**스트림**: 입력과 출력을 **바이트(byte)들의 흐름**으로 생각하는 것

1. **스트림에 버퍼가 포함**
2. **장치 독립성이 장점이다!!(장치에 상관없이 해당 스트림을 사용해 입력과 출력 가능)**
3. 키보드에 입력해서 컴퓨터로 넘어가기 전, 컴퓨터에서 모니터에 출력 되기 전에 버퍼에 내용이 저장되어 있다. (버퍼가 꽉 차면 경고음이 울린다.)

|  |  |  |
| --- | --- | --- |
| 이름 | 스트림 | 연결 장치 |
| stdin | 표준 입력 스트림 | 키보드 |
| stdout | 표준 출력 스트림 | 모니터의 화면 |
| stderr | 표준 오류 스트림 | 모니터의 화면 |

\*헤더 파일 <stdio.h>은 표준입출력 라이브러리로 위의 스트림 3개를 모두 포함하고 있다.

\*stdout은 stderr를 포함하고 있다. 하지만 stderr의 우선순위가 더 높다.

4) 입출력 함수



\*표준 스트림 vs. 일반 스트림

: **표준 스트림**은 스트림을 따로 적어줄 필요 없음. 즉, stdin,stdout이런거 쓸 필요 없이stdio.h사용하면 알아서 필요한 스트림이 적용. 보통 키보드와 모니터화면과 관련된 스트림이다.

**일반스트림**은 스트림을 명시적으로 적어줘야 함. , stdin,stdout 이런거 적어야 함

\*형식이 있는 입출력 함수는 “%s” 이런식으로 형식을 지정해줘야 하는 것

**printf(“%-10.3f”, value)**

1. 플래그: 정렬 방법이나 공백 채움 방법

|  |  |
| --- | --- |
| - | 왼쪽 정렬  **기본** 오른쪽 정렬 |
| + | 양수일 때는 +부호, 음수일 떄는 – 부호  기본: 음수일 때만 – 붙임 |
| 0 | 출력하는 폭의 남은 공간을 0으로 채움  정밀도 지정 시 무시  기본:채우지 X |
| ‘ ‘  (공백) | 양수일 때 부호 출력 안 하고 공백으로  기본:공백 출력X |
| # | 8진수->0 , 16진수->0x  기본: 붙이지 않음 |

1. 필드폭: 총 칸 수,기본10
2. 정밀도: 소수점 이하 얼마까지 나타낼지
3. 형식:

|  |  |
| --- | --- |
| %d | 부호 있는 10진수 형식 (정수) |
| %i | 부호 있는 10진수 형식(정수) |
| %u | 부호 없는 10진수 형식(정수) |
| %o | 부호 없는 8진수 형식 |
| %x | 부호 없는 16진수 형식(소문자) |
| %X | 부호 없는 16진수 형식(대문자) |
| %f | 소수점 고정 표기 형식(실수) |
| %e | 지수표기 형식, 지수 부분을 e로 표시 |
| %E | 지수표기 형식, 지수 부분을 E로 표시 |
| %g | %e 와 %f 중 더 짧은 거 출력 |
| %G | %E 와 %f 중 더 짧은 거 출력 |
| %p | 포인터 형식 |

5) **scanf(“%d”, 변수의 주소값):** 사용자가 입력하는 것은 **원래 문자열 형태인데 원하는 형식대로 변환하여 변수에 저장**

1. 문자/문자열 형식 지정자 종류

|  |  |
| --- | --- |
| %c | char형으로 입력 받음(문자 한 개 입력 받음) |
| %s | 공백 단위로 끊어서 문자열 입력 받음 |
| %[abc] | 대괄호 안에 있는 문자로만 이루어진 문자열 읽어 들임 |
| %[^abc] | 대괄호 안에 있는 문자들을 제외한 문자로만 이루어진 문자열 읽어 들임 |
| %[0-9] | 0-9까지의 범위에 있는 문자들로 이루어진 문자열 읽어 들임 |

1. scanf는 공백 단위로 읽어 들이므로 여러 가지 경우를 주의해야 한다.

경우1- scanf(“%s”,&i): 스페이스 사용하면 X(스페이스 포함X)

경우2- scanf(“%s %s”, &i,&k): 반드시 스페이스 사용해줘야 각 변수에 저장 가능

경우3- scanf(“%s%c%s”, &i,&k,&n): 문자열+스페이스+문자열 입력 시, 스페이스가 변수 k에 저장

1. 입력값을 저장할 **변수의 주소**

\*배열의 경우 **배열의 이름이 그 배열의 시작 주소이다**

**4)** printf() 처럼 형식 제어 문자열 끝에 줄바꿈 문자\n을 쓰는 것을 주의!

**\*scanf(“%d\n”,&i);의 의미는 해당 문자가 반드시 입력 되어야 함을 의미**

**5) 반환하는 것은 올바른 형식으로 입력된 변수의 개수이다.**

**파일**: C에서는 일련의 연속된 바이트, 하드 디스크에 저장되는 단위로 데이터 보존

1. 모든 데이터가 바이트로 변환되어 파일에 저장
2. 이 바이트를 어떻게 해석할 지가 프로그래머의 역할과 책임

**텍스트 파일**: **사람이 읽을 수 있는 텍스트**가 들어있는 파일 ex)소스 파일, 메모장 파일

1. 아스키 코드 이용해 저장
2. 연속적인 라인들로 구성

+) 운영체제 별 c언어에서 \n 표현법 ->윈도우,ms\_dos: \r\n

->유닉스: \n

->매킨토시: \r

**이진 파일:** 사람은 읽을 수 없고, **컴퓨터가 읽을 수 있는 파일**

ex)프로그램 실행파일,사운드,이미지 파일

1. 이진 데이터가 직접 저장
2. 모든 데이터는 문자열로 변환되지 않고 입출력
3. 라인 분리X
4. 이진 파일은 특정 프로그램에 의해서만 판독 가능

파일 처리 개요: **파일 열기->파일 읽기 및 쓰기->파일 닫기**

1. 디스크 파일은 **FILE 구조체** 이용해 접근

\*파일 열기 위한 정보에 접근하는데 정보의 길이들이 모두 다를 수 있기에 구조체에 저장

1. **FILE 구조체 가리키는 포인터-> 파일 포인터**

**파일 열기**

**FILE \*fp;** ->파일 구조체인 FILE 타입 포인터 변수 fp

**fp = fopen(“test.txt”,”w”);** ->’파일 이름’과 ‘파일 모드’

|  |  |
| --- | --- |
| “r” | 읽기 모드/파일 반드시 존재해야 함(오류남) |
| “w” | 쓰기 모드/ 새 파일 생성, 이미 존재하면 기존 내용 지워짐 |
| “a” | 추가 모드/ 파일 없으면 새로 만듬 |
| “r+” | 읽기 모드로 열고 쓰기 모드로 전환 가능(파일 반드시 존재해야 함) |
| “w+” | 쓰기 모드로 열고 읽기 모드로 전환 가능(기존 내용 지워짐) |
| “a+” | 추가 모드로 열고 읽기 모드로 전환 가능/데이터 추가 시, EOF 마커를 가장 뒤로 옮김 |
| “t” | 텍스트 파일 모드로 염 |
| “b” | 이진 파일 모드로 염 |

\* 추가 모드의 경우 절대 기존 데이터가 지워지지 않음

\* + 모드는 수정모드라 하기도 하고, 읽기->쓰기, 쓰기->읽기로 전환하는 경우 반드시

**fflush(): 파일과 관련된 버퍼 비움(=버퍼에 있는 내용을 파일에 보냄)**

ex)

**fsetpos(): 파일에서 위치 지정**

**fseek(): 파일에서 위치 찾음**

**rewind(): 파일 다시 감음** 중 하나를 호출해야 함

**파일 닫기: fclose(fp)**

1. 파일을 닫지 않으면 다른 사용자가 파일 사용 불가

**파일 입출력 함수**

|  |  |  |
| --- | --- | --- |
| 종류 | (파일에서)입력 함수 | (파일에)출력 함수 |
| 문자 | int fgetc(FILE \*fp) | int fputc(int c, FILE \*fp) |
| 문자열 | char \*fgets(char \*buf, int n, FILE \*fp) | char \*fputs(const char \*buf, FILE \*fp) |
| 서식화된 입출력 | int fscanf(FILE \*fp,”%d”,&n) | int fprintf(FILE \*fp,”%d”,n) |
| 이진 데이터 | size\_t fread(char \*buffer, int size, int, count, FILE \*fp) | size\_t fwrite(char \*buffer, int size, int, count, FILE \*fp) |

\*위 3개는 텍스트 입출력 함수 라고 할 수 있고, 맨 아래 하나는 이진 데이터 입출력 함수이다.

\*int count는 읽은 항목의 개수를 반환

**버퍼링**

버퍼: 파일로부터 읽고 쓰는 **데이터의 임시 저장소**로 이용되는 메모리의 블록

(블록: 바이트 여러여러 모인 것)

\*디스트 드라이브: 블록 단위 장치, 블록 단위로 입출력 해야 효율적(버퍼가 쓰이는 이유)

(1024바이트 블록이 일반적)

fflush(fp): 버퍼의 내용이 디스크 파일에 써진다.

(보통 한 작업을 끝내고 다른 작업을 할 때 버퍼의 내용을 비워야 하기 때문에 사용한다.)

setbuf(fp,NULL): 스트림의 버퍼를 직접 지정하는 함수, 버퍼자리에 NULL쓰면 버퍼 제거의 의미

순차 접근: 처음부터 순차적으로

임의 접근: 어느 위치에서든지

**파일 포인터: 동작이 현재 어떤 위치에서 이루어지는지 나타냄**

1. **강제적으로 파일 포인터 이동**
2. **int fseek(FILE \*fp**, long offset**(거리),** int origin**(기준 위치));: 파일 포인터 위치 이동**

SEEK\_SET **(0) :파일의 시작**

SEEK\_CUR **(1) :현재 위치**

SEEK\_END **(2) :파일의 끝**

1. **long ftell(FILE \*fp); :파일 포인터의 현재 위치 알려줌**