들일 데이터가 없으면 널 포인터를 리턴한다. (-1(EOF)가 아니므로 주의한다!)

```
…
res=fgets(str, sizeof(str), fp);
if(res==NULL) break;      // 파일의 끝이면 입력을 종료한다.
…
```

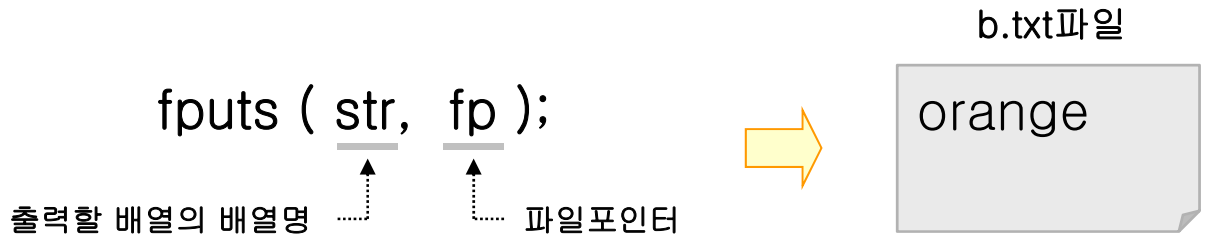
▶ 문자열을 한번에 출력하자(fputs)

- 문자열을 한번에 출력 할 때는 fputs함수를 사용한다.

```
int fputs(char *, FILE *);     // 파일로 문자열을 출력한다.
```

– 첫 번째 전달인자는 출력할 문자열의 위치를 주고 두 번째 전달인자는 파일 포인터를 준다(puts함수와는 달리 자동으로 줄을 바꾸지 않는다).

```
FILE *fp;                      // 파일포인터변수
char str[] = “orange”;         // 출력할 데이터가 저장된 배열, 초기화한다.
fp=fopen(“b.txt”, “w”);        // 파일을 출력용으로 개방
```



▶ 여러 줄의 문장을 한 줄로 출력하는 프로그램

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

int main()
{
    FILE *ifp, *ofp;
    char str[80];
    char *res;

    ifp=fopen("a.txt", "r");
    if(ifp==NULL){
        printf("입력파일 개방 실패.₩n");
        return 1;
    }

    ofp=fopen("b.txt", "w");
    if(ofp==NULL){
        printf("출력파일 개방 실패.₩n");
        return 1;
    }

    while(1){
        res=fgets(str, sizeof(str), ifp);
        if(res==NULL) break;
        str[strlen(str)-1]='₩0';
        fputs(str, ofp);
        fputs(" ", ofp);
    }

    fclose(ifp);
    fclose(ofp);

    return 0;
}
```

입력 파일 a.txt

```
소년은 (엔터)
늙기 쉽고 (엔터)
학문은 (엔터)
이루기 어렵다. (엔터)
```

출력 파일 b.txt

```
소년은 늙기 쉽고 학문은 이루기 어렵다.
```

▶ gets, puts대신 fgets, fputs함수를 사용하자.

- gets 함수는 데이터를 입력할 때 할당되지 않은 기억공간을 침범할 가능성이 있다.

```
char str[10];
gets(str);      // 사용자가 10바이트를 넘는 데이터를 입력하면 문제가 발생한다!
```

- puts 함수는 항상 줄을 바꿔 주므로 문자열을 출력한 후에 바로 입력하는 경우에 사용이 불가능하다.

```
int age;
puts("나이를 입력하세요 : ");
scanf("%d", &age);
```

```
나이를 입력하세요 :
_       // 항상 다음 줄에서 입력한다.
```

- stdin, stdout을 파일포인터로 사용하여 fgets, fputs함수를 사용한다.

```
fgets(str, sizeof(str), stdin);     // 키보드로부터 문자열을 입력한다.
fputs(str, stdout);                 // 모니터로 문자열을 출력한다.
```

▶ 다양한 자료형에 맞게 입출력하자(fscanf, fprintf)

- fscanf, fprintf함수는 scanf, printf함수와 사용법이 같다. 단, 입출력 대상을 파일포인터로 지정해 줄 수 있다.

```
int fscanf(FILE *, char *, …);   // 파일에서 형식에 따라 데이터 입력
int fprintf(FILE *, char *, …);  // 파일로 형식에 따라 데이터 출력
```

  - fscanf함수는 데이터의 입력이 끝나면 -1을 리턴한다.

- 이름, 나이, 키가 저장된 텍스트 파일의 데이터를 형식에 따라 입력한 후에 키, 나이, 이름의 순서로 출력하는 예

텍스트 파일 a.txt

```
박준성  25  188.9
지혜연  23  162.5
조충근  19  175.0
```

텍스트 파일 b.txt

```
188.9  25  박준성
162.5  23  지혜연
175.0  19  조충근
```

▶ fscanf, fprintf함수를 사용한 프로그램 예

```
#include <stdio.h>

int main()
{
    FILE *ifp, *ofp;
    char name[20];
    int age;
    double height;
    int res;

    ifp=fopen("a.txt", "r");
    if(ifp==NULL){
        printf("입력파일 개방 실패.₩n");
        return 1;
    }
    ofp=fopen("b.txt", "w");
    if(ofp==NULL){
        printf("출력파일 개방 실패.₩n");
        return 1;
    }
    while(1){
        res=fscanf(ifp, "%s%d%lf", name, &age, &height);
        if(res==EOF) break;
        fprintf(ofp, "%.1lf %d %s₩n", height, age, name);
    }

    fclose(ifp);
    fclose(ofp);
    return 0;
}
```

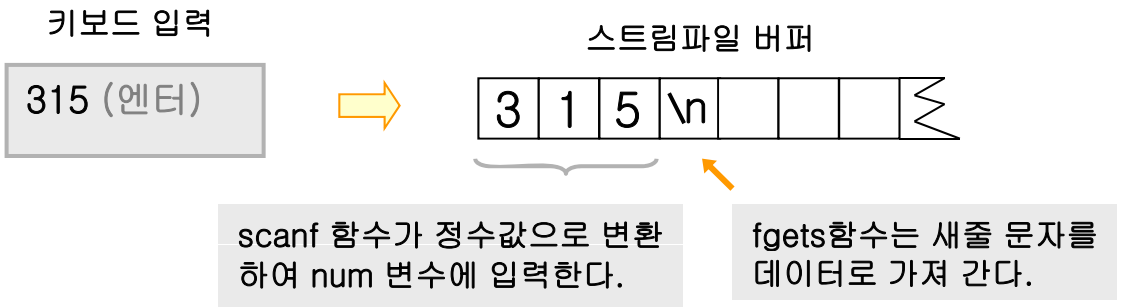
▶ 스트림 파일의 버퍼를 비운다(fflush)

- 입출력 함수들이 버퍼를 공유하기 때문에 예상치 못한 문제가 발생한다.
  - 학번을 입력하고 바로 이름을 입력하는 예

```
int num;
char name[20];
printf("학번을 입력하세요 : ");
scanf("%d", &num);
printf("이름을 입력하세요 : ");
fgets(name, sizeof(name), stdin);
```

```
학번을 입력하세요 : 315 (엔터)
이름을 입력하세요 : 학번 : 315
이름 :
```

- scanf함수가 학번을 입력 받은 후에 버퍼에 남겨진 새줄 문자를 다음에 호출되는 fgets함수가 데이터로 받아들이기 때문이다.



▶ 스트림 파일의 버퍼를 비운다(fflush)

- fflush함수는 버퍼에 남아 있는 불필요한 데이터를 삭제한다.

```
int fflush(FILE *);      // 스트림파일의 버퍼를 비워준다.
```

```
int num;
char name[20];
printf("학번을 입력하세요 : ");
scanf("%d", &num);
fflush(stdin);    // scanf함수와 gets함수가 공유하는 표준 입력 스트림버퍼를 비운다.
printf("이름을 입력하세요 : ");
fgets(name, sizeof(name), stdin);
```

```
학번을 입력하세요 : 315 (엔터)
이름을 입력하세요 : 홍 길동 (엔터)
학번 : 315
이름 : 홍 길동
```