12\_파일 입출력

학습목표

스트림의 개념을 이해한다.

형식화된 입출력을 이해한다.

파일 입출력 함수들을 사용할 수 있다.

이진 파일을 사용할 수 있다.

파일에 임의 접근할 수 있다.

1. 스트림

일반적으로 프로그램에서는 화면이나 키보드, 파일등의 입출력 장치에 대하여 데이터를 쓰거나 읽게 된다. 입출력 장치들은 상당히 다양한 방식으로 데이터를 주고받지만 C에서는 스트림(stream)이라는 개념을 사용하여서 동일한 방법으로 입출력 할 수 있다.

스트림이란 모든 입력과 출력을 바이트(byte)들의 흐름으로 생각하는 것이다. 어떤 입출력 장치던지 상관없이 바이트 단위로 입출력이 이루어진다. 스트림의 최대 장점은 장치 독립성이다. 입출력 장치에 상관없이 동일한 함수를 사용하여 프로그램을 작성할 수 있다.

1. 표준 입출력 스트림

프로그램의 동작에 필수적인 몇 개의 기본적인 스트림은 프로그램 실행 시에 자동으로 생성된다. 이것을 표준 입출력 스트림(standard input/output stream)이라고 한다.

Stdin 표준 입력 스트림 키보드 scanf

Stdput 표준 출력 스트림 화면 printf

Stderr 표준 에러 스트림 화면

1. 파일도 스트림에 연결시킬 수 있다.

C에서 파일은 곧 스트림이다. 앞에서 스트림에 뭐든지 붙일 수 있다고 했는데 파일도 예외가 아니다. 운영 체제에서는 하나의 프로세스 당 100여개의 스트림이 지원되고 스트림 중 0, 1, 2번은 스트림은 stdin, stdout, stderr로 미리 정의된다. 3번 스트림부터 나머지 스트림은 개발자가 파일을 연결해서 사용할 수도 있고 여러 가지 통신 포트를 연결하여 사용할 수 있다.

1. 파일의 기초

지금까지는 키보드와 화면을 통해 스트림을 사용처를 알아봤지만 실제 응용 프로그램에서는 데이터나 현재의 설정 정보를 저장하기 위하여 디스크에 저장되는 파일을 많이 사용한다. 파일을 사용하는 이유는 무엇일까?

지금까지 우리는 모든 데이터를 변수에 저장하였고 이는 영구적인 기억장치가 아니다. 따라서 데이터를 영구적으로 보관하려면 디스크와 같은 보조 기억 장치에 보관해야 한다. C에서는 디스크에 파일을 생성시켜서 데이터를 보관할 수 있다.

1. 파일의 개념

C에서의 모든 입출력은 스트림을 통해 이루어지고 파일도 예외는 아니다. 파일도 스트림으로 취급되기 때문에 파일도 일련의 연속된 바이트라고 생각하면 된다. 따라서 파일에 대한 입출력도 표준 입출력과 동일한 함수들로 이루어진다. 이것이 바로 스트림의 장점이다.

1. 파일의 유형

파일의 유형에는 텍스트 파일과 이진 파일이 있다.

텍스트 파일(text file)은 사람이 읽을 수 있는 텍스트가 들어 있는 파일이다. C 프로그램 소스 파일이나 메모장 파일이 텍스트 파일의 예이다. 텍스트 파일에는 문자들을 나타내는 아스키 코드들이 들어 있다. 텍스트 파일이 중요한 이유는 모니터, 키보드, 프린터 등이 모두 문자 데이터만 처리하기 때문이다.

이진 파일(binary file)은 사람이 읽을 수는 없으나 컴퓨터는 읽을 수 있는 파일이다. 즉 문자 데이터가 아니라 이진 데이터가 직접 저장되어 있는 파일이다. 이진 파일은 텍스트 파일과는 달리 라인들로 분리되지 않는다. 라인의 끝을 표시할 필요도 없다. 모든 문자들은 특별한 의미를 가지지 않고 데이터로 취급된다. 이진 파일은 특정 프로그램에 의해서만 판독이 가능하다. C프로그램 실행 파일, 사운드 파일, 이미지 파일 등이 이진 파일의 예이다.

1. 파일 처리의 개요

프로그램에서 파일을 연다(open)는 것은 파일에서 데이터를 읽거나 쓸 수 있도록 모든 준비를 마치는 것을 의미한다. 내부적으로는 파일과 연결된 스트림을 만들게 된다. 파일을 연 다음에는 데이터를 읽고 수정할 수 있다. 파일을 사용한 후에는 파일을 닫아야(close) 한다.

파일 열기 -> 파일 읽기와 수정 -> 파일 닫기

1. 파일 열기

FILE \*fp;

fp = fopen(“text.txt”, “w”); //text(name)라는 이름의 파일을 w(mode)모드로 연다. FILE포인터를 반환한다.

Fopen()은 주어진 파일 이름을 가지고 파일을 생성하여 이 파일을 가리키는 FILE포인터를 반환한다. FILE은 stdio.h에 선언된 구조체이다. FILE구조체에는 파일의 열기, 읽기, 쓰기, 닫기에 관련된 모든 상태 정보가 들어 있다. 각각의 파일에 대하여 FILE 포인터가 하나씩 필요하다. Fopen()이 실패하면 NULL포인터가 반환된다.

첫 번째 매개 변수인 name은 파일의 이름을 나타내는 문자열이다. 큰 따옴표로 둘러싸인 문자열 상수로 직접 입력할 수도 있고 배열에 저장한 뒤에 배열을 넘길 수도 있다.

두 번째 매개 변수인 mode는 파일을 여는 모드를 의미한다. 파일 모드는 파일과 관련된 여러 가지 선택 사항을 결정하는 문자열이다. 모드는 파일의 유형이 텍스트 파일인지 이진 파일인지, 파일에 데이터를 쓸 것인지 파일에서 데이터를 읽을 것인지를 나타내는데 사용된다.

파일 모드

“r” 읽기 모드로 파일을 연다

“w” 쓰기 모드로 파일을 연다. 만약 파일이 존재하지 않으면 파일을 생성한다. 파일이 이미 존재하면 기존의 내용이 모두 지워진다.

“a” 쓰기 모드로 파일을 연다. 기존의 파일의 내용을 지우지 않고 끝에 내용을 추가시킨다.

이진 파일로 파일을 열려면 b를 추가한다. “rb”, “wb”, “ab” 등

1. 파일 닫기

사용이 끝나면 반드시 파일을 닫아야 한다. 파일을 닫는 함수는 fclose()이다.

Fclose(name); //stream에 의하여 지정된 파일을 닫는다.

성공적으로 파일을 닫는 경우에는 0이 반환된다. 만약 실패한 경우에는 -1이 반환된다.

1. 텍스트 파일 읽고 쓰기
2. 한 글자씩 쓰기

텍스트 파일에 한 글자씩 쓰는 함수는 fputc()이다. Fputc.c참조

1. 한 글자씩 읽기

Fgetc(fp): fp에서 하나의 문자를 읽어서 반환한다.

인수 fp는 fopen()을 이용하여 파일을 열 때 반환되는 FILE에 대한 포인터이다.

1. 한 줄씩 읽고 쓰기

텍스트 파일에서 한 줄씩 읽고 쓰려면 fgets()와 fputs()를 사용한다. Fputget.c 참조

1. 형식화된 입출력

정수나 실수는 화면에 문자열로 변환되어서 출력되는 것처럼 파일에서도 정수나 실수를 문자열을 바꾸어서 저장하는 것이 보통이다. 이러한 종류의 입출력을 형식화된 입출력이라고 한다. 형식화된 입출력은 프로그래머가 특정한 형식을 지정하고 이 형식으로 파일에 입출력 하는 것이다. 형식화된 입출력은 fprintf()와 fscanf()을 이용하여 이루어진다.

1. 이진 파일

텍스트 파일(text file)에서는 모든 정보가 문자열로 변환되어서 파일에 기록되었다.

이진 파일(binary file)은 데이터가 직접 저장되어 있는 파일이다. 즉 정수 123456이 있으면 문자열로 변환되지 않고 이진수 상태로 파일에 기록되는 것이다. 이진 파일의 장점은 효율성이다. 이진 파일은 변환과정을 거치지 않기 때문에 시간이 단축되고 텍스트 파일에 비해 저장 공간도 적게 차지한다.

이진 파일의 단점은 인간이 파일의 내용을 파악하기 힘들다는 점이다. 문자 데이터가 아니므로 모니터나 프린터로 출력하는 것이 불가능하다. 텍스트 파일은 컴퓨터 기종이 달라도 아스키 코드로 되어 있기 때문에 다른 컴퓨터에서도 이식이 가능하다. 그러나 이진 파일의 경우, 정수나 실수 데이터를 표현하는 방식이 컴퓨터 시스템마다 다를 수 있기 때문에 이식성이 떨어진다.

1. 이진 파일 사용하기

이진 파일을 생성하려면 fopen()에서 파일 모드에 “b”를 붙이면 된다.

이진 파일에서 데이터를 읽고 쓰려면 fread()와 fwrite()를 사용한다.

1. 이진 파일 읽고 쓰기

쓰기: fwrite(name, sizeof(자료형 크기), SIZE(배열의 크기), fp);

읽기: fread(name, sizeof(자료형 크기), SIZE(배열의 크기), fp);

1. 임의 접근
2. 순차 접근과 임의 접근

지금까지의 파일 입출력 방법은 모두 데이터를 파일의 처음부터 순차적으로 읽거나 기록하는 것이었다. 이것을 순차 접근(sequential access)방법이라고 한다. 이러한 방법은 한번 읽은 데이터를 다시 읽으려면 현재의 파일을 닫고 파일을 다시 열어야 한다. 또한 앞부분을 읽지 않고 중간이나 마지막으로 건너뛸 수도 없다. 또 다른 파일 입출력 방법으로 임의 접근(random access)방법이 있다. 임의 접근 방법은 파일의 어느 위치에서든지 읽기와 쓰기가 가능하다.

1. 임의 접근의 원리

모든 파일에는 파일 포인터(file pointer)라는 것이 존재한다. 파일 포인터는 64비트의 값으로 읽기와 쓰기 동작이 현재 어떤 위치에서 이루어지는지를 나타낸다. 새 파일이 만들어지게 되면 파일 포인터는 값이 0이고 이것은 파일의 시작 부분을 가리킨다. 기존의 파일의 경우, 추가 모드에서 열렸을 경우에는 파일의 끝이 되고, 다른 모드인 경우에는 파일의 시작 부분을 가리킨다.

파일에서 읽기나 쓰기가 수행되면 파일 포인터가 갱신된다. 우리가 입출력 함수를 사용하면 그 함수의 내부에서 파일 포인터의 값이 변경된다. 사실 프로그래머는 파일 포인터에 대하여 크게 신경 쓸 필요는 없다. 데이터를 임의의 위치에서 읽고 싶을 경우에는 위치 표시자를 조작해야 만이 파일의 원하는 임의의 위치에서 읽을 수 있다. 위치 표시자를 조작하는 함수는 fseek()이다. 현재의 위치 표시자는 ftell(fp)을 호출하면 알 수 있다.

형식: int fseek(FILE \*fp, long offset, int origin);

Fp는 FILE에 대한 포인터이고, offset은 기준 위치로부터 위치 표시자가 이동하는 거리를 나타낸다. Offset은 long형이기 때문에 정수 상수를 인수로 사용하는 경우에는 3000L과 같이 L를 붙이는 것이 좋다. Offset이 양수면 앞으로 가고 음수면 뒤로 간다. Origin은 위치 표시자를 이동시키는 기준 위치를 나타낸다. origin에는 3개의 값을 사용할 수 있다.

SEEK\_SET 0 파일의 시작

SEEK\_CUR 1 현재 위치

SEKK\_END 2 파일의 끝

Fseek()는 성공하면 0를 반환하고 실패한 경우에는 0이 아닌 값을 반환한다. 파일이 열렸을 경우, 파일 포인터의 초깃값은 0이다.

도대체 어떤 경우에 순차 접근이 아닌 임의 접근을 할 필요가 있을까? 음악 파일을 듣거나 동영상 파일을 보는 경우, 중간을 건너뛰고 뒷부분으로 가는 경우가 있다. 보통 멀티미디어 파일들은 그 크기가 커서 메모리에 전부 적재하지 못한다. 따라서 그런 경우에는 파일의 위치 표시자를 뒷부분으로 보낸 후에 그 위치부터 읽으면 된다.

Fseek()를 사용하면 파일의 크기를 알 수 있다. Fseek.c참조