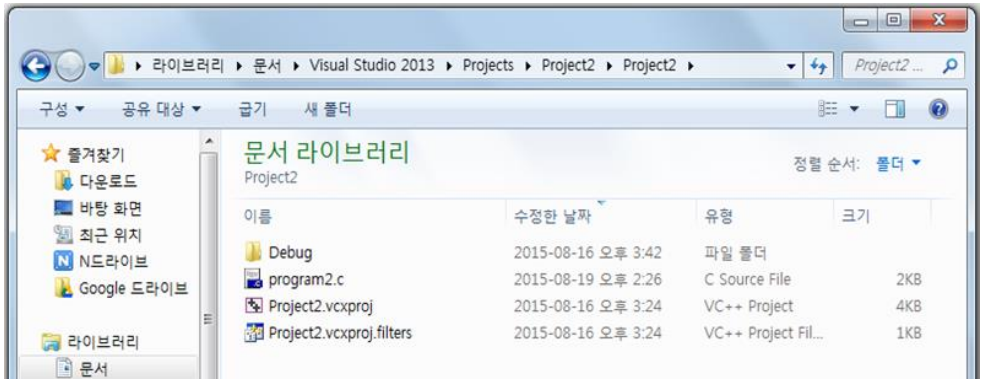
14차시 - 파일(2)

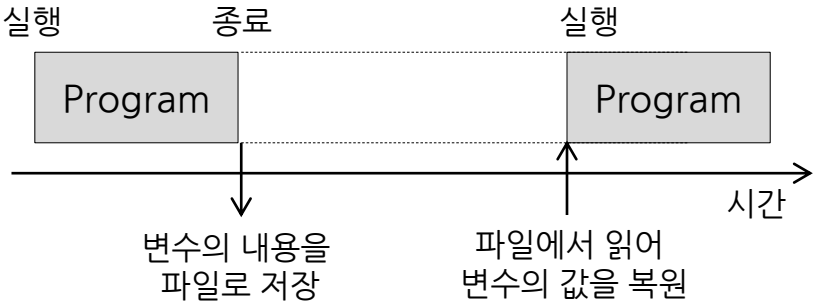
메모리

- **메모리**
  - **주메모리** : 속도가 빠르다. 가격이 비싸다. 휘발성. 프로그램 실행에 필수
  - **보조메모리** : 속도가 느리다. 가격이 싸다. **영구적.**
  - 영구적인 자료 보관, 대용량의 데이터는 보조메모리 이용
- **파일**
  - 이름 + 확장자, 날짜, 크기
  - 폴더

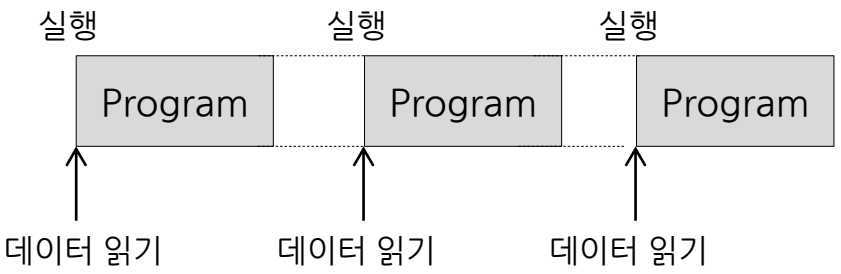


프로그램이 파일을 지원하면

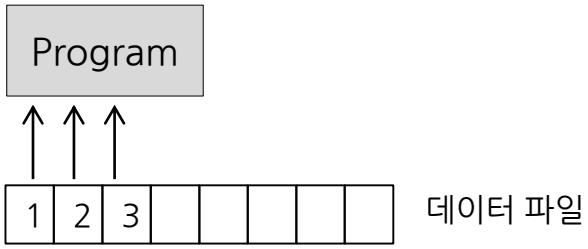
① 프로그램 실행의 연속성



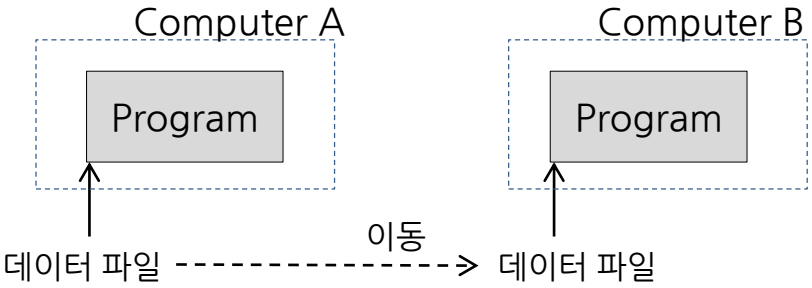
② 번거로운 데이터 입력 자동화



③ 데이터의 분할 처리



④ 데이터의 이동

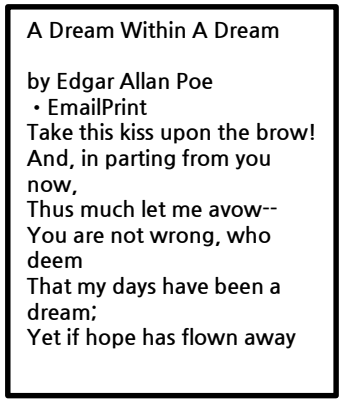


프로그램이 한층 강력해진다.

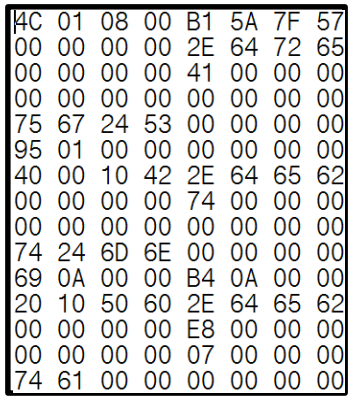
C 언어의 파일

파일의 종류

- **텍스트 파일** : 문자로 구성된 파일. 문자열을 저장한 것
  - 확장자가 .txt, .log, .ini, .xml 로 구성된다.
  - 문서 편집기, 메모장을 통하여 내용을 쉽게 확인할 수 있다.
- **이진(binary) 파일** : 각종 숫자를 포함한 이진 데이터를 저장한 것(포괄적)
  - mp3, jpg, avi, .exe를 비롯한 대부분의 파일



텍스트 파일



이진 파일

C 언어에서 파일 생성과 기록

- **파일 열기**
  - 파일 이름과 목적(읽기 또는 쓰기) 지정
  - 파일의 스트림 지시자(핸들)를 얻는다.
- **파일 읽기/쓰기**
  - 용도에 맞도록 파일의 내용을 읽어 메모리로 가져오거나, 메모리의 내용을 파일에 기록한다.
- **파일 닫기**
  - 파일 작업이 완료되면 파일을 마무리한다.
  - 파일을 닫으면 새로 열기 전까지 파일 접근을 할 수 없다.

파일의 이름

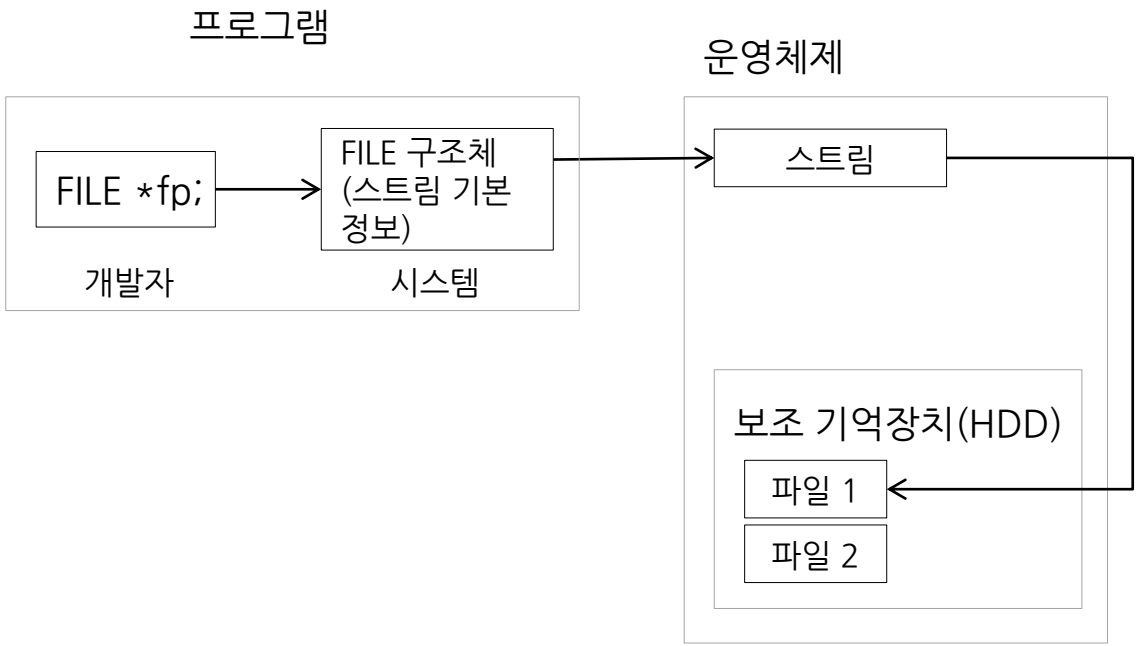
- **파일 이름**
  - 저장될 데이터의 특성을 잘 나타낼 수 있는 이름을 선택
  - 문자열이므로, “ ” 사이에 기록한다.
  - 알파벳, 숫자, 일부 기호 사용 가능. 대소문자 구별 없음.
  - ₩, /, *, *, ?, “, <, >, | 는 사용 불가 (운영체제마다 약간 다를 수 있다)
- **폴더**
  - 폴더와 폴더, 폴더와 파일 이름 사이에 ‘₩’ 또는 ‘/’을 붙여야 한다(운영체제 지시 사항. 리눅스의 경우 ‘/’ 만 가능)
  - “ ” 안에 ₩를 쓸 때에는 “₩₩”처럼 두 개를 써야 한다(이스케이프시퀀스).

읽기 쓰기 모드

- **파일을 열 때 모드를 지정해야 한다.**
- **보통 읽기 또는 쓰기 중 하나를 지정한다.**
  - 두 가지를 모두 하더라도, 읽기를 모두 한 후, 쓰기를 한다.
  - 두 가지 일을 섞어서 하면 관리가 어렵고 성능도 저하된다.
- **읽기를 하려면 파일이 존재해야 한다.**
- **쓰기를 하려면 파일 존재가 무관하다.**

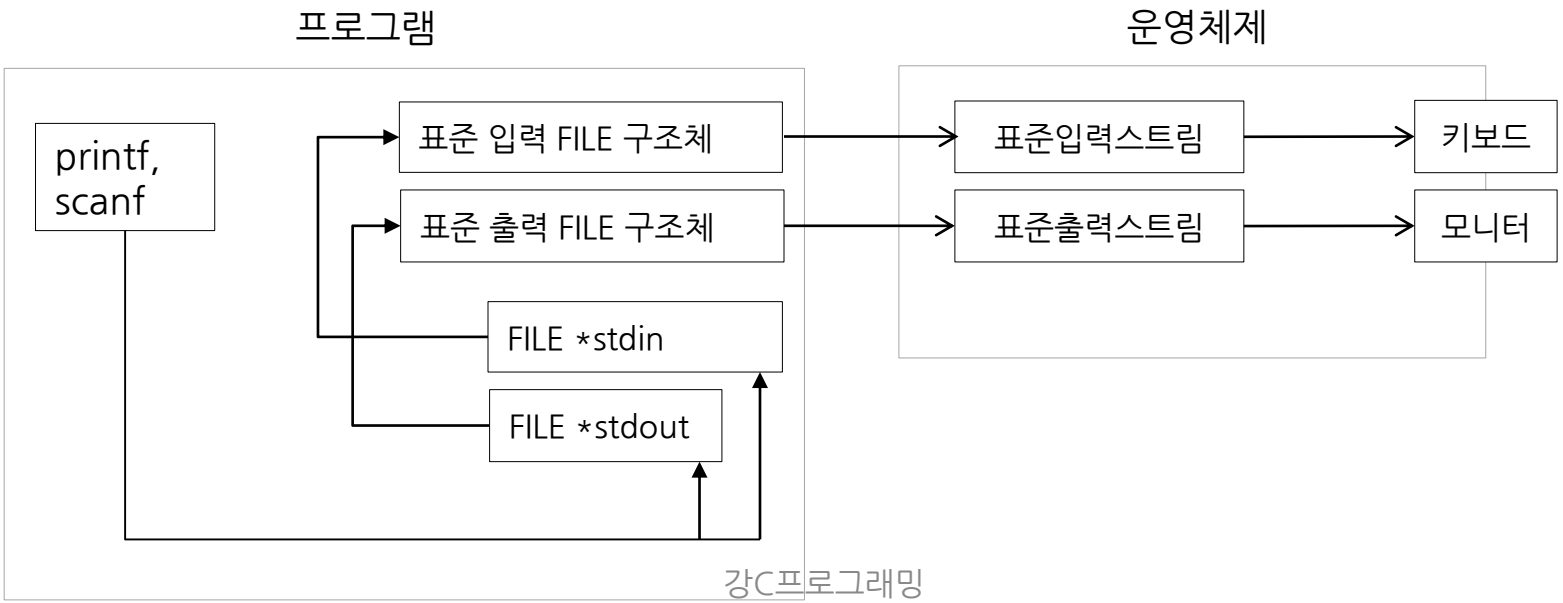
파일과 운영체제

- 파일 관리는 운영체제의 역할이다(메모리처럼).
- FILE 구조체는 파일 접근을 위한 정보 구조체이다.



표준 입출력

- `stdin, stdout, stderr` 도 FILE * 이다.
- 사전에 정의된 입출력 스트림 구조체의 포인터이다.
- 준비되어 있으므로 입출력에 바로 사용 가능하다.



파일 열기

- FILE *fopen(const char *filename, const char *mode)
  - filename : 열고자 하는 파일 이름
  - mode : 파일 열기 모드
  - 반환값 : 지정한 파일의 FILE 구조체 포인터 (스트림 지시자)
- 예
  - FILE *pFile = fopen("hporter.txt", "r");
  - FILE *pFile = fopen("score.dat", "wb");
  - FILE *pFile = fopen("data\\report.txt", "w");
  - FILE *pFile = fopen("diary.txt", "r+");

파일 열기

모드

모드		의미
기본	“r”	읽기 (파일이 반드시 존재)
	“w”	쓰기 (파일을 새로 생성. **기존 파일 있으면 내용 삭제**)
	“a”	기존 파일에 추가하기(마지막 이후에 쓰기)
추가	추가 “b”, “t”	이진 파일 또는 텍스트 파일(“t”는 생략 가능)
	추가 “+”	업데이트 가능. 기본 동작에 추가로 업데이트를 한다.
	추가 “x”	“w”와 결합하여, 기존 파일이 있으면 열기 실패(C11)

- 예) “r”, “rt”, “wb”

반환값

- FILE 구조체 포인터

파일 열기

여러 파일을 사용할 때

- 하나의 FILE 구조체 포인터를 이용

```
FILE  *pFile;
pFile = fopen("1번파일", "r");
…
fclose(pFile);     // 1번 파일 사용 종료
pFile = fopen("2번파일","wb");
…
```

- 동시에 여러 개의 파일을 이용한다면 여러 개의 FILE 구조체 포인터를 이용

```
FILE *pFile1, *pFile2, *pFile3;
pFile1 = fopen("1번파일", "r" );
pFile2 = fopen( "2번파일", "wb");
pFile3 = fopen( "3번파일", "w");
```

파일 입출력(텍스트 모드)

- `int fprintf(FILE * stream, const char * format, ... );`
- `int fscanf (FILE * stream, const char * format, ... );`

- `char *fgets(char * str, int num, FILE * stream );`
- `int  fputs(const char * str, FILE * stream );`

- `int  fgetc (FILE * stream );`
- `int  fputc (int character, FILE * stream );`

문자 입출력

- `int fgetc(FILE *stream)`
- `int fputc(int character, FILE *stream)`
  - 하나의 문자 읽기/쓰기
  - 읽기/쓰기에 성공한 크기 반환 (보통 1)
  - 실패할 수 있다.

문자 입출력

```c
#include <stdio.h>

int main ( )
{
  FILE * pFile;
  char c;

  pFile = fopen ("alphabet.txt“ ,"w");
  if (pFile != NULL) {
     for (c = 'A' ; c <= 'Z' ; c++)
         fputc ( c , pFile );
     fclose (pFile);
  }
  return 0;
}
```

```c
#include <stdio.h>

int main ( )
{
  FILE * pFile;
  int c;

  pFile = fopen ("alphabet.txt","r");

  if (pFile == NULL) perror("파일 열기 실패");
  else {
    do {
       c = fgetc(pFile); //파일에서 한 문자 가져와
       if (c != EOF)        // 제대로 가져왔으면
           putc(c, stdin );// 화면에 출력
    } while (c != EOF);    // 파일의 끝까지 반복
    fclose (pFile);
  }  // end of if
  return 0;
} // end of main
```

문자열 입출력

- `char * fgets ( char * str, int num, FILE * stream );`
- `int fputs ( const char * str, FILE * stream );`
  - 행 단위 문자 입출력
  - 최대 num-1 바이트를 읽는다. 마지막 1바이트는 ₩0

```
pFile = fopen ("sentence.txt" , "r");
if (pFile == NULL) perror ("파일 열기 실패");
else {
    if(fgets(mystr, 100, pFile) != NULL)  // 파일 가져오기에 성공하면
        puts (mystr);                     // 화면에 출력
    fclose (pFile);
}
```

문자열 입출력 (형식 지정)

- `int fprintf (FILE *stream, const char * format, ...);`
- `int fscanf (FILE *stream, const char * format, ...);`
  - printf, scanf와 같은 형식으로 사용한다.
  - `fscanf(pFile, "%d %d", &i, &j);`

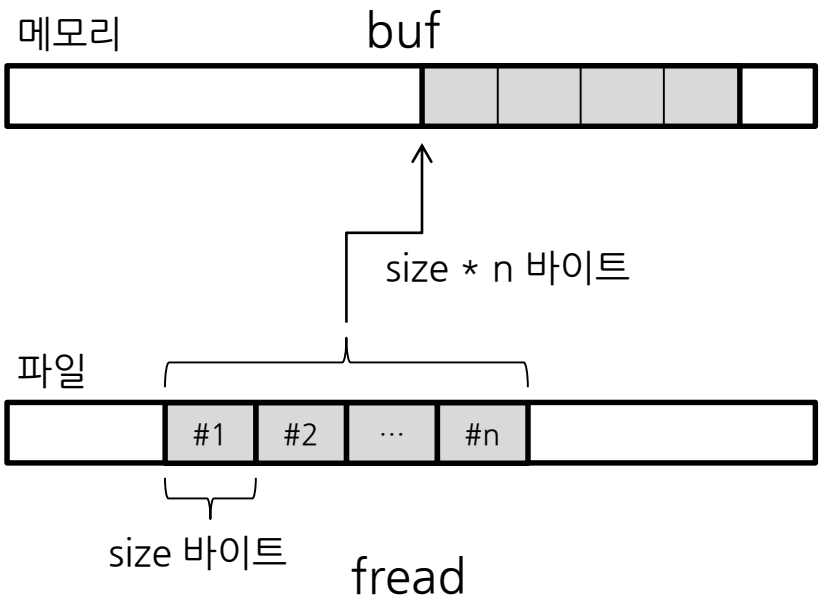
- sscanf, sprintf**를 이용하여 메모리에 작업한 후, 파일에 기록하는 방법도 있다.**
  - 결과물에 별도 가공이 가능하다.
  - `fgets(str, sizeof(str), pFile);`
  - `sscanf(str, "%d %d", &i, &j);`

파일 입출력(바이너리 모드)

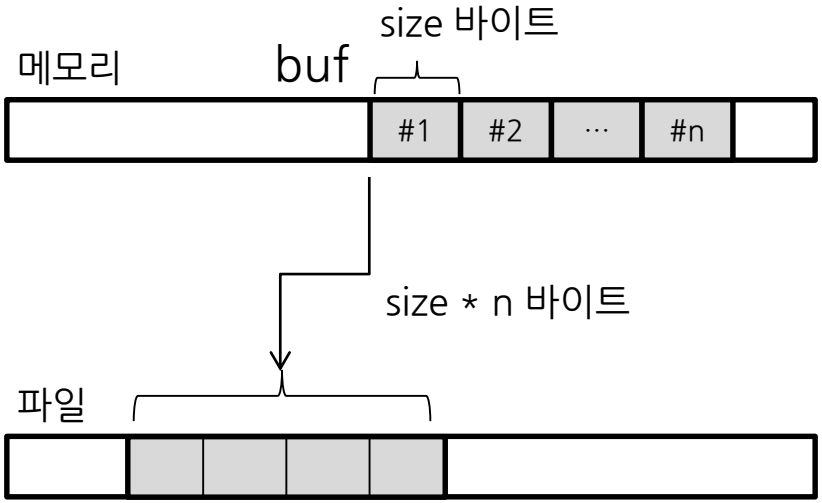
- 파일 열기, 닫기는 텍스트 파일과 같다(모드만 다르다).
- 이진 파일은 텍스트 파일과 다르다.

- `int fread(void *buf,  int size, int n, FILE *fp);`
- `int fwrite(const void *buf, int size, int n, FILE *fp);`
  - buf : 읽기/쓰기를 데이터 공간
  - size : 한 블록의 크기 n : 블록의 개수
  - 반환값 : 읽기/쓰기에 성공한 블록 수

이진 파일 읽기/쓰기



fread



fwrite

이진 파일 읽기/쓰기

- `int fread(void *buf, int size, int n, FILE *fp);`
  - size 크기의 블록을 n개만큼 이진 파일을 읽어 buf에 저장하라. (size * n 바이트)
  - 다음의 세 가지는 동일하다.
    - `fread(buf, 10, 100, fp);`
    - `fread(buf, 100, 10, fp);`
    - `fread(buf, 1000, 1, fp);`
  - 반환값은 작업한 크기 (블록 수)

- `int fwrite(const void *buf, int size, int n, FILE *fp);`
  - 반환값은 작업한 크기 (블록 개수) - size와 다르면 쓰기 에러

파일 닫기

- `int fclose(FILE *fp);`
  - 정상실행이면 0, 문제가 있으면 EOF 반환

- **닫기를 꼭 해야 하나**

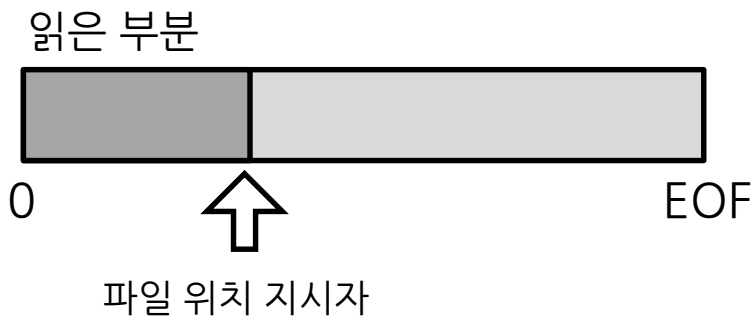
미니 과제

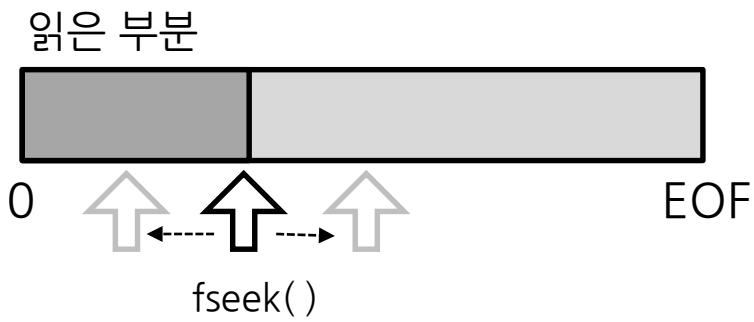
- 프로그램을 처음 실행하면 관리자의 생일(년 월 일)을 묻는다. 그 다음 프로그램을 실행할 때부터는 관리자의 생일을 물어 정확하게 알면 GOOD이라고 출력하고 틀리면 다시 묻는 프로그램을 작성하라.

```
C:\> birth.exe
관리자의 생일을 입력하시면 기억하겠습니다. 1997 5 12
관리자의 생일이 저장되었습니다. 1997년 5월 12일.
C:\> birth.exe
관리자를 확인합니다. 생일을 입력하세요. 1998 3 3
관리자가 아닙니다. 실행할 수 없습니다.
C:\> birth.exe
관리자를 확인합니다. 생일을 입력하세요. 1997 5 12
관리자님 안녕하세요.
```

파일 위치 조정

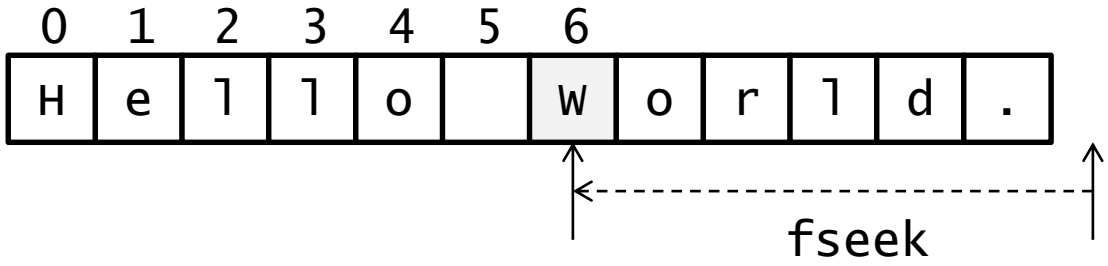
- **파일은 순차 접근 메모리**
  - 파일은 앞에서 뒤로 차례로 기록되어 있다.
  - 읽기 위치를 조정할 수 있고, 앞 부분을 건너뛰고 뒷 부분을 읽을 수 있다.
- `int fseek (FILE *fp, long offset, int mode);`
  - offset과 mode로 지정한 위치로 읽는 위치를 이동한다.
  - mode는 SEEK_SET(파일의 시작부터), SEEK_CUR(현재 위치), SEEK_END(파일의 끝)중 하나.





파일 위치 조정

```
FILE * pFile;
pFile = fopen (“text.txt", "wb"); // 텍스트 파일이지만, 이진 모드로 기록했다.
fputs (“Hello World.", pFile);    // 일단 문자열을 기록한다.
fseek (pFile , 6 , SEEK_SET);     // 기록 위치를 조정한다.
fputs (“Cprog", pFile );          // 문자열을 추가 기록한다.
fclose ( pFile );
```



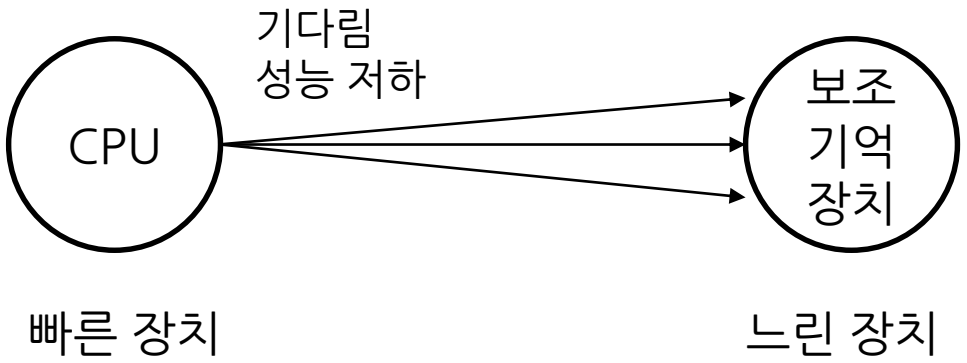
text.txt 파일에 Hello Cprog.가 들어있다.

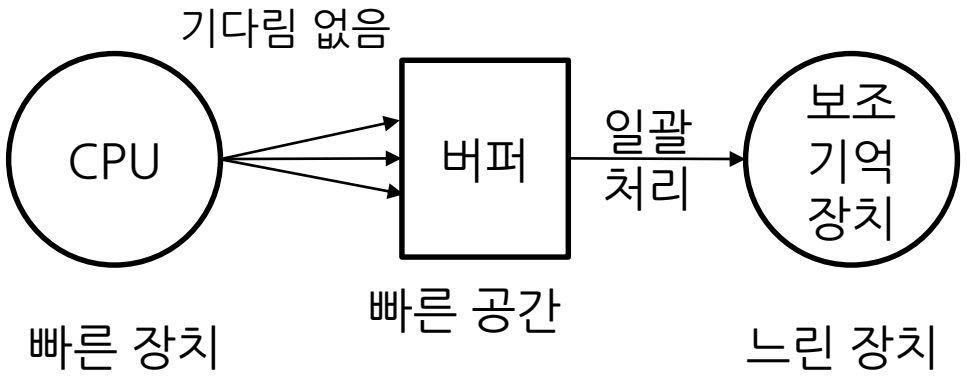
좋은 방법은 아니다. 불가피할 때에만 쓴다.

파일 위치 조정

- `void rewind( FILE *fp )`
  - 읽는 위치를 처음으로 되돌린다.
  - 읽기 쓰기 겸용 모드(+)에서 모드 전환시 이용한다(읽기 후 쓰기 전환)
- `int fflush( FILE *fp )`
  - 성능 향상을 위해 버퍼링을 하는데, 버퍼의 내용을 디스크에 강제 반영한다.
  - 비정상 종료에 대비한다.
- `long ftell( FILE *fp )`
  - 읽는 위치를 반환한다.

버퍼링


기다림
성능 저하
CPU
보조
기억
장치
빠른 장치
느린 장치


기다림 없음
CPU
버퍼
일괄
처리
보조
기억
장치
빠른 공간
빠른 장치
느린 장치

- 주의할 점은?

파일의 끝, EOF

- `int feof ( FILE * stream );`
  - 파일의 끝이면 True를, 파일의 끝이 아니면 False를 반환

```
FILE * pFile;
int  size = 0;
pFile = fopen ("document.txt","r");
if (pFile==NULL) perror ("Error opening file");
else {
  while (fgetc(pFile) != EOF) {
    ++size;
  }
  if (feof(pFile)) {
    printf("파일 크기는 %d 바이트\n",
            size);
  }
  fclose (pFile);
}
```

또는

```
while (!feof(pFile) ) {
   fgetc(pFile); ++size;
}

printf("파일 크기는 %d 바이트\n",
         size);
}
```

파일 운용

- `int rename (const char *oldname, const char *newname);`
  `// 파일 이름 변경`
- `int remove (const char *filename);  // 파일 삭제`

- **주의할 점**

파일 삭제로 남을 괴롭히지 말자.

그 외

- 휴지통이란

- 그 외의 파일 관련 기능
  - 폴더 생성, 삭제
  - 파일 목록

실습 문제

- 파일의 이름을 입력하면 파일의 크기를 표시하는 프로그램을 작성하라.

- 키보드를 통하여 입력한 문장을 파일로 저장하는 프로그램을 작성하라. Ctrl-Z 를 누르면 입력이 종료되고 파일에 저장한다.