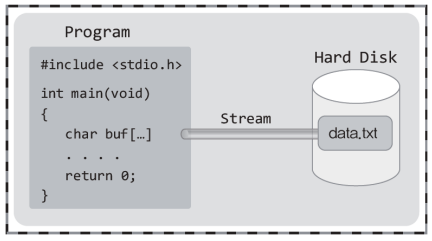
파일 입출력

**1. 파일과 스트림, 그리고 기본적인 파일의 입출력**

- 파일 스트림

프로그램과 파일 사이에 스트림을 형성해야 데이터를 주고받을 수 있다.



스트림이라는 것은 운영체제에 의해서 형성되는 소프트웨어적인 상태를 의미하는 것일 뿐, 실제로 위 그림과 같은 형태의 다리가 놓여지는 것은 아니다. 따라서 프로그램과 파일 사이에 스트림이 형성되었다고하면 파일로부터 데이터를 읽어 들일(파일에 데이터를 쓸) 기본적인 준비가 완료되었다고 이해하면 된다. 파일은 운영체제에 의해서 그 구조가 결정되고 관리되는 대상이기 때문에, 파일뿐만 아니라 스트림의 형성도 운영체제의 몫이다.

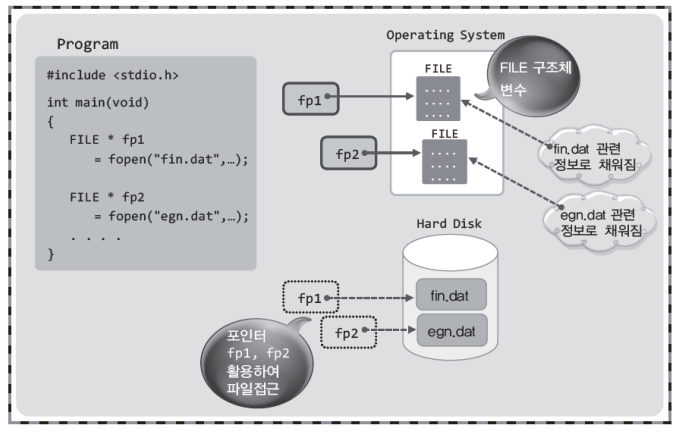
- fopen 함수호출을 통한 파일과의 스트림 형성과 file 구조체

다음은 스트림을 형성할 때 호출하는 함수이다. 이 함수의 호출을 통해서 프로그램상에서 파일과의 스트림을 형성할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위 함수의 첫 번째 인자로는 스트림을 형성할 파일의 이름을, 두 번째 인자로는 형성할 스트림의 종류에 대한 정보를 문자열의 형태로 전달한다. 그러면 이 함수는 해당 파일과의 스트림을 형성하고 스트림 정보를 FILE 구조체 변수에 담아서 그 변수의 주소 값을 반환한다.



\* fopen 함수가 호출되면 FILE 구조체 변수가 생성된다.

\* 생성된 FILE 구조체 변수에는 파일에 대한 정보가 담긴다.

\* FILE 구조체의 포인터는 사실상 파일을 가리키는 ‘지시자’의 역할을 한다.

- 입력 스트림과 출력 스트림의 생성

스트림은 한 방향으로 흐르는 데이터의 흐름을 의미한다. 때문에 스트림은 데이터를 파일로부터 읽어 들이기 위한 입력 스트림과 데이터를 파일에 쓰기 위한 출력 스트림으로 구분된다. fopen 함수를 호출할 때에는 다음 두 가지가 인자로 전달되어야 한다.

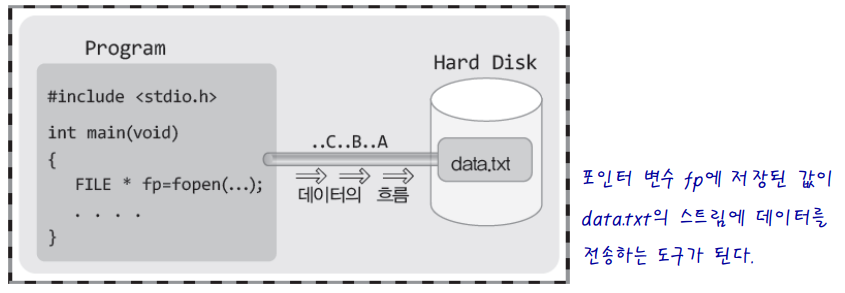
\* 첫 번째 전달인자 | 스트림을 형성할 파일의 이름

\* 두 번째 전달인자 | 형성하고자 하는 스트림의 종류

먼저 출력 스트림의 형성을 요청하는 fopen 함수의 호출문을 보이겠다.

FILE \* fp = fopen(“data.txt”, “wt”); // 출력 스트림의 형성

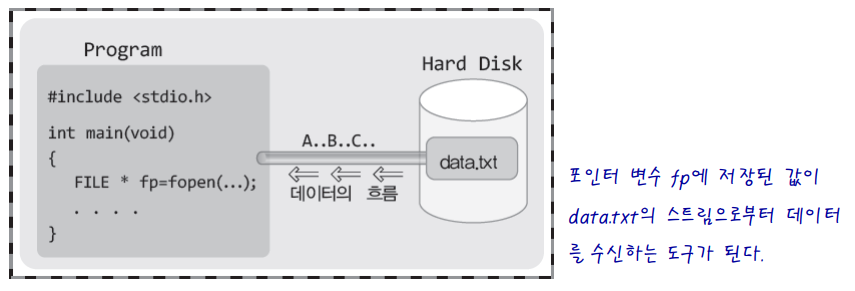
이것은 파일 data.txt와 스트림을 wt모드로 형성하라는 의미이다. wt모드의 스트림은 텍스트 데이터를 쓰기 위한 출력 스트림을 뜻한다.



위의 그림에서 보이는 스트림은 출력 스트림이기 때문에 파일에 데이터를 쓸 수는 있어도 읽지는 못한다. 만약에 파일로부터 데이터를 읽기 원한다면 다음의 문장을 통해서 별도로 입력 스트림을 형성해야 한다.

FILE \* fp = fopen(“data.txt”, “rt); // 입력 스트림의 형성

이것은 파일 data.txt와 스트림을 rt모드로 형성하라는 의미이다. rt모드의 스트림은 텍스트 데이터를 읽기 위한 입력 스트림을 뜻한다.



- 파일에 데이터 쓰기

텍스트, 실내, 화면이(가) 표시된 사진

자동 생성된 설명

위 예제 5행에서는 파일 data.txt와의 출력 스트림을 형성하고 있다. 이렇듯 출력 스트림을 형성하는 경우에는 해당 파일이 생성된다.

경로를 포함해서 **FILE \* fp = fopen(“C:\\project\\data.txt”, “wt”);**로 파일의 이름을 지정해도 된다. 위와 같이 파일의 경로를 지정하려면 C드라이브에 Project라는 디렉터리가 생성되어 있어야 한다. 그러면 해당 디렉터리에서 파일 data.txt가 생성됨을 확인할 수 있다. 이렇게 해서 스트림이 형성되면, 이제부터 fp는 파일 data.txt를 지칭하는 포인터가 된다. 따라서 **fputc(‘A’, fp);**가 실행되면, fp가 지칭하는 파일 data.txt에 문자 A가 저장된다.

텍스트이(가) 표시된 사진

자동 생성된 설명



- 스트림의 소멸을 요청하는 fclose 함수

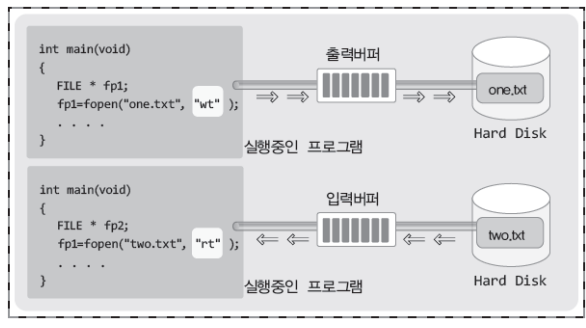
fclose 함수는 fopen 함수의 반대 기능을 제공한다. fopen 함수가 스트림을 형성하는 함수라면, fclose 함수는 스트림을 해제하는 함수이다. 다시 말해서 fopen 함수가 파일을 개방하는 함수라면, fclose 함수는 파일을 닫는 함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

함수의 호출을 통해서 스트림의 형성을 요청하는 것은 우리지만, 실제로 스트림을 형성하는 주체는 운영체제이다. 그리고 운영체제는 스트림의 형성을 위해서 시스템의 자원을 할당한다. 그런데 이 자원은 파일을 닫아주지 않으면 할당된 채로 남아있게 되어, 그만큼의 자원손실을 초래하기 때문에 파일의 사용이 끝나는 즉시 fclose 함수를 호출해서 자원을 반환해줄 필요가 있다.

파일 스트림 중간에 입력버퍼와 출력버퍼가 존재한다. 아래 그림에서 보이듯이 운영체제는 프로그램과 파일 사이에 입출력 버퍼를 둬서 성능의 향상을 도모하고 있다. 때문에 여러분이 fputc와 같은 함수의 호출로 데이터를 파일로 전송한다고 해서 파일에 바로 저장되는 것이 아니라, 일단은 출력버퍼에 저장되었다가 운영체제가 정해놓은 버퍼링 방식에 따라서 뒤늦게 파일에 저장이 된다. 그런데 이러한 버퍼링 방식이 지니는 문제점이 하나 있다. 예를 들어서 문자 A와 B가 출력버퍼에 존재하는 상태에서 컴퓨터의 전원이 꺼졌다고 가정해 보자. 문자 A와 B는 파일에 저장이 되겠는가? 저장되지 않는다. 그러나 fclose 함수의 호출을 통해서 파일을 닫아주면 출력 버퍼에 저장되어 있던 데이터가 파일로 이동하면서 출력버퍼는 비워지게 된다. 물론 그 이후에는 컴퓨터의 전원이 꺼져도 파일로 저장된 데이터는 소멸되지 않는다. 때문에 사용이 끝난 파일은 곧바로 fclose 함수를 호출해 주는 것이 좋다.



- fflush 함수

스트림을 종료하지 않고 버퍼만 비우고 싶을 때에는 다음 fflush 함수를 호출하면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

\* 출력버퍼를 비운다는 것은 출력버퍼에 저장된 데이터를 목적지로 전송한다는 의미

\* 입력버퍼를 비운다는 것은 입력버퍼에 저장된 데이터를 소멸시킨다는 의미

\* fflush 함수는 출력버퍼를 비우는 함수이다.

\* fflush 함수는 입력버퍼를 대상으로 호출할 수 없다.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 파일로부터 데이터를 읽기

실행을 통해서 만든 파일을 열어서 그 안에 저장된 문자를 읽어 들이는 프로그램을 작성해 볼 텐데, 이를 위해서 fgetc 함수를 **int ch=fgetc(fp);** 형태로 호출할것이다.

fgetc 함수는 파일에 저장된 문자 하나를 반환하는 함수이다.

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명 이전에 문자가 써진 순서대로 읽힌다.

예제의 실행환경 및 설정 그리고 실행방법을 달리하지 않는다면, fopen 함수가 파일을 생성하는 위치와 찾는 위치는 동일하며 또 달라지지 않는다. 따라서 앞서 보인 예제에서 생성한 파일을 옮기지 않는다면 6행의 fopen 함수호출을 통해서 문제없이 파일을 찾을 수 있다.

**2. 파일의 개방 모드**

- 스트림을 구분하는 기준1: 읽기 위한 스트림? 쓰기 위한 스트림?

|  |  |  |
| --- | --- | --- |
| 모드(mode) | 스트림의 성격 | 파일이 없으면? |
| r | 읽기 가능 | 에러 |
| w | 쓰기 가능 | 생성 |
| a | 파일의 끝에 덧붙여 쓰기 가능 | 생성 |
| r+ | 읽기/쓰기 가능 | 에러 |
| w+ | 읽기/쓰기 가능 | 생성 |
| a+ | 읽기/덧붙여 쓰기 가능 | 생성 |

\* 모드의 +는 읽기, 쓰기가 모두 가능한 스트림의 형성을 의미한다.

\* 모드의 a는 쓰기가 가능한 스트림을 형성하는데, 여기서 말하는 쓰기는 덧붙이기이다.

- 텍스트 파일과 바이너리 파일

이어서 스트림을 구분하는 두 번째 기준에 대한 설명을 시작하겠다. 그리고 이를 위해서 파일에 담을 수 있는 데이터들의 유형을 나열해 보았다.

\* 개인이 소유하는 도서의 목록 | 문자 데이터

\* 슈퍼마켓의 물품 가격 | 문자 데이터

\* 제임스 캐머런 감독의 타이타닉 영상파일 | 바이너리 데이터

\* 소녀시대의 히트곡 음원파일 | 바이너리 데이터

정리하면, 사람이 인식할 수 있는 문자를 담고 있는 파일을 가리켜 텍스트 파일이라 하며, 그 이외에 컴퓨터가 인식할 수 있는 데이터를 담고 있는 파일을 가리켜 바이너리 파일이라 한다. 그렇다면 데이터의 입출력을 위해서 스트림을 형성할 때 이와 관련해서 특별히 신경 쓸 부분은 무엇일까? 그것은 바로 문장의 끝을 의미하는 개행의 표현방식이다.

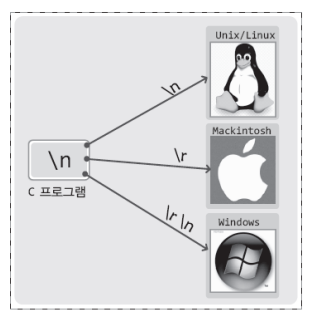
- 텍스트 스트림이 별도로 존재하는 이유

C언어에서는 개행을 \n으로 표시한다. 그런데 이는 c언어만의 약속이다. 그렇다면 다른 환경에서는 개행을 어떻게 표시할까?

\* MS-DOS(Windows)의 파일 내 개행: \r\n

\* Mac의 파일 내 개행: \r

\* Unix 계열의 파일 내 개행: \n



개행 정보를 정확히 저장하기 위해서는 위와 같은 종류의 변환 과정을 거쳐야 한다. 텍스트 모드로 데이터를 입출력 하면 이러한 형태의 변환이 운영체제에 따라서 자동으로 이뤄진다.

**3. 파일 입출력 함수의 기본**

- 파일 입출력 함수들

int fputc(int c, FILE \* stream); // 문자 출력

int fgetc(FILE \* stream); // 문자 입력

int fputs(const char \* s, FILE \* stream); // 문자열 출력

char \* fgets(char \* s, int n, FILE \* stream); // 문자열 입력

매개변수 stream에 FILE 구조체의 포인터를 인자로 전달하여 파일대상의 입출력을 진행할 수 있다.

텍스트, 실내, 화면, 닫기이(가) 표시된 사진

자동 생성된 설명

위 예제의 13, 14행에서는 문자열을 출력하고 있으며, 그 문자열에는 개행문자도 포함되어 있다. 따라서 5행에서 보이듯이 반드시 텍스트 모드로 파일을 개방해야한다. 그럼 이어서 위 예제의 실행을 통해서 생성된 파일에 저장된 데이터를 읽어보겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

위의 두 예제를 통해서 주목할 사실이 하나 더 있다. 그것은 첫 번째 예제에서 출력한 문자열의 형태가 “My name is Hong \n”, “Your name is Yoon \n”으로 문자열의 끝에 \n이 존재한다는 것이다. 문자열이 파일에 저장될 때에는 문자열의 끝을 의미하는 널 문자는 저장되지 않는다. 때문에 파일에서는 개행을 기준으로 문자열을 구분한다. 즉 아래 예제에서는 총 두 개의 문자열을 읽어들이기 위해서 fgets 함수를 2번 호출했는데, 매번 호출이 될 때마다 개행 문자를 만날 때까지 문자열을 읽어 들이게 된다. 따라서 fgets 함수의 호출을 통해서 읽어 들일 문자열의 끝에는 반드시 \n 문자가 존재해야한다.

- feof 함수 기반의 파일복사 프로그램

때로는 파일의 마지막에 저장된 데이터까지 모두 읽어 들여야 하는 상황이 존재한다. 그리고 이를 위해서 파일의 끝을 확인하는 방법이 필요한데, 다음 함수가 이러한 목적으로 정의된 함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이 함수는 인자로 전달된 FILE 구조체의 포인터를 대상으로, 더 이상 읽어 들일 데이터가 존재하지 않으면(파일의 끝까지 모두 읽어 들인 상태이면) 0이 아닌 값을 반환한다. 따라서 다음의 파일복사 프로그램과 같이 파일의 끝을 확인해야 하는 경우에 유용하게 사용이 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

5,6행: 파일 src.txt를 읽기 모드로, 파일 dst.txt를 쓰기 모드로 개방하고 있다. 이는 src.txt의 내용을 dst.txt라는 이름의 파일을 생성해서 복사하기 위함이다. 따라서 여러분이 복사하고자 하는 파일의 이름으로 파일 이름을 대체해서 실행하기 바란다.

14,15행: 파일의 내용을 한 문자씩 복사하고 있다. 이는 fgetc 함수가 EOF를 반환할 때까지 계속되는데, fgetc 함수는 파일의 끝에 도달해서 더 이상 읽을 데이터가 없거나 오류가 발생하는 경우에 EOF를 반환한다.

17행: fgetc 함수가 EOF를 반환했다고 해서 무조건 파일의 끝에 도달했다고 판단할 수는 없다. 오류가 발생하는 경우에도 EOF가 반환되기 때문이다. 따라서 이와 같이 feof 함수의 호출을 통해서 EOF의 반환 원인을 확인할 필요가 있다.

복사가 제대로 완료된다면 파일복사 완료하는 문자열의 출력을 확인하게 된다. 그럼 이어서 예제 하나를 더 보이겠다. 이 예제 역시 텍스트 파일을 복사한다. 다만 위의 예제와 달리 문자가 아닌 문자열 단위로 복사를 진행할 뿐이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

fgets 함수는 파일의 끝에 도달해서 더 이상 읽을 데이터가 존재하지 않거나 오류가 발생하는 경우에 NULL을 반환한다. 때문에 이 예제에서도 파일복사의 성공을 확인하기 위해서 feof 함수를 호출하였다.

- 바이너리 데이터의 입출력: fread, fwrite

지금까지 설명한 입출력 함수들은 텍스트 데이터의 입출력을 진행하는 함수들이다. 반면에 이번에 설명하는 함수들은 바이너리 데이터의 입출력을 진행하는 함수들이다. 참고로 바이너리 데이터의 입출력은 텍스트 데이터의 입출력보다 단순하기 때문에 다양한 함수를 제공하지 않고 있다. 그럼 먼저 바이너리 데이터의 입력에 사용되는 fread함수를 공부하겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위 함수는 다음과 같이 호출이 된다.

int buf[12];

fread((void\*)buf, sizeof(int), 12, fp); // fp는 FILE 구조체 포인터

그리고 위의 fread 함수의 호출문은 “sizeof(int) 크기의 데이터 12개를 fp로부터 읽어 들여서 배열 buf에 저장하라”로 해석이 된다.

즉 fread 함수는 두 번째 전달인자와 세 번째 전달인다의 곱의 바이트 크기만큼 데이터를 읽어 들이는 함수이다. 따라서 위의 fread 함수호출을 통해서 int형 데이터 12개를 fp로부터 읽어서 배열 buf에 저장하게 된다.

그리고 이 함수는 실제로 읽어 들인 데이터의 개수를 반환하는데, 위 문장은 sizeof(int) 크기의 데이터를 12개 읽어 들이는 경우이니, 함수의 호출이 성공을 하고 요청한 분량의 데이터가 모두 읽혀지면 12가 반환된다.

반면 함수의 호출이 성공을 했지만 파일의 끝에 도달을 해서 12개를 모두 익어 들이지 못했거나 오류가 발생하는 경우에는 12보다 작은 값이 반환된다. 그럼 이어서 바이너리 데이터의 출력에 사용되는 fwrite 함수를 소개하겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위 함수는 다음과 같이 호출을 한다.

int buf[7]={1, 2, 3, 4, 5, 6, 7};

fwrite((void\*)buf, sizeof(int), 7, fp);

그리고 위의 fwrite 함수 호출문은 “sizeof(int) 크기의 데이터 7개를 buf로부터 읽어서 fp에 저장해라”는 의미로 해석이 된다.

이제 바이너리 파일을 복사하는 프로그램을 작성해보겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**4. 텍스트 데이터와 바이너리 데이터를 동시에 입출력 하기**

하나의 파일을 대상으로 입출력 할 데이터가 텍스트 데이터와 바이너리 데이터 둘로 이뤄져 있다면 어떠한 방법을 택해서 입출력을 해야 할까? 지금부터는 텍스트 데이터인 문자와 문자열, 그리고 바이너리 데이터인 int형 정수 하나를, 하나의 파일을 대상으로 동시에 입출력 해야 하는 상황에 대해서 이야기하겠다.

- 서식에 따른 데이터 입출력: fprint, fscanf

두 개의 텍스트 데이터와 하나의 바이너리 데이터를 입출력 해야 하는 상황에서 제일 먼저 생각할 수 있는 방법은 fscanf 함수와 fprintf 함수의 호출이다. 이 두 함수는 scanf, printf 함수와 유사하다. 다만 입출력의 대상이 콘솔이 아닌 파일이라는 점에서 차이가 있다. 그럼 먼저 fprintf 함수의 호출방법을 보이겠다.

char name[10]=”홍길동”; // 텍스트 데이터

char sex=’M’; // 텍스트 데이터: 남자 M, 여자 F

int age=24; // 바이너리 데이터

fprintf(fp, “%s %c %d”, name, sex, age); // fp는 FILE 구조체 포인터

위의 fprintf 함수의 호출문이 printf 함수의 호출문과 차이를 보이는 부분은 FILE 구조체의 포인터가 첫 번째 전달인자라는 점이다. 그래서 printf 함수와 달리 fprintf 함수는 첫 번째 인자로 전달된 FILE 구조체의 포인터가 지칭하는 파일로 출력이 이뤄진다. 위의 fprintf 함수 호출 문장도 두 번째 이후의 전달인자를 통해서 다음 문자열이 만들어진다.

“홍길동 M 24”

그리고 이렇게 만들어진 문자열이 첫 번째 전달인자가 가리키는 파일에 저장이 된다. 결국 텍스트 데이터와 바이너리 데이터를 하나의 문자열로 묶어서 저장하는 셈이다. 그럼 예제를 통해서 이러한 내용을 확인해보자. 다음 예제에서는 텍스트 데이터인 이름과 성별, 그리고 바이너리 데이터인 나이 정보를 하나의 파일에 저장하는 방법을 fpritnf 함수를 기준으로 보여준다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트, 전자기기, 검은색이(가) 표시된 사진

자동 생성된 설명

위의 예제를 통해서 생성된 파일 friend.txt를 직접 열어서 프로그램 사용자가 입력한 데이터가 저장되었음을 확인할 수도 있다. 그러나 우리는 fscanf 함수의 호출을 통해서 이를 확인하고자 한다. fprintf함수의 호출을 통해서 저장된 데이터는, 동일하게 서식을 지정해서 fscanf 함수의 호출을 통해서 읽어들일 수 있다. fscanf 함수의 호출방식은 다음과 같다.

char name[10];

char sex;

int age;

fscanf(fp, “%s %c %d”, name, &sex, &age);

위의 fscanf 함수 호출문이 scanf 함수 호출문과 차이를 보이는 부분은 첫 번째 인자로 FILE 구조체의 포인터가 전달된다는 점이다. 따라서 scanf 함수와 달리 fscanf 함수는 첫 번째 인자로 전달된 포인터가 지칭하는 파일로부터 데이터를 읽어 들인다. 그리고 fscanf 함수는 파일의 끝에 도달하거나 오류가 발생하면 EOF를 반환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

- 텍스트와 바이너리 데이터의 집합체인 구조체 변수의 입출력

앞서 보인 예제에서는 다음의 데이터들을 대상으로 파일 입출력을 진행하였다.

char name[10], char sex, int age

그런데 실제 프로그램에서는 이들을 다음과 같이 구조체로 묶어서 정의하는 것이 보통이다. 때문에 구조체 변수 단위로의 파일 입출력에 대해 고민할 필요가 있다.

typedef struct fren

{

char name[10];

char sex;

intage;

} Friend

아무래도 구조체 변수 단위의 입출력이 구조체 멤버 단위의 입출력보다 입출력의 형태가 단순해질 수밖에 없다. 그렇다면 어떻게 해야 구조체 변수를 통째로 저장하고 통째로 읽어 들일 수 있겠는가? 구조체 변수를 하나의 바이너리 데이터로 인식하고 처리하면 가능하다. 그러면 fwrite 함수를 통해서 통째로 저장하고 fread 함수를 통해서 통째로 복원할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

