1) 정수 자료형

- **정수 자료형의 크기 및 표현할 수 있는 값의 범위**
  - VS 2017 기준

부호	자료형	메모리 크기	값의 범위
있음	char	1 byte	-128 ~ 127
	short	2 bytes	$-2^{15} \sim 2^{15} - 1$
	int	4 bytes	$-2^{31} \sim 2^{31} - 1$
	long	4 bytes	$-2^{31} \sim 2^{31} - 1$
없음	unsigned char	1 byte	0 ~ 255
	unsigned short	2 bytes	0 ~ 65,535
	unsigned int	4 bytes	0 ~ 4,294,967,295
	unsigned long	4 bytes	0 ~ 4,294,967,295

2) 동적 메모리 사용 절차

- **메모리 사용 절차**
  - 메모리 사용을 위해서는 메모리를 **할당**하고 **해제**하는 과정 필요
    - ✓ **할당:** 필요한 메모리를 시스템으로부터 받기
    - ✓ 사용: 할당된 메모리 사용
    - ✓ **해제:** 사용이 끝난 메모리를 시스템에 반납
  - 정적 할당 vs. 동적 할당
    - ✓ **정적 할당:** 이 과정이 프로그램과 함수의 실행 및 종료에 따라 자동으로 수행됨
    - ✓ **동적 할당:** 프로그래머가 필요한 과정을 코드에 **명시적으로 작성**해 주어야 함

2) 동적 메모리 사용 절차

- **동적 할당 기본 예제**
  - 동적으로 메모리를 할당하고 해제하는 라이브러리 함수는 <stdlib.h> 헤더 파일에 선언

```
#include <stdlib.h>    // 동적 메모리 관련 함수 사용을
                       // 위한 헤더
int main() {
   int *p = NULL;      // 동적 메모리 접근을 위한 포인터 변수
   p = (int *) malloc( 5*sizeof(int) ); // 동적 메모리 할당

   p[0] = 1;           // 동적 메모리 사용: 배열 형태
   *(p+2) = 3;         // 동적 메모리 사용: 포인터 형태

   free(p);            // 동적 메모리 해제
   return 0;
}
```

1) 파일 입출력 개요

- **파일 저장 방식에 따른 구분**

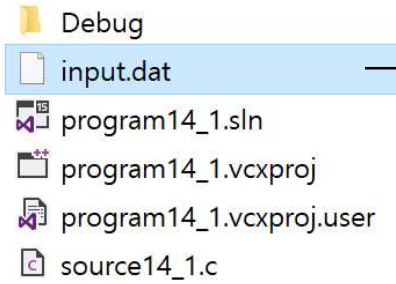
	텍스트(text) 파일	이진(binary) 파일
특성	✓사람이 인식할 수 있는 문자를 담고 있는 파일 ✓특별한 응용 프로그램 없이 내용 볼 수 있음 ✓메모장 프로그램을 통해 파일을 열었을 때, 읽을 수 있는 문자들로 표현됨 ✓모든 데이터가 문자열로 변환되어 기록됨	✓컴퓨터가 인식할 수 있는 데이터를 담고 있는 파일 ✓특정 응용 프로그램을 이용해야 액세스할 수 있는 파일(예, ppt 파일) ✓메모장 프로그램을 통해 파일을 열었을 때, 알아볼 수 없는 이상한 문자들로 표시 ✓수치 데이터가 문자로 변환되지 않고 곧바로 수치로 저장 ✓텍스트 파일보다 저장 공간을 적게 차지 ✓읽고 쓰기가 빠름

1) 파일 입출력 개요

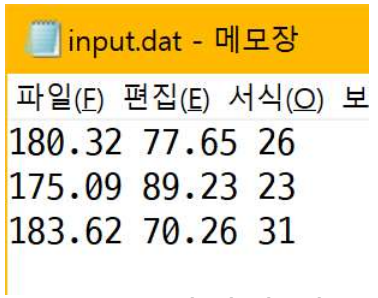
- **파일 입출력 따라 해보기**

  ✓ input.dat 파일로부터 키, 몸무게, 나이 정보를 입력받아

  ➔ 이 내용을 그대로 output.dat 파일에 출력하기



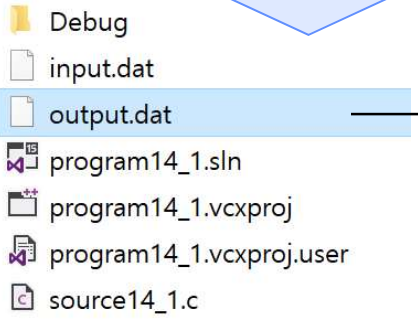
<현재 작업 폴더 내용 – 실행 전>



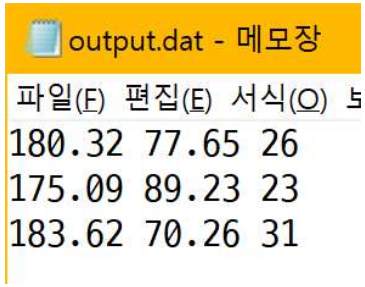
메모장으로 작성

<input.dat 파일의 내용>

(새로 생성)



<현재 작업 폴더 내용 – 실행 후>



<output.dat 파일의 내용>

1) 파일 입출력 개요

- **예제 프로그램 14.1**

```
#include <stdio.h>

int main() {
   double height, weight;
   int age, i;
   FILE *fp1, *fp2;                        // FILE 구조체 포인터 선언

   fp1 = fopen("input.dat", "r");          // input.dat 파일 열기
   fp2 = fopen("output.dat", "w");         // output.dat 파일 열기

   for (i = 0; i < 3; i++)   {
      fscanf(fp1, "%lf %lf %d", &height, &weight, &age); // 입력 받기
      fprintf(fp2, "%.2f %.2f %d\n", height, weight, age); // 출력하기
   }

   fclose(fp1);                               // input.dat 파일 닫기
   fclose(fp2);                               // output.dat 파일 닫기
   return 0;
}
```

2) 파일 입출력 절차

- **FILE 포인터 선언**
  - **FILE**은 파일 입출력 시 필요한 정보를 담은 구조체
  - 파일 입출력 시 각 파일마다 하나의 FILE 포인터를 연결하여 사용
  - 선언 형식

```
FILE *fp;   ⇨ 주의 - FILE은 반드시 대문자!
```

  - 표준 스트림
    - ✓ 표준 입출력 장치(키보드, 모니터)도 논리적으로 파일로 간주
    - ✓ 표준 스트림에 대한 FILE 포인터 명 (정해져 있음)

FILE 포인터 이름	스트림	의미
stdin	표준 입력 스트림	키보드로부터 입력 받음
stdout	표준 출력 스트림	모니터로 결과 출력
stderr	표준 오류 출력 스트림	모니터로 오류 메시지 출력

2) 파일 입출력 절차

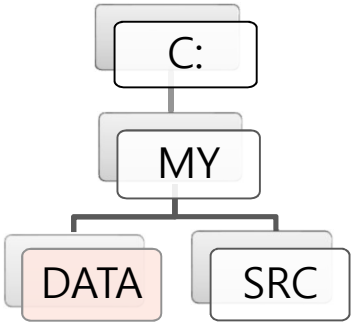
- **파일 열기: fopen() 함수**
  - 해당 파일에 대한 연결 요청을 의미
  - 해당 파일을 사용할 수 있도록 파일 포인터를 반환

함수원형	`FILE *fopen(char *filename, char *filemode);`	
함수인자	filename	연결 할 파일 이름
	filemode	파일 접근 방식에 따른 모드
반환값	✓ 파일열기에 성공 → FILE 포인터를 반환 ✓ 파일열기에 실패 → NULL을 반환	

```
1) FILE *fp = fopen("test.txt", "r");
2) FILE *fp2 = fopen("C:\\MY\\DATA\\test.txt", "a");
3) FILE *fp3 = fopen("..\\DATA\\test.txt", "w");
4) FILE *fp4 = fopen("DATA\\test.txt", "r");
```

2) 파일 입출력 절차

- **함수인자 filename**
  - 특정한 위치를 지정하지 않는 경우 → 현재 작업 폴더
    - ✓ 현재 작업 폴더 = (Visual studio에서는) 현재 소스 프로그램이 위치한 폴더
    - ✓ 예) `fopen("test.txt", "r");`



< 예시 폴더 구조 >

  - 절대경로
    - ✓ 드라이브 명부터 해당 파일이 있는 위치까지 전체 경로
    - ✓ 예) `fopen("C:\\MY\\DATA\\test.txt", "a");`

  - 상대경로
    - ✓ 현재 작업 폴더를 기준으로 해당 파일이 있는 위치까지의 경로
    - ✓ 예) `fopen("..\\DATA\\test.txt", "w");  // ..은 상위 폴더`
    - ✓ 예) `fopen("DATA\\test.txt", "r");`

2) 파일 입출력 절차

- **함수인자 filemode**
  - 개방할 파일의 용도에 따라 적합하게 지정해야 함
    - ✓ 적합한 모드 지정은 파일을 잘못 사용하는 것을 막을 수 있음

< 파일 접근 방식에 따른 모드 구분>

텍스트 모드	이진 모드	기능	설명
r	rb	읽기 전용 (read)	✓ 파일을 열 수 없는 경우 → NULL을 반환
w	wb	쓰기 전용 (write)	✓ 파일이 없는 경우 → 새로 빈 파일을 생성 ✓ 같은 이름의 파일이 존재하는 경우 → 해당 파일의 내용 삭제하고 새로 파일 생성
a	ab	추가 전용 (append)	✓ 파일이 없는 경우 → 새로 빈 파일을 생성 ✓ 같은 이름의 파일이 존재하는 경우 → 기존 파일의 마지막 부분에 내용을 추가

✓ 참고) '+' 기호를 붙이면 (예: r+, w+, ab+ 등) 수정 모드(읽기 쓰기 모두 가능)

2) 파일 입출력 절차

- **fopen() 함수 사용 시 주의사항**
  - fopen() 함수 호출 시, 이 함수의 반환 값을 반드시 검사하여 파일이 정상적으로 열렸는지 확인해야 함

```
FILE *fp = fopen("input.dat", "r");
if (fp == NULL)  {       // 파일 열기에 실패한 경우
     printf("Couldn't open file");   // 오류 처리 코드
     return -1;
}
```

2) 파일 입출력 절차

- **파일 닫기: fclose() 함수**
  - 현재 열린 파일과 FILE포인터와의 연결을 해제

함수 원형	**int fclose(FILE *fp);**	
함수 인자	fp	파일 포인터 변수명
반환 값	✓ 파일 닫기에 성공 → 0을 반환 ✓ 파일 닫기에 실패 → EOF를 반환	

```
FILE *fp = fopen("test.dat", "r");
if (fp == NULL) {
        printf("파일열기에 실패했습니다!\n");
        return -1;
}
fclose(fp);
```

3) 텍스트 파일 입출력

- **지정 형식 파일 입력 함수: fscanf()**
  - 형식을 지정하여 파일의 데이터를 읽기 위한 함수
  - 여러 형태의 자료들(정수, 문자, 문자열 등)을 한번에 입력 가능
  - 함수의 첫 번째 인자로 파일 포인터가 사용된다는 것을 제외하고는 scanf() 함수와 사용법 동일

함수원형	**int fscanf(FILE *fp, char *format, …);**	
함수인자	fp	파일 포인터 변수명
	format	형식 제어 문자열
	…	입력하고자 하는 변수리스트
반환 값	✓ 성공 → 입력한 변수의 개수를 반환 ✓ 읽기 실패 → EOF를 반환	

3) 텍스트 파일 입출력

- **fscanf() 함수 사용 예**

```
char name[10] = {'\0'};
double weight;
int age;

FILE *fp = fopen("input.dat", "r");

fscanf(fp, "%s %d %lf", name, &age, &weight);
          // input.dat 파일로부터 데이터를 입력받음
fscanf(stdin, "%s %d %lf", name, &age, &weight);
          // 키보드로부터 데이터를 입력받음
...
```

✓ 첫 번째 인자를 stdin으로 지정하면, 표준입력으로부터 입력

✓ scanf() 함수를 사용한 것과 동일한 기능

✓ stdin, stdout, stderr는 따로 open, close 할 필요 없음 (자동 수행)

3) 텍스트 파일 입출력

- **지정 형식 파일 출력 함수: fprintf()**
  - 지정한 형식에 맞추어 파일로 출력
  - 함수의 첫 번째 인자로 파일 포인터가 사용된다는 것을 제외하고는 printf() 함수와 사용법 동일

함수원형	**int fprintf(FILE *fp, char *format, ...);**	
함수인자	fp	파일 포인터 변수명
	format	형식 제어 문자열
	...	출력하고자 하는 변수리스트
반환 값	✓ 성공 → 출력한 데이터의 바이트 수 ✓ 실패 → 음수를 반환	

3) 텍스트 파일 입출력

- **fprintf() 함수 사용 예**

```
char name[10] = "Tommy";
double weight = 78.45;
int age = 25;

FILE *fp = fopen("output.dat", "w");

fprintf(fp, "%s %d %.2lf", name, age, weight);
            // output.dat 파일로 데이터를 출력

fprintf(stdout, "%s %d %.2lf", name, age, weight);
            // 모니터로 데이터를 출력...
```

4) 이진 파일 입출력

- **블록 단위의 파일 입출력 함수**
  - 이진 파일은 바이트 단위의 연속된 데이터 집합인 **블록 단위로** 데이터를 저장하고 읽음
  - 일정한 크기의 데이터를 한 번에 읽거나 쓸 수 있음
  - 이진 파일의 데이터를 읽을 때에는 **fread()** 함수를, 데이터를 저장할 때에는 **fwrite()** 함수를 사용

4) 이진 파일 입출력

- **블록 단위 파일 입력: fread()**
  - 이진파일에서 (블록 크기 x 블록 개수) 만큼의 데이터를 읽음

함수 원형	`unsigned int fread(void *buf, unsigned int size, unsigned int count, FILE *fp);`	
함수 인자	buf	읽은 데이터를 저장할 버퍼의 시작 주소
	size	읽어올 데이터 블록 하나의 크기 (바이트)
	count	읽어올 데이터 블록의 개수
	fp	파일 포인터 변수명
반환 값	✓ 성공 → 파일로부터 읽은 블록의 개수를 반환 ✓ 파일의 끝 혹은 실패 → count 보다 작은 값을 반환	

4) 이진 파일 입출력

- **fread() 함수 사용 예**

```
unsigned char age, ID[10];
FILE *fp = fopen("input.dat", "rb");

fread(&age, sizeof(unsigned char), 1, fp);
                // fp에 연결된 이진 파일로부터
                // 0~255 숫자 1개를 읽어 age 변수에 저장

fread(ID, sizeof(unsigned char), 10, fp);
                // fp에 연결된 이진 파일로부터
                // 0~255 숫자 10개를 읽어 ID 배열에 저장
```

4) 이진 파일 입출력

- **블록 단위 파일 출력: fwrite()**
  - 이진파일에 (블록 크기 x 블록 개수) 만큼의 데이터를 출력

함수 원형	**unsigned int fwrite(const void *buf, unsigned int size, unsigned int count, FILE *fp);**	
함수 인자	buf	출력할 데이터를 저장하고 있는 버퍼의 시작 주소
	size	출력할 데이터 블록 하나의 크기 (바이트)
	count	출력할 데이터 블록의 개수
	fp	파일 포인터 변수명
반환 값	✓ 성공 → 파일에 출력한 블록의 개수를 반환 ✓ 실패 → count 보다 작은 값을 반환	

4) 이진 파일 입출력

- **fwrite() 함수 사용 예**

```
unsigned char age, ID[10];
FILE *fp = fopen("output.dat", "wb");

fwrite(&age, sizeof(unsigned char), 1, fp);
                // age 변수에 저장된 0~255 숫자 1개를
                // fp에 연결된 이진 파일에 저장

fwrite(ID, sizeof(unsigned char), 10, stdout);
                // ID 배열에 저장된 0~255 숫자 10개를
                // 표준출력장치에 출력
```