CSE 206 - 파일 처리론
(File Processing)

4 장. Sequential File

File Access Pattern에 따른 File 저장 방식

- **Sequential (Access) File**
  - Application이 파일에 저장된 내용에 순차적으로 접근하는 경우가 많을 경우.
  - 파일에 저장된 내용 전체에 접근하는 경우가 많을 경우.

- Random Access File
  - Application이 파일에 저장된 임의의 내용에 접근하는 경우가 많을 경우.

Sequential file

- **정의**
  - 순차 파일은 입력되는 데이터 즉 Record들을 파일에 기록할 때, Record들의 **논리적인 순서와 같은 순서로 저장 장치에 물리적으로 기록**하는 파일
    - 파일 생성시 Record들을 연속적으로 저장하기 때문에 Record들을 접근할 때도 **저장 순서에 따라 연속적으로 접근하는 것이 효율적**
    - Record들을 조직하는 가장 기본적인 방법

앞

<사람이 관리하는 Record 순서>

학번	이름	나이	본적	성
1243	홍길동	10	서울	남
1257	김철수	20	경기	남
1332	박영희	19	충청	여
1334	이기수	21	전라	남
1367	정미영	20	경상	여
1440	최미숙	21	강원	여

뒤

앞

<디스크에 저장된 Record 순서>

학번	이름	나이	본적	성
1243	홍길동	10	서울	남
1257	김철수	20	경기	남
1332	박영희	19	충청	여
1334	이기수	21	전라	남
1367	정미영	20	경상	여
1440	최미숙	21	강원	여

뒤

Contents

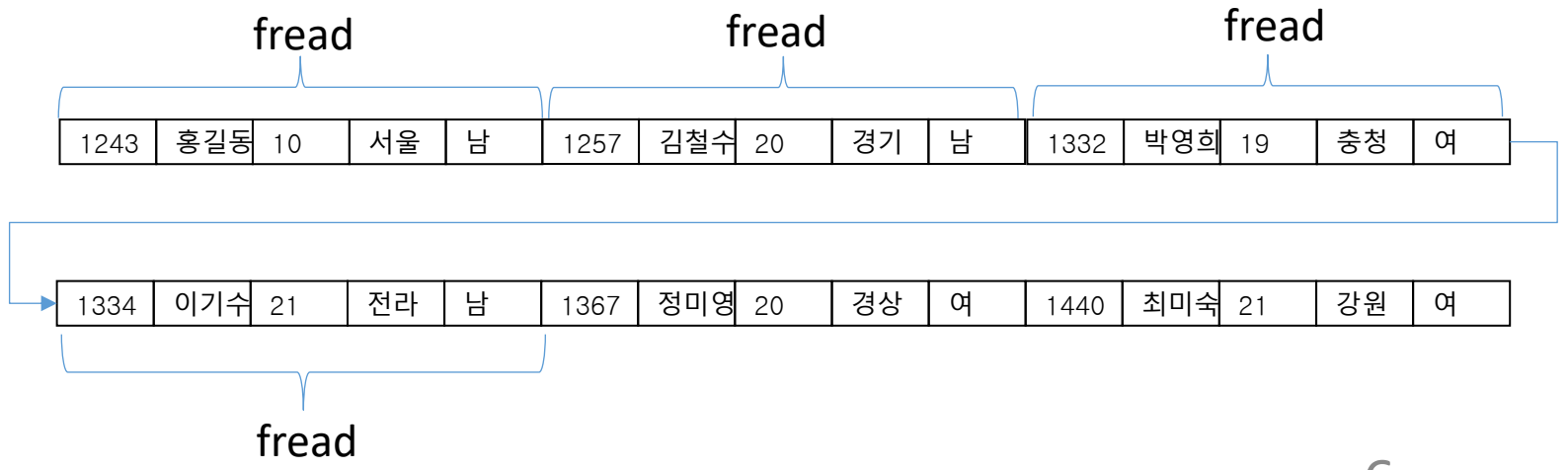
4.1. Stream File

Stream file

- **특성**
  - 데이터가 하나의 연속된 바이트 스트림(byte stream)으로 구성
  - 연속적인 읽기 연산을 통해 Record가 파일에 저장되어 있는 순서에 따라 데이터를 접근

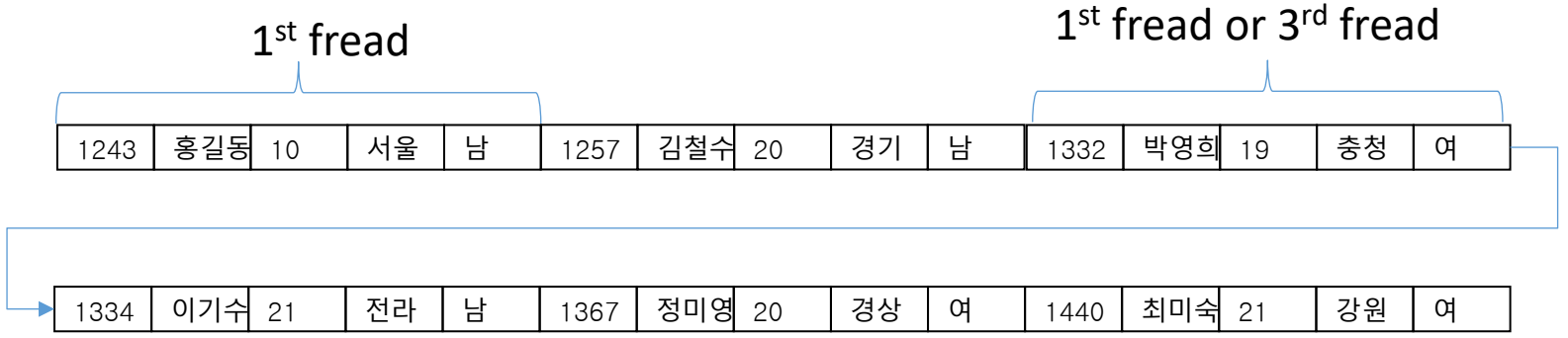
Program A

```
int main() {
  ..
  ..
  ..
   char my_buf[RECORD_SIZE];
   read_file = fopen("input_a.txt", "r");
   fread(read_file, my_buf, RECORD_SIZE);
   fread(read_file, my_buf, RECORD_SIZE);
   fread(read_file, my_buf, RECORD_SIZE);
   fread(read_file, my_buf, RECORD_SIZE);
   record_printf("%s", my_buf)
   ...
}
```



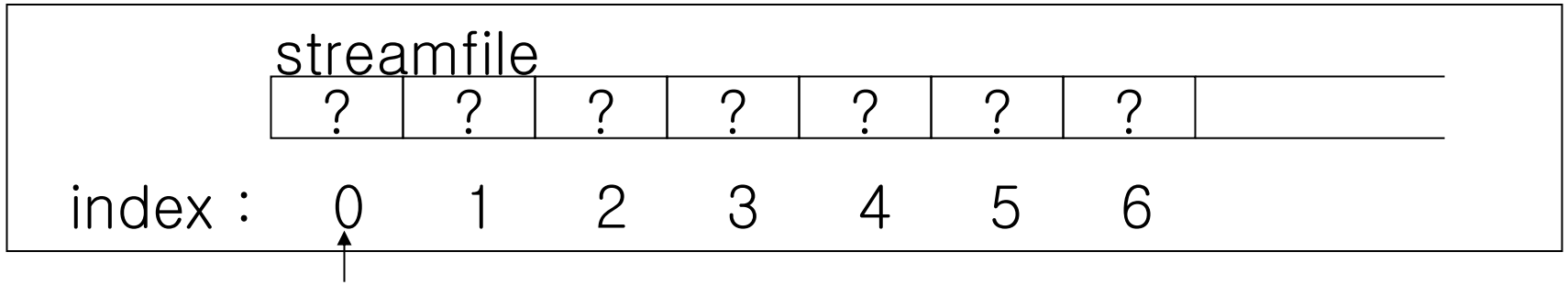
Stream file

- **종류**
  - **순차 접근 스트림 파일(sequential access stream file)**
    - 순차 접근만 허용
    - open, read, write, close 함수만 제공된다.
  - **임의 접근 스트림 파일(random access stream file)**
    - 임의 접근이 허용
    - 위에 더해 임의 위치에 접근 할 수 있는 seek 함수도 제공
- **파일 접근 모드(access mode)**
  - 파일 열기 시 수행하려는 연산을 명세: 읽기(read), 쓰기(write), 붙이기(append) 등



Stream File – Open

- **C 언어를 이용해 streamfile이라는 Stream File을 열기**
  - 함수 fopen(filename, mode)은 파일을 열거나 파일이 없으면 생성해서 파일 포인터(file pointer)를 반환
    - 파일이 열리면 파일 이름 대신 파일 포인터를 사용하여 연산할 파일을 지시함.
  - streamfp = fopen("streamfile", "w");
    - 파일이 공백인 경우에는 공백 Stream File을 생성하고 연다
    - streamfp는 File Pointer
    - Mode에는 "r"(read), "w"(write), "a"(append) 등

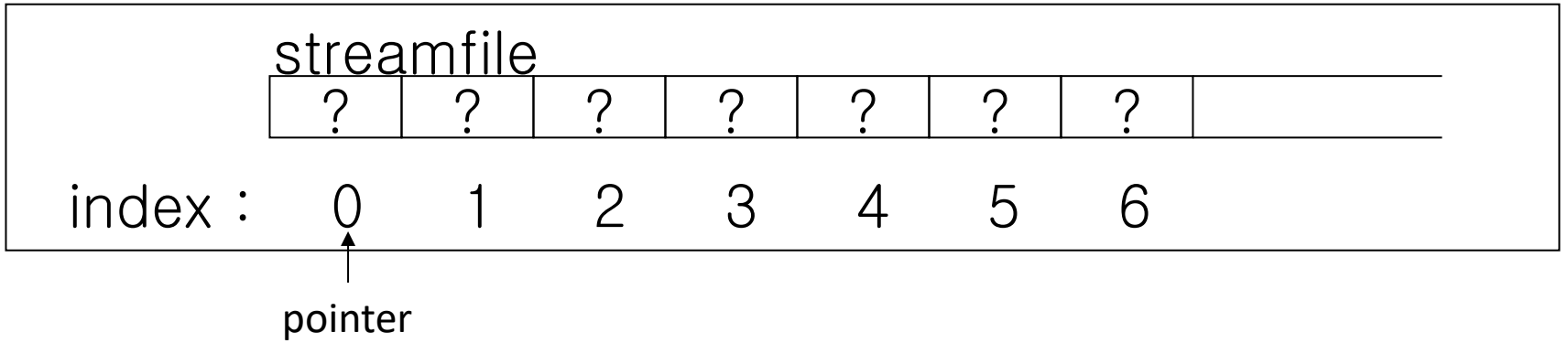


화살표는 읽기/쓰기 위치를 표현하는 인덱스

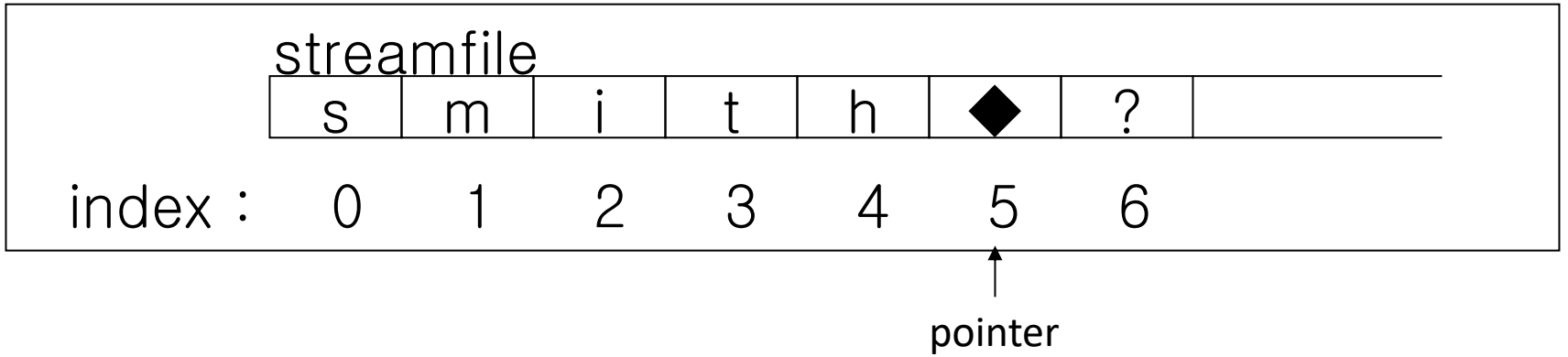
Stream File – Open

- Stream File을 read 모드로 열면 read pointer는 파일의 첫 번째 바이트를 가리킴.
- 파일을 write모드로 열면 write pointer는 파일의 첫 번째 바이트를 가리킴.
- 파일을 append 모드로 열면 write pointer는 파일의 마지막(EOF (◆)) 바이트를 가리킴.

Open with “w” or “r”



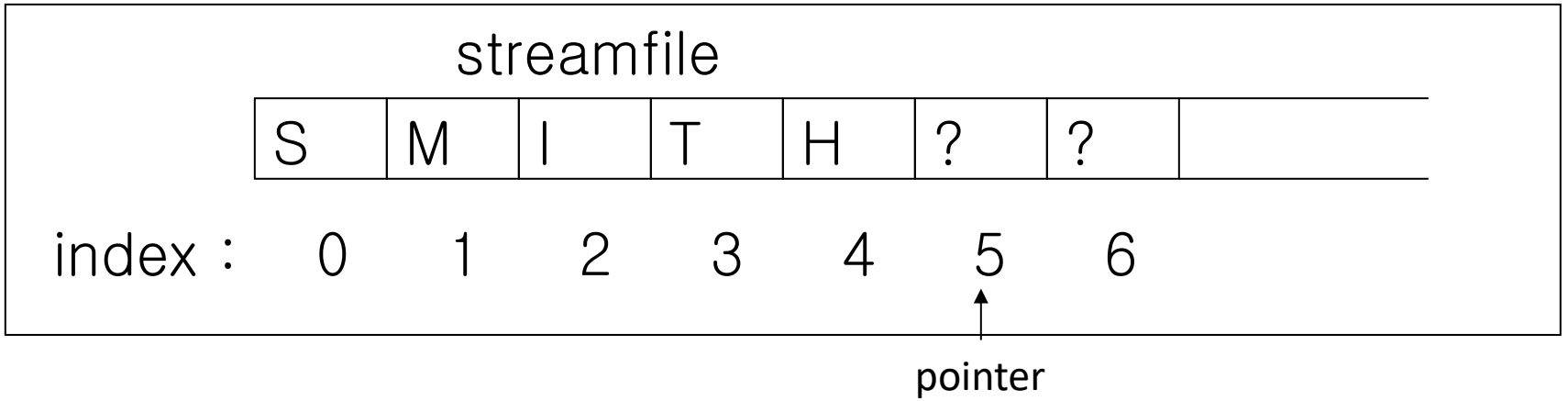
Open with “a”



Sequential Access Stream File – Write

- **Write**
  - 쓰기 포인터 위치에서 시작하여 바이트 값을 기록하고, 쓰기 포인터를 다음 바이트가 쓰여질 위치로 변경.
  - n번째 바이트를 쓰기 위해서는 반드시 (n-1)번 쓰기 연산을 수행해야 함.

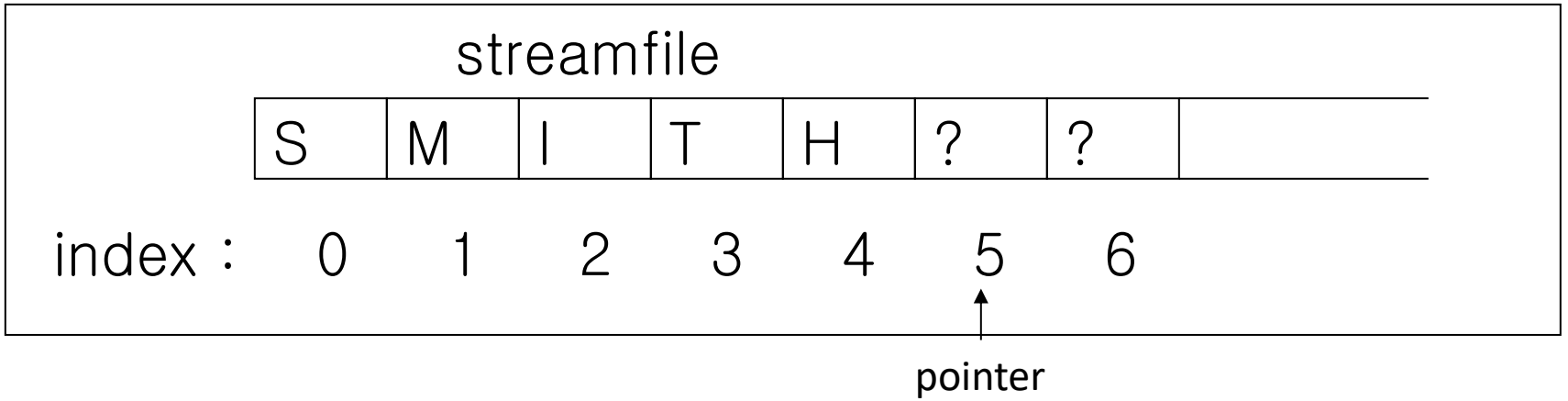
```
Int main() {
   :
  streamfp = fopen("streamfile", "w");
  fputc('s',streamfp);
  fputc('m',streamfp);
  fputc('i',streamfp);
  fputc('t',streamfp);
  fputc('h',streamfp);
   :
}
```



Sequential Access Stream File – Read

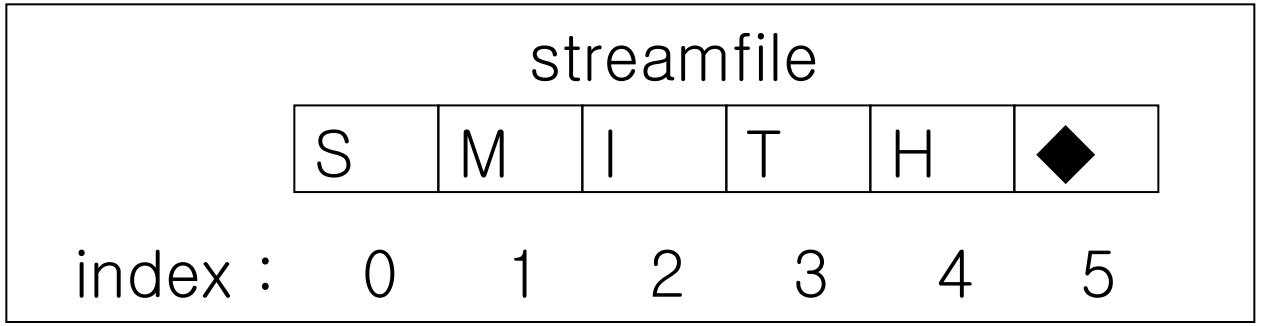
- **Read**
  - 읽기 연산
    - 읽기 포인터 위치에서 시작하여 해당 바이트를 전송하고, 읽기 포인터를 스트림 파일의 다음 바이트의 시작 위치로 변경.
    - N번째 바이트를 읽기 위해서는 반드시 (n-1)번째 바이트를 읽어야 함.

```
Int main () {
   :
  streamfp = fopen("streamfile", "r");
  read_char = fgetc(streamfp);
  read_char = fgetc(streamfp);
  read_char = fgetc(streamfp);
  read_char = fgetc(streamfp);
  read_char = fgetc(streamfp);
}
```



Stream File – Close

- fclose(streamfp) 함수 로 파일을 닫음
- fclose(streamfp);
  - 파일 포인터 streamfp가 지시하는 파일 streamfile을 닫음
  - end-of-file(◆) 표시가 파일 끝에 첨가



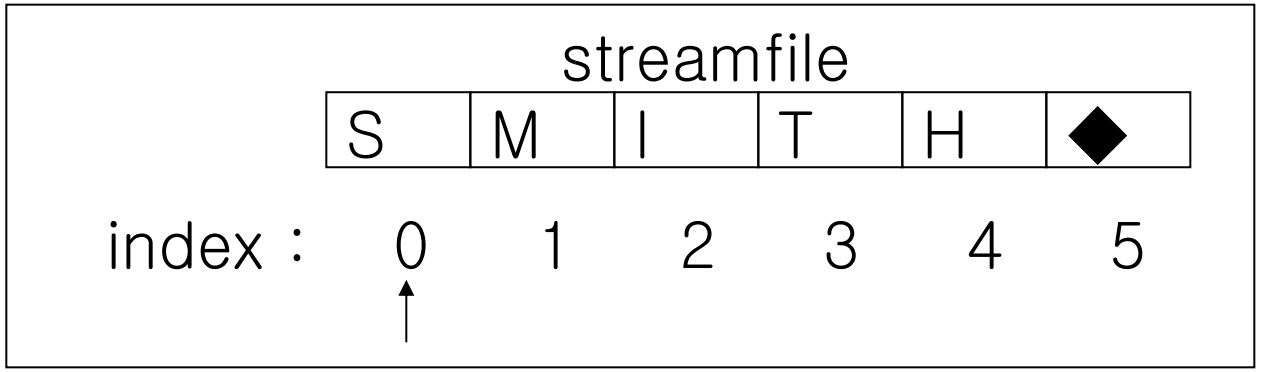
Random Access Stream File

- 파일의 시작, 끝, 현재의 읽기/쓰기 위치로부터 오프셋(offset) 값을 이용하여 이전의 바이트를 접근하지 않고 직접 접근
  - 읽기/쓰기 위치 인덱스를 이동

- Random Access를 위한 함수
  - fseek(filepointer, offset, WhereFrom)
    - 파일 스트림에서 읽기/쓰기 위치 인덱스(혹은 포인터)를 변경하는 데 사용
  - ftell(filepointer)
    - 파일에서 현재의 읽기/쓰기 위치 인덱스 (혹은 포인터)값을 알기 위해 사용

Random Access Stream File

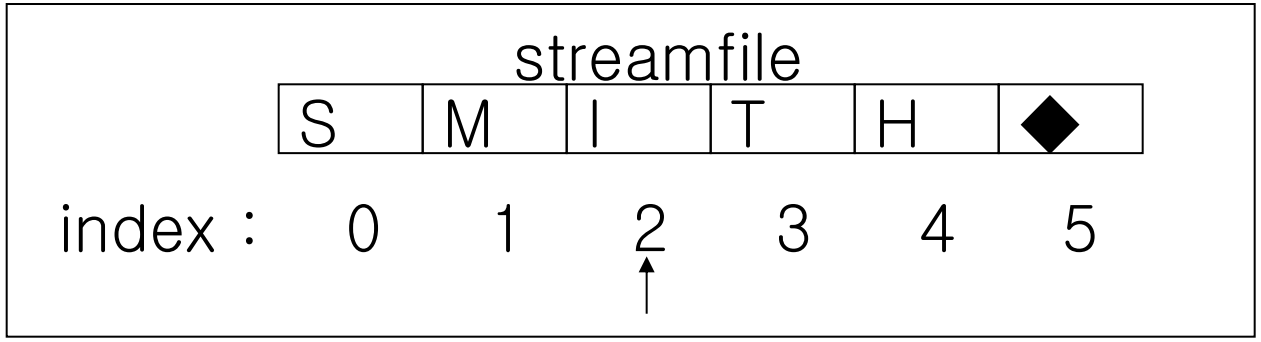
- “r”(읽기) 모드로 열린 스트림 파일
  - Streamfp = fopen(“streamfile”, “r”);
    - 파일이 열리면 읽기 위치 인덱스는 첫 번째 바이트를 가리키게 설정



  - ftell(streamfp)
    - 현재의 읽기/쓰기 인덱스 값을 반환
    - 현재 streamfile의 읽기/쓰기 인덱스 값이 0이므로, 0이 반환됨.

Random Access Stream File

- fseek(streamfp, 2, SEEK_SET);
  - 시작 위치(SEEK_SET)로부터 읽기/쓰기 인덱스를 2바이트(offset)만큼 이동시킴.
    - SEEK_SET:시작 위치
    - SEEK_END:끝 위치
    - SEEK_CUR:현재 위치

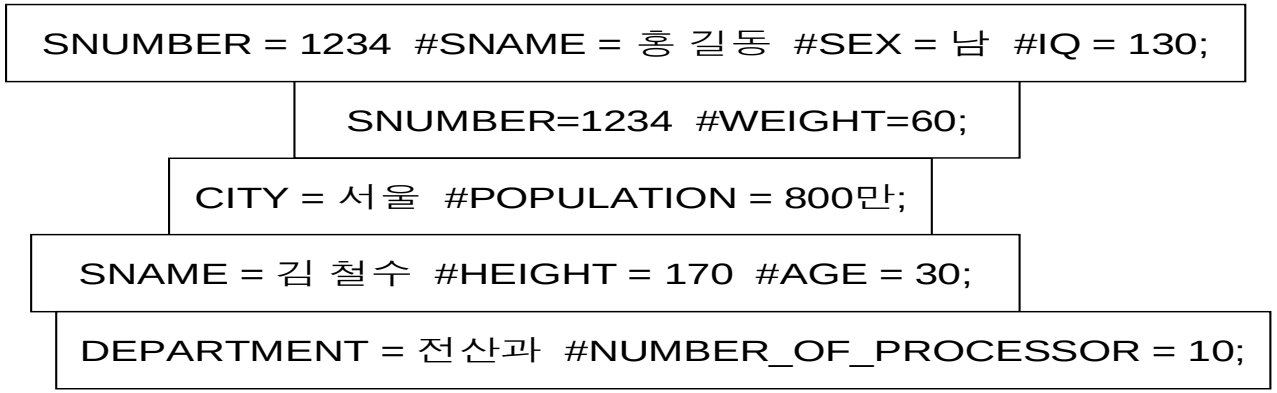


- ftell(streamfp)
  - 현재 읽기/쓰기 인덱스 값이 2이므로, 2가 반환됨.

4.2. Types of Sequential Files

Entry-sequenced file (입력 순차 파일)

- Entry-Sequenced File : 데이터(Record)가 **입력되는 순서대로 저장**된 파일
  - **Heap file, Pile file이라고도 함**
  - **Structure Data에 대한 제약 없음.**
    - Record에 대한 분석, 분류, 표준화 과정을 거치지 않아도 됨.
    - 필드의 순서, 길이 등에 제한 없음
    - Record의 길이, 타입이 일정하지 않아도 됨.
  - 1개의 Record는 여러 개의 <Field, Value> 쌍으로 구성

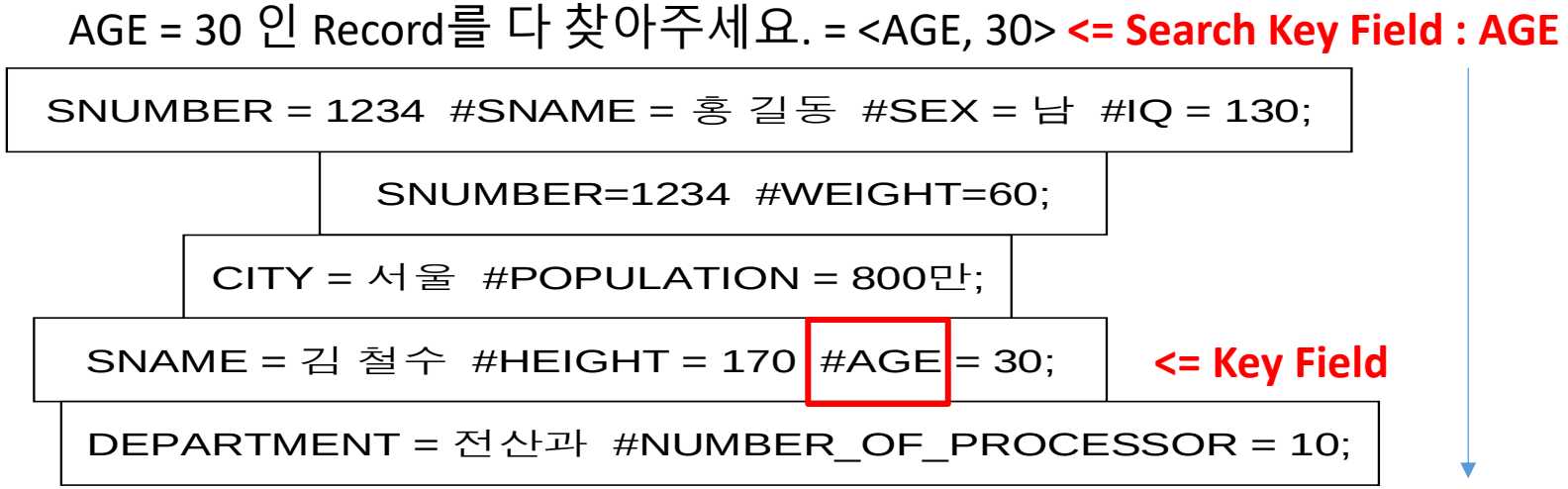


< 5 개의 Record 예 >

(그림에서 #는 <필드, 값> 쌍 사이의 분리 문자. (delimiter))

Entry-sequenced file

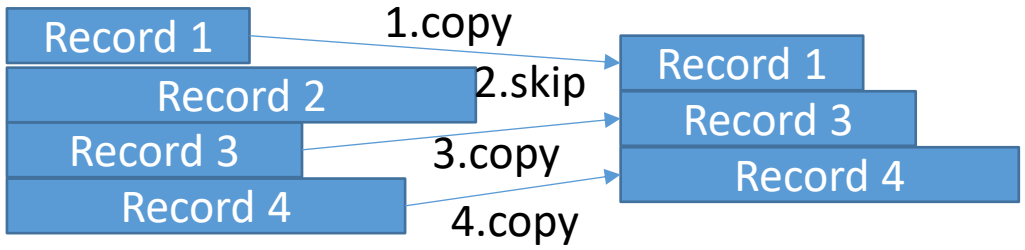
- **Insertion (삽입)**
  - 삽입되는 Record는 파일의 끝에 첨가
- **Search (검색)**
  - 주어진 search parameter의 값과 이에 대응하는 Record Filed 값을 **파일 시작부터 비교하여 Record를 선정 후,** 선정된 Record에서 원하는 필드 값을 검색
  - **Search key field** : 탐색 Parameter의 필드
  - **Key field** : Record 선정할 때 비교하는, 탐색 매개변수와 대응되는 Record의 데이터 필드



Entry-sequenced file – Record Deletion/Modification (Cont'd)

- **Deletion/Modification 작업**
  - 새로운 순차 파일을 동시에 생성하면서 수행
    - 기본적으로 원본 파일에서 Record 하나씩 읽어서 새 파일에 복사해 가면서 수행

- **Deletion (삭제)작업**
  - 삭제 대상 Record를 탐색하면서 기존의 Record를 새로운 파일로 출력
  - **삭제 대상 Record만 새로운 파일로 출력 하지 않는다.**



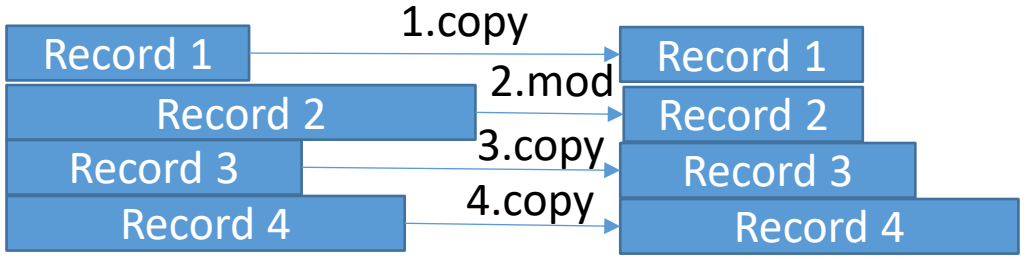
파일 원본 (Master file)

Record 2 삭제 후

- 파일 삭제/변경이 끝나 새 파일 이 만들어지면
- 파일 원본(Master file)을 지우고 새파일을 새 파일 원본 (New master file)으로 취급한다.

Entry-sequenced file – Record Deletion/Modification (Cont'd)

- **Modification (변경) 작업**
  - **변경 대상 Record를 탐색하면서 변경 대상 Record를 찾으면 해당 Record의 값을 변경하고 새로운 파일에 출력**
  - 변경 대상이 아닌 Record들은 그대로 새로운 파일에 출력



파일 원본 (Master file)　　Record 2 변경 후

- 파일 삭제/변경이 끝나 새 파일 이 만들어지면
- 파일 원본(Master file)을 지우고 새파일을 새 파일 원본 (New master file)으로 취급한다.

Entry-sequenced file – Use case

- **Entry-sequenced file 사용 예**
  - **Structure Data를 조직할 만큼, 사용 목적이 명확하지 않은 상태에서의 데이터의 임시 저장해 둘 때 사용.**
    - 데이터를 처리하기 전에 임시로 수집만 해 놓는 경우
      - 예 Log File.
    - 파일 조직을 결정하지 못한 경우
    - 파일의 용도가 결정되지 않은 경우
      - 데이터 은행(data bank)
    - 파일 조직의 변경 과정에서 중간 단계의 파일 형태로 이용

Key-sequenced file (키 순차 파일)

- **Key-sequenced file (키 순차 파일)**
  - **저장 장치에서의 Record 저장 순서와 Record의 논리적 순서가 같은** 구조의 파일
  - 파일 내에서의 Record는 **Key Field 값에 따라 정렬**
  - Structured Data로 저장됨.
    - 모든 Record는 똑같은 순서의 Data Filed로 구성
    - Data Field는 File Descriptor에 한 번만 저장하면 됨
  - **용어 :**
    - Sorted file : Record들이 특정 Key-Field 값에 따라 Sorting(정렬)된 파일
    - Sort key : Sorting에 사용된 값의 Field
    - Ascending / Descending 정렬

학번	이름	나이	본적	성
1243	홍길동	10	서울	남
1257	김철수	20	경기	남
1332	박영희	19	충청	여
1334	이기수	21	전라	남
1367	정미영	20	경상	여
1440	최미숙	21	강원	여

Key-sequenced file (키 순차 파일)

- **Key-sequenced file 의 정렬 순서**
  - 응용에 따라 결정
    - 전화번호부: 가입자 이름 순 또는 상호 순
  - 하나의 순차 파일은 **두 개의 상이한 정렬 순서를 만족시킬 수 없음**.
  - 여러 가지 상이한 정렬 순서의 파일이 필요한 경우에는 임시 파일을 만들어 사용했다가 용도가 끝나면 파일을 삭제

Sorted By Student ID

학번	이름	나이	본적	성
1243	홍길동	10	서울	남
1257	김철수	20	경기	남
1332	박영희	19	충청	여
1334	이기수	21	전라	남
1367	정미영	20	경상	여
1440	최미숙	21	강원	여

Sorted By Age

학번	이름	나이	본적	성
1243	홍길동	10	서울	남
1332	박영희	19	충청	여
1257	김철수	20	경기	남
1367	정미영	20	경상	여
1334	이기수	21	전라	남
1440	최미숙	21	강원	여

Key-sequenced file (키 순차 파일)

- **순차 파일의 특징**
  - 장점 : 데이터를 순차 접근할 경우, 다음 위치 Record의 접근이 신속
  - 데이터의 접근 형태를 고려한 뒤 그 접근 방법에 맞는 파일 구조를 선정
  - Interactive Processing (대화식 처리) 보다는 Batch Processing (일괄 처리)에 유리

Batch processing (일괄처리)

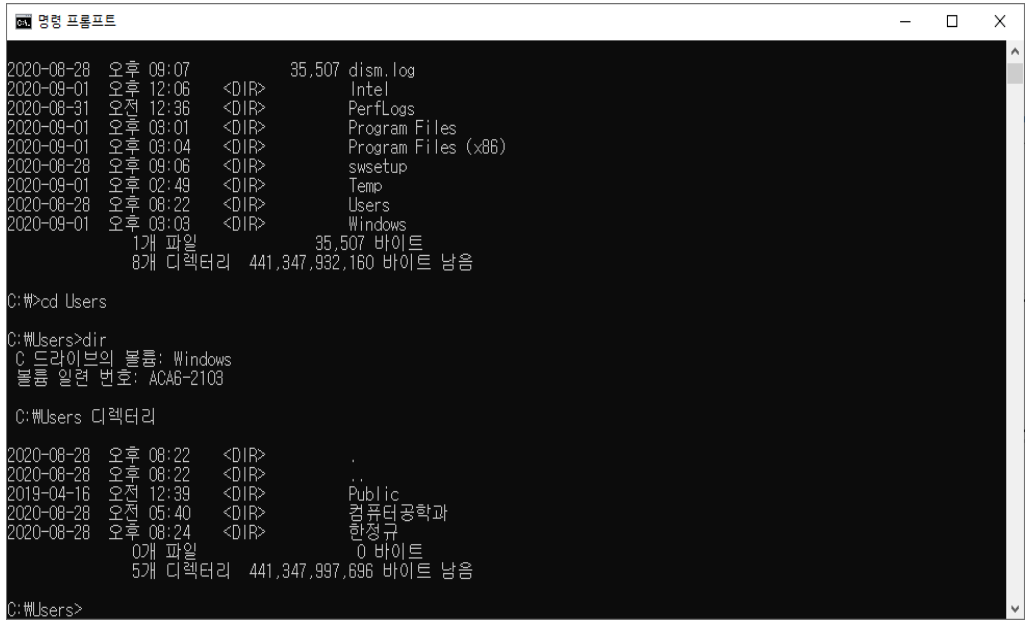
- **Batch processing (일괄처리)**
  - 미리 정의된 **일련의 처리 (Process sequence)를 한번에 모아서 처리**하는 것.
  - **처리하는 동안 유저와 상호 작용 (interaction 하지 않는다.)**
  - 일반적으로 많은 Record가 있는 환경에서 Record 각각에 대해 동일한 처리를 수행한다.
  - 특정 일, 특정 시에 주기적으로 실행되는 경우가 많다.
    - 예1) 국민 전체를 대상으로 개개인의 계좌에 재난 지원금 60만원을 지급하라.
    - 예 2) 학생들의 남녀별 평균 나이를 구하라.

학번	이름	나이	본적	성
1243	홍길동	10	서울	남
1257	김철수	20	경기	남
1332	박영희	19	충청	여
1334	이기수	21	전라	남
1367	정미영	20	경상	여
1440	최미숙	21	강원	여

남녀별 나이 평균을 위해서는
모든 Record가 필요

Interactive Processing (대화식 처리)

- **Interactive Processing (대화식 처리)**
  - 유저 요청에 의해 비교적 적은 수의 데이터를 대상으로 작업을 처리하고 결과에 대해 유저의 피드백을 받는다.
  - 유저와의 상호 작용이 핵심이다.
  - 임의의 타이밍(유저의 요청이 올 때) 에 실행된다.
- **Interactive Processing 예**



4.3. Sequential File의 설계 및 생성

Sequenced File의 설계

- 설계 시 고려 사항 : 크게 세 가지 고려사항이 있다.
  1. Record 내의 Data Filed 배치는 어떻게 할 것인가?
  2. Key Field는 어느 것으로 할 것인가?
  3. 적정 Blocking Factor (한 Block에 Record 몇 개 저장 할까)는 얼마이어야 하는가?

Sequenced File의 설계

- **설계 시 고려 사항 : 크게 세 가지 고려사항이 있다.**
  1. **Record 내의 Data Filed 배치는 어떻게 할 것인가?**
     - Active file (활동 파일)과 Inactive file (비활동 파일)로 분리하여 저장
       - **Idea : 자주 접근하는 데이터를 저장한 파일의 크기를 줄여 접근 비용을 낮추자**
     - **Fixed Length Record Vs. Variable Length Record**

Active File vs. Inactive File

- **Active File**
  - 상대적으로 자주 사용되는 데이터를 저장하여 자주 접근하는 파일의 크기 감소
- **Inactive File**
  - 상대적으로 자주 사용되지 않는 데이터를 저장.
- **Active 와 Inactive는 Record Filed의 활용 빈도 혹은 Record 활용 빈도에 따라서 구분**
  - 빈번히 처리할 파일(**Active File**)의 크기를 감소시켜 데이터 파일에 대한 처리 시간 감소

Key

활동파일에 들어갈 필드

비 활동 파일에 들어갈 필드

활동 파일에 들어갈 Record

비활동 파일에 들어갈 Record

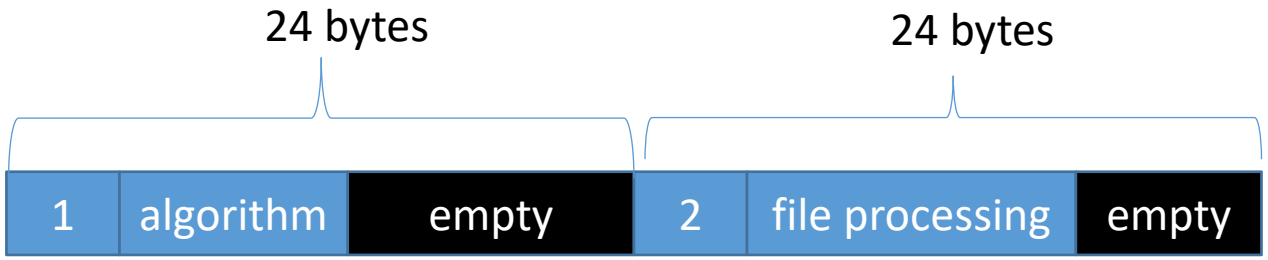
고객 번호	고객이름	고객 연락처	다음 미팅 일시	매상	VIP여부	생년월일	성별	고객 주소
1	홍길동	000-001-0001	2020/10/01	100,000,000	VIP	1970.09.13	남	서울
2	김철수	000-001-0002	2020/10/02	10,000,000	VIP	1990.11.11	남	부산
3	박영희	000-001-0003	2020/10/03	100,000,000	VIP	1980.10.21	여	대전
4	이기수	000-001-0004	2020/10/30	100,000	NO	1998.08.12	남	인천
5	정미영	000-001-0005	2021/01/01	100,000	NO	2000.01.01	여	광주
6	고길동	000-001-0006	2021/01/02	10,000	NO	1960.02.12	남	대구
7	최미숙	000-001-0007	2021/01/03	1,000	NO	1997.03.23	여	용인

Fixed Length Record Vs. Variable Length Record

- **Fixed Length Record (고정 길이 Record) :**
  - Record의 길이가 이미 정해져 있고, 모든 Record의 길이가 같음.
  - **Procs:** Data Read/Write 처리가 단순
  - **Cons:** 사용하지 않는 공간 낭비

```
struct course {
    int course_code = 1;
    char course_name[20] = “algorithm”;
};
```

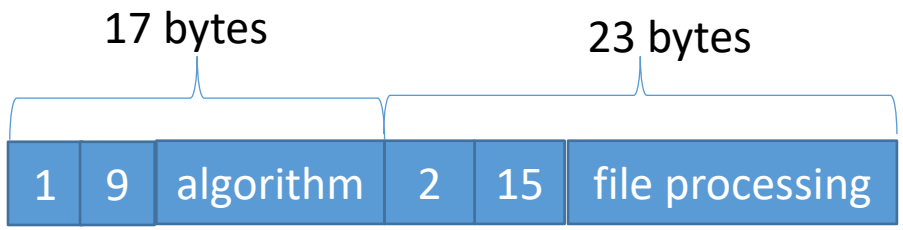
```
struct course {
    int course_code = 2;
    char course_name[20] = “file processing”;
};
```



Fixed Length Record Vs. Variable Length Record

- **Variable Length Record (가변 길이 Record):**
  - Record마다 최적의 길이를 가진다. Record 마다 길이가 다름.
  - **각** Record **길이를** 데이터와 함께 **저장**
    - Record 길이 정보의 저장 위치는 설계자가 정한다.
    - **Pros :** 저장 공간 절약.
    - **Cons:** Record 들이 동일한 Data Filed 를 가지고 있지만 각 Record 마다 값의 길이가 다름(Data Filed의 길이가 다름)을 고려한 처리를 해야 하므로 처리 비용 증가.

```
struct course {
   int course_code = 1;
   int course_name_len = 9;
   char course_name = “algorithm”;
};
struct course {
   int course_code = 2;
   int course_name_len = 15;
   char course_name = “file processing”;
};
```



Fixed Length Record Vs. Variable Length Record - Example

C 프로그램 예 : 학생-수강 정보를 위한 구조체 선언

```
typedef struct course_type          /* 과목 데이터 타입 선언 */
{   char   dept[4];
    char   course_name[20];
    char   prof_ID[7];
    int    credit;
};
typedef struct STUDENT              /* 학생 Record 구조 선언 */
{   int  st_num;
    struct name_type
    {   char  last[20];
        char  midinit;
        char  first[20];
    } name;
    struct address_type
    {   char  street[25];
        char  city[10];
        char  state[2];
        int   zip;
    } address;
    int       no_of_courses;
    course_type  course[10]; /* 한 학생이 최대 10강좌 수강 가능 */
}
```

1 과목 정보를 위해 최소 36 Bytes (그 중 사용 35 Bytes) 필요

10 과목 정보를 위해 최소 360 Bytes 필요

… 한 학기에 수업 10개를 들을까?

공간 낭비 Vs. 처리 비용 증가

순차 파일의 설계

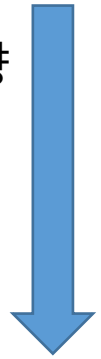
- **설계 시 고려 사항 : 크게 세 가지 고려사항이 있다.**
  - **1. Record 내의 Data Field 배치는 어떻게 할 것인가?**
  - **2. Key Field는 어느 것으로 할 것인가?**
    - 응용 요건에 따라 선정
  - **3. 적정 Blocking Factor (한 Block에 Record 몇 개 저장 할까)는 얼마이어야 하는가?**
    - 일반적으로 가능한 한 Block을 크게 하는 것이 바람직함
    - Buffer의 크기와 운영 체제가 지원하는 페이지 크기에 의해 제한 받을 수 있다.
    - Disk에서 한번의 입출력 단위 크기 (섹터 혹은 트랙. 디스크에 따라 다름)와 최대한 가깝게 한다.

4.4. Sequential File Modification

Search in sequential files (순차 파일에서의 검색)

- **Search in sequential files**
  - Query : 검색의 목적
    - **예) 10대인 사람을 찾아주세요.**
  - Query를 만족하는 Record를 Record의 저장 순서에 따라 연속적으로 검색
    - Record 처음 (File 처음)부터 순차적으로 Record 하나하나 읽어가면서 검색 조건에 맞는 Record 선택
  - 따라서, Record 검색 순서에 따라 Record Sorting 순서를 결정할 수 도 있다.

검색 방향



학번	이름	나이	본적	성
1243	홍길동	10	서울	남
1257	김철수	20	경기	남
1332	박영희	19	충청	여
1334	이기수	21	전라	남
1367	정미영	20	경상	여
1440	최미숙	21	강원	여

Search in sequential files : Query

- **Sequential File 구조상 연속적이고 Record 전체에 대한 접근이 필요한 검색의 경우에 효율적**
  - 학생의 나이 평균과 표준 편차는 얼마인가? (IHO = 1.0)
  - 4대 보험에 각각 몇 명의 사원들이 가입되어 있는가? (IHO = 1.0)

- **IHO : inquiry hit ratio (파일의 Query 적중 비율)**

$$= \frac{\text{질의 응답을 위해 접근해야 되는 Record 수}}{\text{파일 전체의 Record 수}}$$

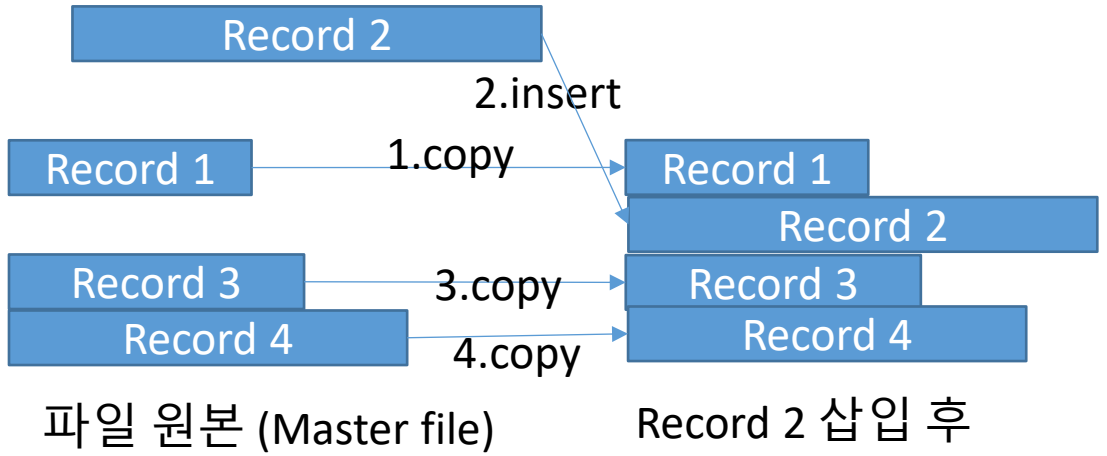
  - 질의 적중 비율이 높으면 순차 파일 구조가 적합

학생의 나이 평균과 표준 편차는 얼마인가? (IHO = 1.0)

학번	이름	나이	본적	성
1243	홍길동	10	서울	남
1257	김철수	20	경기	남
1332	박영희	19	충청	여
1334	이기수	21	전라	남
1367	정미영	20	경상	여
1440	최미숙	21	강원	여

Key Sequenced File의 Insertion (삽입)

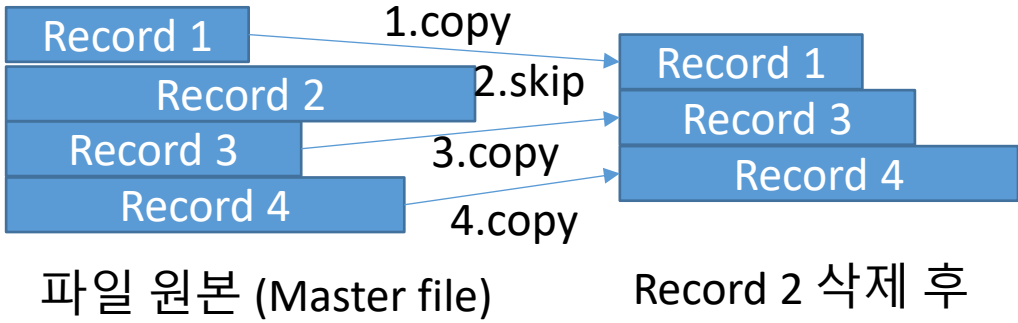
- **Key Sequenced File 에 대한 Record 삽입**
  - Key 값에 따라 오름차순/내림차순을 유지해야 하므로 복잡
    1. 두 기존 Record 사이에 삽입 위치를 검색
    2. 이 삽입 점 앞에 있는 모든 Record들은 새로운 파일로 복사
    3. 새로운 Record를 삽입
    4. 삽입 점 뒤에 있는 나머지 Record들을 새로운 파일로 복사



- 파일 삭제/변경이 끝나 새 파일 이 만들어지면
- 파일 원본(Master file)을 지우고 새파일을 새 파일 원본 (New master file)으로 취급한다.

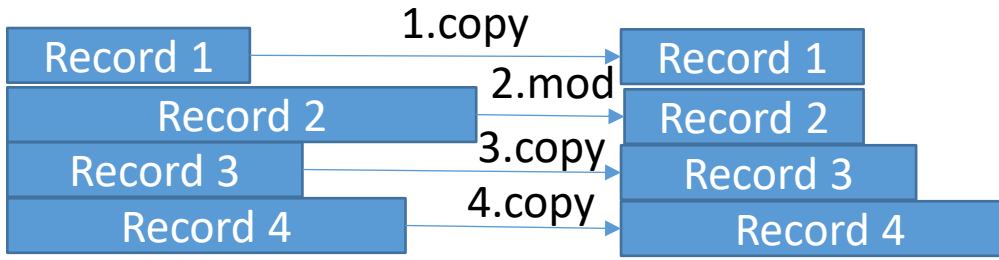
Key Sequenced File의 Deletion(삭제) & Update (수정)

- **Key Sequenced File 에서의 Record 삭제**
  - 삽입과 거의 같은 단계를 거친다.
  - 삭제할 Record를 제외하고 나머지 Record만 새로운 파일로 복사



파일 원본 (Master file)　　Record 2 삭제 후

- **Key Sequenced File 에서의 Record 수정**
  - 수정할 Record 앞의 모든 Record를 새로운 파일로 복사
  - 원하는 Record를 수정해서 새로운 파일에 삽입
  - 나머지 Record들을 새로운 파일로 복사
  - 예외) 직접 접근 저장 장치에 저장된 파일
    - 순차 파일의 임의 접근이 가능하므로, Record를 수정해서 기존 Record 위치에 수정된 Record를 기록



파일 원본 (Master file)　　Record 2 변경 후

- 파일 삭제/변경이 끝나 새 파일 이 만들어지면
- 파일 원본(Master file)을 지우고 새파일을 새 파일 원본 (New master file)으로 취급한다.

Key Sequenced File 로 되어 있는 Master File의 Update(갱신)

- **Modification Transaction을 Transaction File에 모아서 일괄 처리**
  - **Transaction File** 의 Transaction Record
    - Master File과 똑같은 Key로 정렬
    - Transaction Record는 대응하는 Master Record의 키 값과 갱신 타입을 나타내는 갱신 코드(update code)를 포함
  - Update Code
    - 새 Record의 삽입(I)
    - 기존 Record의 삭제(D)
    - 기존 Record의 갱신(U)

트랜잭션 코 드	사원 번호	성 명	주 소	부 서	전화 번호
I	12751	김 철수	당산 1동 128	경리과	

갱신을 위한 Transaction Record

Key Sequenced File 로 되어 있는 Master File의 Update

- **Insertion Transaction**
  - Record 키 값은 필수
  - 기타 데이터 필드 값은 선택적(나중에 삽입 가능)
- **Deletion Transaction**
  - 보통 해당 Master Record의 키 값 만을 명세
- **Update Transaction**
  - Record 키 값과 수정될 필드들과 해당 값만 명세
- **오류 처리**
  - 이미 저장되어 있는 Record의 삽입(키 값의 중복), 파일에 없는 Record 삭제나 수정 등의 오류
  - 수행하지 못한 모든 Transaction의 내용과 이유를 출력하는 것이 좋다

Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
2	바나나	150	50
3	감	50	150
4	배	60	140
5	복숭아	70	130
EOF			

Transaction File

Key	OP	내용
2	D	
3	U	판매개수 = 100, 재고 = 100
6	I	이름 = 포도, 판매개수 = 0, 재고 = 300
7	D	
EOF		

Master File의 Update 빈도

- **Master File의 Update** 를 결정하는 요인
  - 데이터 내용의 변경율
  - Master File의 크기
  - Master File에 대한 최신 데이터 요구
  - 파일 활동 비율

  } 얼마나 갱신될 가능성이 높은가를 측정하는 요소

- 파일 활동 비율(file activity ratio)

$$= \frac{\text{일련의 Transaction에 의해 영향을 받는 파일의 Record 수}}{\text{파일의 총 Record 수}}$$

Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
2	바나나	150	50
3	감	50	150
4	배	60	140
5	복숭아	70	130
EOF			

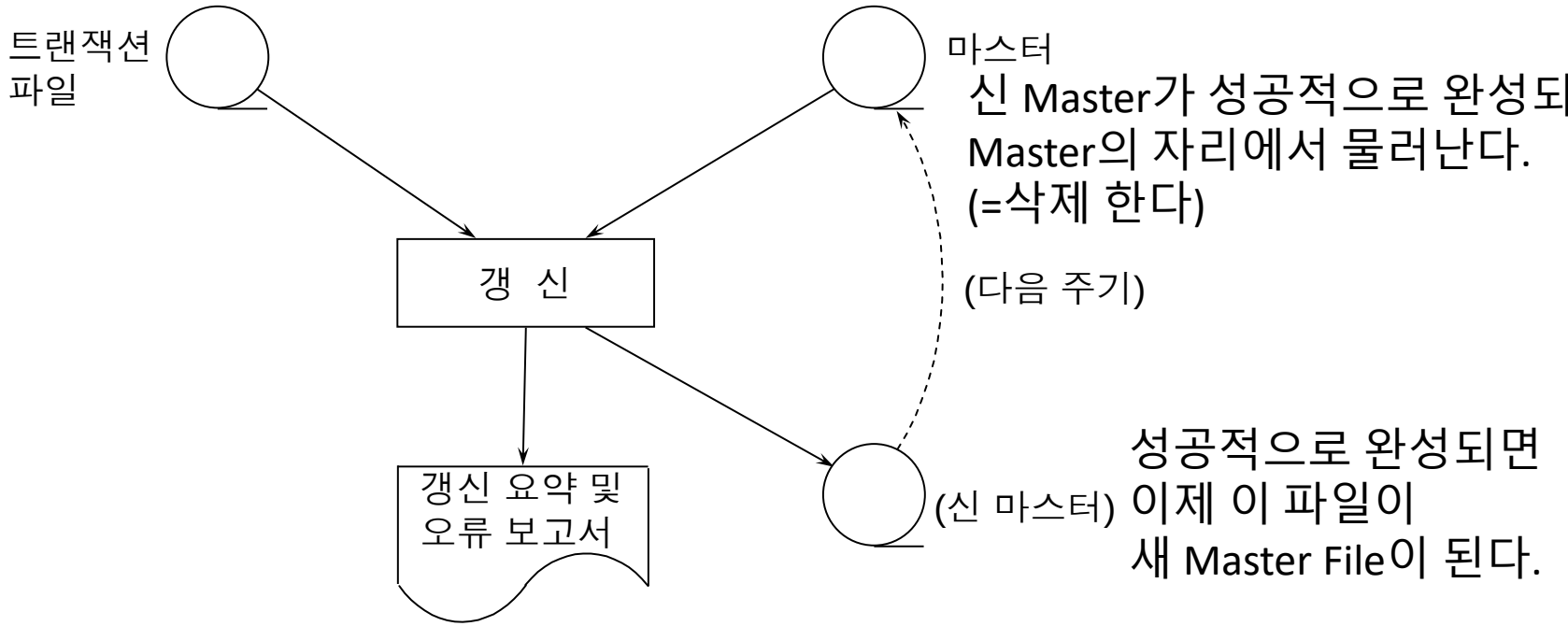
Transaction File

Key	OP	내용
2	D	
3	U	판매개수 = 100, 재고 = 100
6	I	이름 = 포도, 판매개수 = 0, 재고 = 300
7	D	
EOF		

활동 비율 = 3/5

Key Sequenced File 로 되어 있는 Master File의 Update 작업

- Master File / 신Master File
  - 파일 활동 비율이 낮을수록 신Master File로 단순히 복사하는 Record 수가 증가
  - 갱신 작업 종료 시 실행 과정에서 일어난 여러 가지 오류와 갱신 요약을 보고서 (혹은 로그 파일)로 생성



Key Sequenced Master File의 변경(Modification) 알고리즘

- **가정**
  - **Transaction 파일과 순차 Master File은 똑같은 Key로 오름차순**으로 정렬
  - 각 파일의 **Record Key 값은 유일(중복되지 않음)**
- **TransKey와 masterKey의 비교**
  - 갱신 알고리즘의 핵심은 transKey와 masterKey의 비교작업
  - transKey : Transaction 파일에서 현재 보고 있는 Record 키
  - masterKey : Master File에서 현재 보고 있는 Record 키
  - 가정 : 파일의 끝을 표시하는 EOF는 어떤 Record 키 값보다 크다.

Key Sequenced Master File의 변경(Modification) 알고리즘

transKey : Transaction File의 현재 보고 있는 Record Key
masterKey : Master File의 현재 Record Key

1. masterKey < transKey
   - Master Record에 적용할 Transaction Record가 없는 경우
     - Master Record를 새로운 Master File로 복사만 하고, 다음 Master Record를 읽어 온다.

2. masterKey = transKey
   - Transaction Record의 갱신코드에 따라 적절한 연산을 수행
     - **수정(U)인 경우 :** Record를 변경해서 새로운 Master File에 삽입하고, 다음 Transaction Record를 읽어 온다.
     - **삭제(D)인 경우 :** Master Record는 삭제, 즉 무시된다.
     - **삽입(I)인 경우 :** Master File에 이미 같은 Key 값을 가진 Record가 있으므로 중복 Record라는 오류 메시지를 출력하고 다음 Transaction Record를 읽어 온다.

키 순차 Master File의 갱신 알고리즘

3. masterKey > transKey
   - Transaction Record Key와 일치하는 Master Record가 없는 경우
     - **갱신 코드가 삽입(I)인 경우:** Record를 구성해서 새로운 Master File에 삽입하고 다음 Transaction Record를 처리
     - **그 이외 갱신 코드가 수정(C)이거나 삭제(D)인 경우:** 적절한 오류 메시지를 출력하고, 다음 Transaction Record를 처리

- **하나의 Master Record에 적용할 Transaction이 다수인 경우**
  - Transaction Record들을 발생 시간 순서에 따라 적용
    - Transaction Record들을 transKey (1차 정렬 기준), Transaction 발생 시간(2차 정렬 기준) 으로 정렬한 뒤에 갱신 작업을 시작
    - 갱신된 Record를 새로운 Master File에 출력하기 전에 관련 Transaction들이 모두 처리되었는지 확인

Key Sequenced Master File의 Update 알고리즘 실행 예 - 1/4

Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
2	바나나	150	50
3	감	50	150
4	배	60	140
5	복숭아	70	130
EOF			

Transaction File

Key	OP	내용
2	D	
3	U	판매개수 = 100, 재고 = 100
6	I	이름 = 포도, 판매개수 = 0, 재고 = 300
7	D	
EOF		

New Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100

Step 1: masterKey < transKey

=> Just copy master record; masterKey = the next master record;

Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
2	바나나	150	50
3	감	50	150
4	배	60	140
5	복숭아	70	130
EOF			

Transaction File

Key	OP	내용
2	D	
3	U	판매개수 = 100, 재고 = 100
6	I	이름 = 포도, 판매개수 = 0, 재고 = 300
7	D	
EOF		

New Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100

Step 2: masterKey == transKey && OP == D

=> Do not copy; masterKey = the next master record; transKey = the next transaction record;

Key Sequenced Master File의 Update 알고리즘 실행 예 - 2/4

Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
2	바나나	150	50
3	감	50	150
4	배	60	140
5	복숭아	70	130
EOF			

Transaction File

Key	OP	내용
2	D	
3	U	판매개수 = 100, 재고 = 100
6	I	이름 = 포도, 판매개수 = 0, 재고 = 300
7	D	
EOF		

New Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
3	감	100	100

Step 3: masterKey == transKey && OP == U

=> Modify 판매개수 = 100, 재고 = 100 and copy; masterKey = the next master record; transKey = the next transaction record;

Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
2	바나나	150	50
3	감	50	150
4	배	60	140
5	복숭아	70	130
EOF			

Transaction File

Key	OP	내용
2	D	
3	U	판매개수 = 100, 재고 = 100
6	I	이름 = 포도, 판매개수 = 0, 재고 = 300
7	D	
EOF		

New Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
3	감	100	100
4	배	60	140

Step 4: masterKey < transKey

=> Just copy master record; masterKey = the next master record;

Key Sequenced Master File의 Update 알고리즘 실행 예 - 3/4

Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
2	바나나	150	50
3	감	50	150
4	배	60	140
5	복숭아	70	130
EOF			

Transaction File

Key	OP	내용
2	D	
3	U	판매개수 = 100, 재고 = 100
6	I	이름 = 포도, 판매개수 = 0, 재고 = 300
7	D	
EOF		

New Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
3	감	100	100
4	배	60	140
5	복숭아	70	130

Step 5: masterKey < transKey

=> Just copy master record; masterKey = the next master record;

Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
2	바나나	150	50
3	감	50	150
4	배	60	140
5	복숭아	70	130
EOF			

Transaction File

Key	OP	내용
2	D	
3	U	판매개수 = 100, 재고 = 100
6	I	이름 = 포도, 판매개수 = 0, 재고 = 300
7	D	
EOF		

New Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
3	감	100	100
4	배	60	140
5	복숭아	70	130
6	포도	0	300

Step 6: masterKey > transKey && OP == I

=> Copy trans record; transKey = the next transaction record;

Key Sequenced Master File의 Update 알고리즘 실행 예 - 4/4

Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
2	바나나	150	50
3	감	50	150
4	배	60	140
5	복숭아	70	130
EOF			

Transaction File

Key	OP	내용
2	D	
3	U	판매개수 = 100, 재고 = 100
6	I	이름 = 포도, 판매개수 = 0, 재고 = 300
7	D	
EOF		

New Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
3	감	100	100
4	배	60	140
5	복숭아	70	130
6	포도	0	300

Step 7: masterKey > transKey && OP == D
=> Error : non-existing key deletion; transKey = the next transaction record;

Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
2	바나나	150	50
3	감	50	150
4	배	60	140
5	복숭아	70	130
EOF			

Transaction File

Key	OP	내용
2	D	
3	U	판매개수 = 100, 재고 = 100
6	I	이름 = 포도, 판매개수 = 0, 재고 = 300
7	D	
EOF		

New Master File

상품번호 (Key)	이름	판매개수	재고
1	사과	100	100
3	감	100	100
4	배	60	140
5	복숭아	70	130
6	포도	0	300
EOF			

Step 8: masterKey == EOF && transKey == EOF
=> Finished. Append EOF;