4. 순차 화일

❖ 순차 화일(sequential file)

- 정의
  - 레코드들을 조직하는 가장 기본적인 방법
  - 화일 생성시 레코드들을 연속적으로 저장하므로, 레코드들을 접근할 때도 저장할 때의 순서대로 연속적으로 접근해야 한다.

- 종류
  - 입력(수록) 순차 화일(entry-sequenced file)
    - 레코드가 입력되는 순서대로 저장, heap file, pile file
  - 키 순차 화일(key-sequenced file)
    - 레코드의 특정 필드 값 순서에 따라 저장

❖ 스트림 화일(stream file)

- 정의
  - 연속적인 판독 연산을 통해 레코드가 화일에 저장되어 있는 순서에 따라 데이타를 접근하는 화일
  - 데이타가 하나의 연속된 바이트 스트림으로 구성

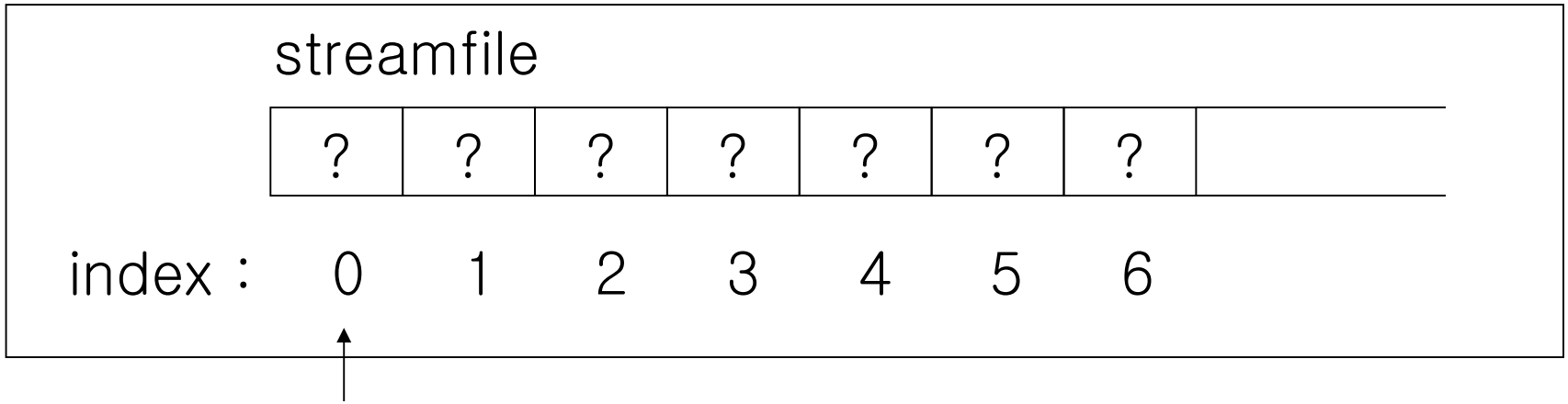
- 종류
  - 순차 접근 스트림 화일(sequential access stream file)
    - 순차 접근만을 허용
  - 임의 접근 스트림 화일(random access stream file)
    - 임의 접근이 허용됨

- **접근 모드(access mode)**
  - 화일에서 수행하려는 연산에 따라 판독(read), 기록(write), 갱신(read/write), 첨가(append) 등을 명세

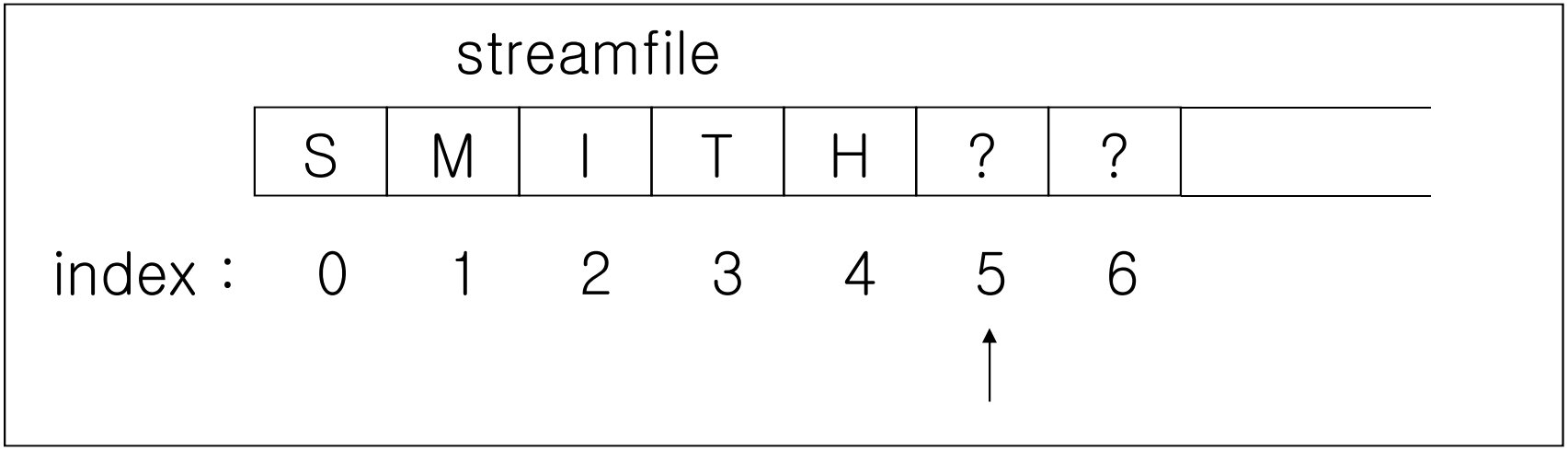
▶ 순차 접근 스트림 화일

- **판독(read)**
  - 기본 스트림 화일을 판독(read) 모드로 열면 판독 포인터는 화일의 첫 번째 바이트를 가리킨다.
  - 판독 연산
    - 해당 위치에서 시작하여 해당 바이트 값을 전송하고, 판독 포인터를 스트림 화일의 다음 바이트 시작 위치로 변경한다.
    - n번째 바이트 값을 판독하기 위해서는 반드시 (n-1)번째 바이트 값을 판독해야 한다.
- **기록(write)**
  - 화일을 기록(write) 모드로 열면 기록 포인터는 화일의 첫 번째 바이트가 기록될 위치를 가리킨다.
  - 기록 연산
    - 해당 위치에서 시작하여 해당 바이트 값을 기록하고, 기록 포인터를 다음 바이트가 기록될 위치로 변경한다.
    - n번째 바이트 값을 기록하기 위해서는 반드시 (n-1)번 기록 연산을 수행해야 한다.

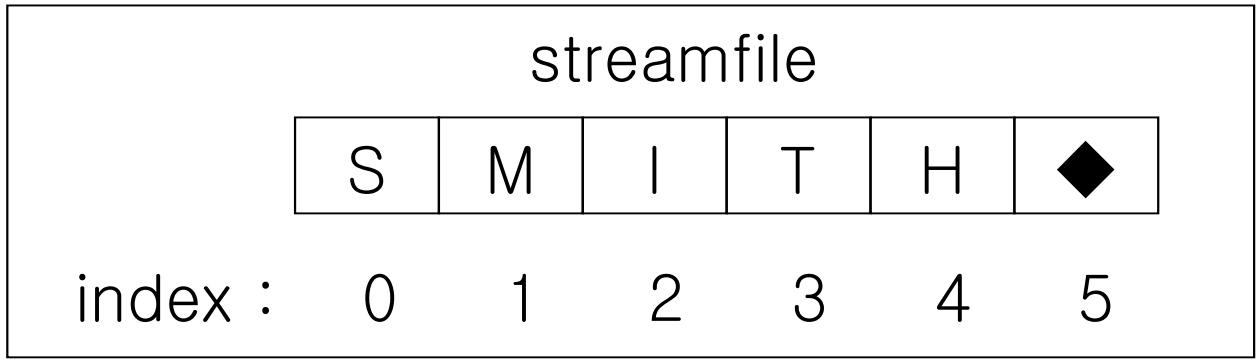
- **C를 이용한 스트림 화일 streamfile 생성**
  - streamfile = fopen(“stream.txt”, “w”);
    - 공백 스트림 화일이 생성되어 개방된다.
    - 화살표는 인덱스 값을 갖는 포인터를 나타낸다.



- fputc(ch,streamfile);
  - 스트림 화일에 한 글자를 쓴다.
  - fputc()를 연속적으로 사용하여 S, M, I, T, H 값을 입력한다.
  - 입력이 끝나면 다음과 같은 스트림 화일이 생성된다.



- fclose(streamfile);
  - 스트림 화일 streamfile을 닫는다.
  - ◆로 표현된 end-of-file 표시가 스트림 화일 끝에 첨가된다.
  - 이 스트림 화일은 stream.txt라는 화일 이름으로 저장된다.



◆ 순차 접근 스트림 화일

- 연속적으로 화일을 접근하고, 화일에 있는 모든 바이트를 처리하는 경우에 유용
- 화일을 순차적으로 접근하는 과정은 배열을 순차적으로 접근하는 것과 유사
- 특정 바이트를 찾기 위한 방법으로는 좋지 않다.

◆ 임의 접근 스트림 화일

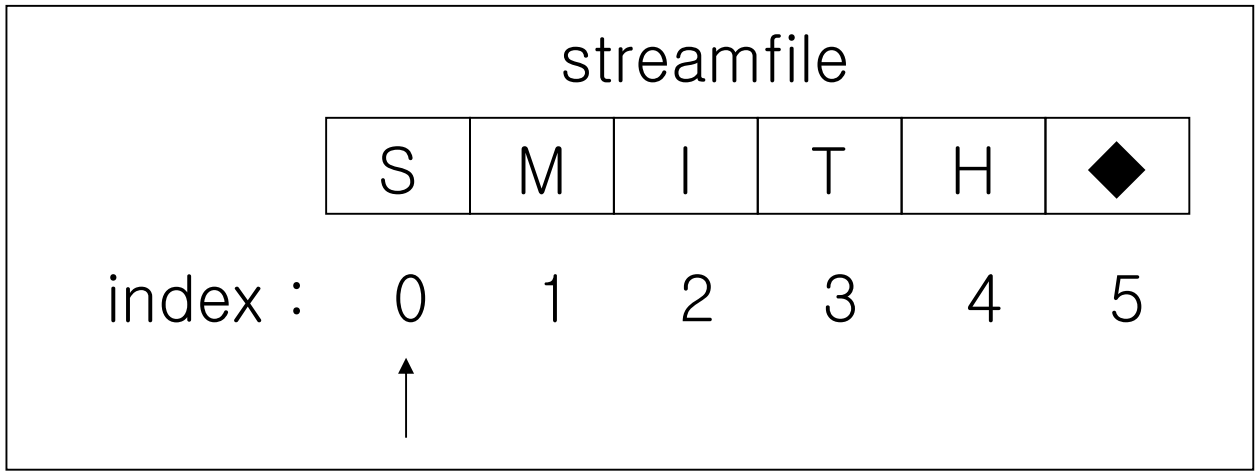
- 이원 탐색법
  - 배열의 인덱스를 이용하여 배열의 원소를 직접(임의로) 접근
- 이원 탐색법을 화일에 적용
  - 반드시 화일에 있는 바이트를 임의로 접근할 수 있어야 한다.
  - 임의 접근 스트림 화일은 가능

▶ 임의 접근 스트림 화일

- 오프셋(offset) 값을 이용
  - 이전의 바이트를 접근하지 않고 직접 접근

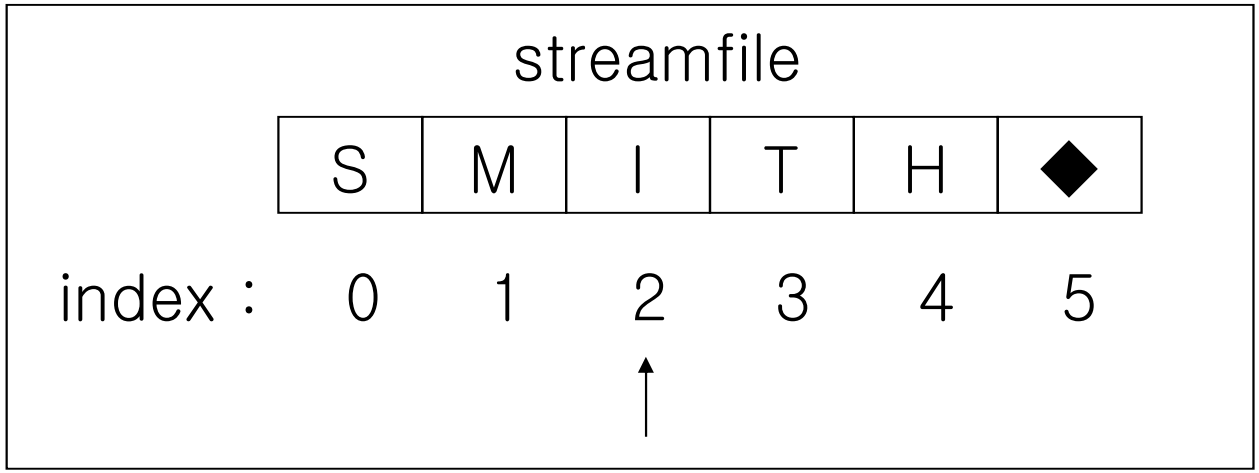
- 임의 접근을 위한 함수
  - fseek(FileName, Offset,WhereFrom) 함수
    - 화일 스트림에서 판독 또는 기록 포인터를 변경하는 데 사용
    - 화일의 시작, 끝, 현재의 위치로부터 Offset 크기만큼 판독 또는 기록 포인터를 이동시킨다.
  - ftell(FileName) 함수
    - 화일 스트림에서 판독 또는 기록 포인터의 인덱스 값을 반환하는 데 사용

- **"r"(판독) 모드로 개방한 스트림 화일**
  - streamfile = fopen("stream.txt", "r");
    - 화일이 열리면 판독 포인터는 첫 번째 바이트를 가리키게 설정



  - ftell(streamfile)
    - 현재의 포인터 값을 반환해 주는 함수
    - 현재 판독 포인터의 인덱스 값이 0이므로, 0이 반환된다.

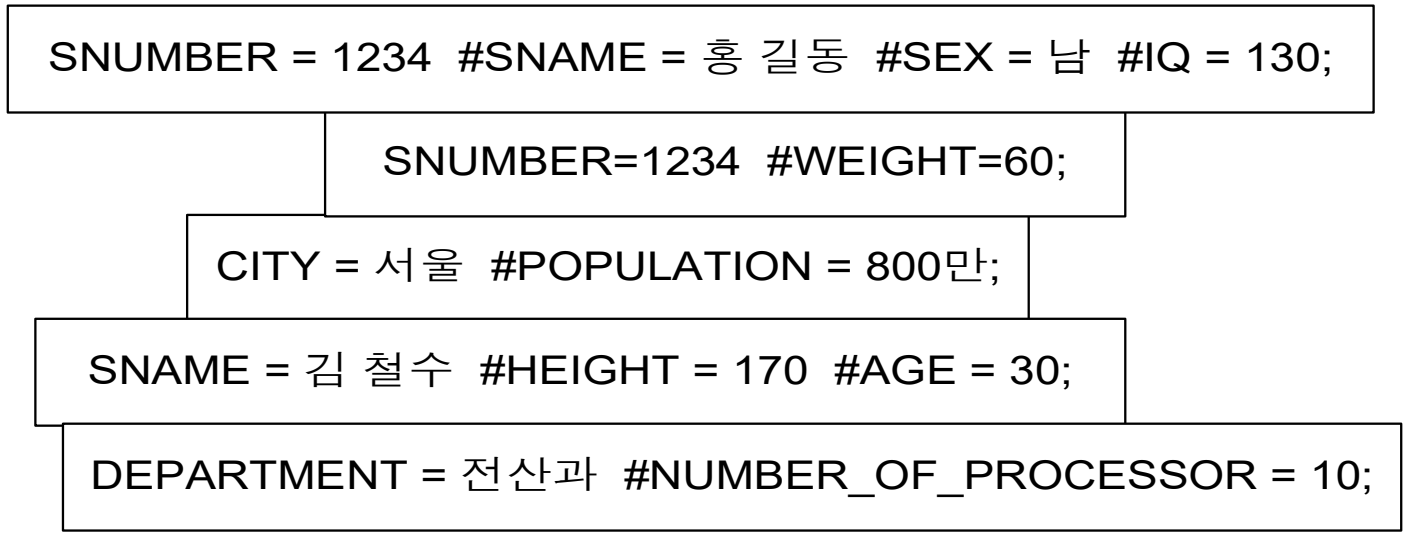
- fseek(streamfile, 2, SEEK_SET);
  - 시작 위치로부터 판독 포인터를 2바이트 이동시킨다.
  - SEEK_SET:시작 위치/SEEK_END:끝 위치/SEEK_CUR:현재 위치



- ftell(streamfile)
  - 현재 판독 포인터의 인덱스 값이 2이므로, 2가 반환된다.

❖ 순차 화일의 유형

- **입력 순차 화일(entry-sequenced file)**
  - heap file, pile file이라고도 함
  - 특징
    - 레코드에 대한 분석, 분류, 표준화 과정을 거치지 않음
    - 필드의 순서, 길이 등에 대해서도 제한 없음
    - 레코드의 길이, 타입도 일정하지 않을 수 있음
    - 오직 입력 순서만 존재



- 성질
  - 저장 장치 내의 레코드 순서 = 레코드 리스트의 순서
  - 레코드는 애트리뷰트 이름-값 쌍으로 구성
  - 화일 기술자(file descriptor)가 필요없다.

- 갱신 작업
  - 새로운 레코드 삽입, 기존 레코드 삭제, 기존 레코드 변경 등
  - 레코드 삽입 : 기존 화일 끝에 첨가

- 검색 작업
  - 주어진 필드 값인 탐색매개변수와 이에 대응하는 화일 레코드 필드 값을 화일 시작부터 비교하여 레코드를 선정하고, 선정된 레코드에서 원하는 필드 값을 검색
  - 키 필드(key field) : 레코드 선정 시 탐색매개변수와 대응되는 필드
  - 탐색 키 필드(search key field) : 탐색매개변수의 필드

- 삽입, 삭제, 변경 작업
  - 새로운 순차 화일을 생성하면서 동시에 수행
  - 작업 대상 레코드를 검색하면서 기존의 레코드를 새로운 화일로 출력 → 해당 레코드 검색 시 해당 작업 수행 → 나머지 남은 레코드들을 다시 새로운 화일로 모두 출력

- 사용 예
  - 데이타를 처리하기 전에 임시로 수집만 하는 경우
  - 적당한 화일 조직을 결정하지 못한 경우
    - 데이타 은행(data bank)에서 많이 이용
  - 레코드 구조가 결정되기 이전의 텍스트 형태의 데이타 저장
  - 화일 조직을 변경해야 할 때 중간 과정으로 이용

◆ 키 순차 화일(key-sequenced file)

- 특징
  - 레코드는 특정 필드, 즉 정렬 키(sort key) 값에 따라 정렬
    - 오름차순(ascending), 내림차순(descending)
    - 정렬 키는 필드의 집합이 될 수 있음.
  - 데이터의 구조는 테이블의 형태와 비슷

- 성질
  - 저장 장치 내의 레코드 순서 = 레코드 리스트의 순서
  - 모든 필드들이 분류되어, 각 레코드는 똑같은 순서의 필드값을 가지고 있다.
  - 화일 기술자(file descriptor)에 데이터 필드 이름이 정의되어 있다.

- 검색
  - 정렬 키의 순서대로 레코드 접근
  - 차위 레코드를 신속하게 접근할 수 있음
  - 두 가지 정렬이 필요하면, 두 개의 순차 파일을 구성해야 함
  - 일괄 처리(batch)에 사용

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

File descriptor

File

– 오름차순 정렬

if (레코드 i의 키 값 ≤ 레코드 j의 키 값)
then 레코드 i는 레코드 j 앞에 위치;

◆ 순차 화일의 정렬 순서

- 응용에 따라 결정 예) 전화번호부 – 가입자 이름
- 하나의 순차 화일이 두 개의 상이한 정렬 순서를 만족시킬 수 없다.
- 여러 정렬 순서 화일이 필요한 경우에는 임시 화일을 생성했다가 용도가 끝나면 화일을 삭제

◆ 순차 화일의 특징

- 일괄 처리(batch processing)에서 많이 사용
- 장점 : 차위 레코드를 신속하게 접근할 수 있다.
- 데이타의 접근 요구를 고려한 후 그 접근 방법에 맞게 화일을 구성해야 한다.

❖ 순차 화일의 설계

◆ 설계 시 고려 사항

1. 레코드 내의 필드 배치는 어떻게 할 것인가?

- ◆ 활동 화일(active file) / 비활동 화일(inactive file)을 구분하여 저장
  - 활동 화일의 크기를 감소시켜 데이타 화일에 대한 처리 시간 감소
  - 필드 타입이 아닌 레코드 어커런스 활용에 따라 구분하기도 한다.

- ◆ 고정 길이(fixed length) / 가변 길이(variable length)
  - 고정 길이 레코드 사용 : 사용하지 않는 공간 낭비
  - 가변 길이 레코드 사용
    - 각 레코드 길이를 나타내기 위한 적절한 제어 정보와 함께 저장
    - 각 레코드 앞부분에 있는 시스템용 필드에 제어 정보를 저장

◆ C 프로그램 예

```
typedef struct course_type        /* 과목 데이타 타입 선언 */
{     char    dept[4];
      char    course_name[20];
      char    prof_ID[7];
      int      credit;
};
typedef struct STUDENT            /* 학생 레코드 구조 선언 */
{     int  st_num;
      struct name_type
      {     char   last[20];
            char   midinitial;
            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 */
}
```

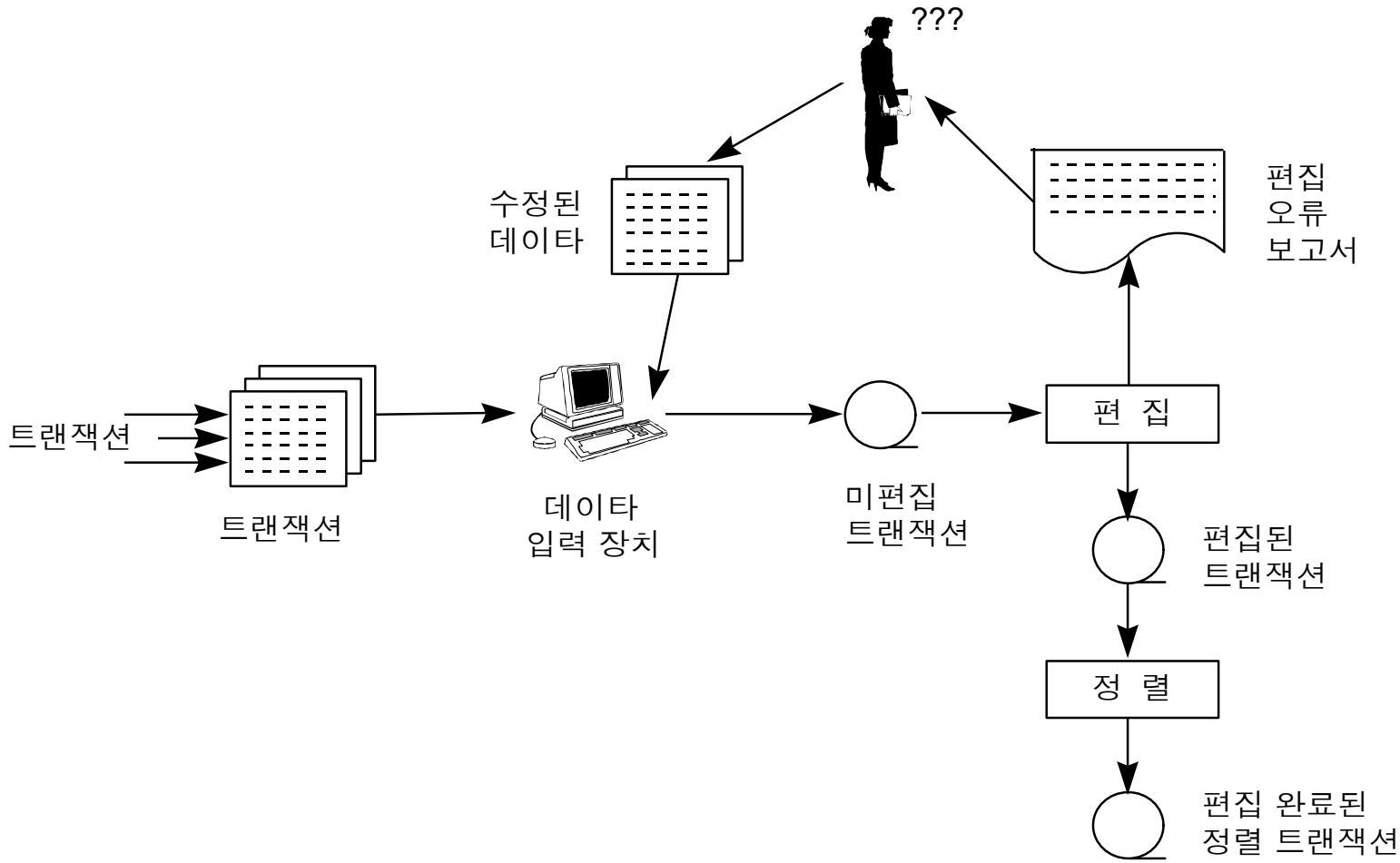
2. 키 필드는 어느 것으로 할 것인가?
   - 응용 요건에 따라 선정
   - 순차 화일에서 이 키는 레코드 접근 순서를 결정
   - ex., 트랜잭션 화일 : 마스터 화일과 같은 키
   - ex., 보고서 화일 : 출력 형태(순서)에 따라 결정

3. 적정 블럭킹 인수는 얼마이어야 하는가?
   - 일반적으로 가능한 한 블록을 크게 하는 것이 바람직
   - 메인 메모리 내 버퍼 공간, 운영 체제가 지원하는 블록 크기에 의해 제한받을 수 있다.
   - 순차 화일의 디스크 저장
     - 섹터 주소 기법 사용 : 블록 크기를 가능한 한 섹터 크기에 가깝게
     - 실린더 주소 기법 사용 : 블록 크기를 트랙 크기에 가깝게

❖ 순차 화일의 생성

- 생성
  - 생성 : 데이타 저장 장치에 레코드들을 순서대로 입력
  - 트랜잭션 화일(transaction file) 생성
    - 키 순차 화일의 갱신 : 트랜잭션 화일(transaction file) 이용
    - 데이타 수집, 레코드 형식으로 변환, 데이타 레코드 편집

- 편집
  - 트랜잭션 화일 생성 과정에서 입력되는 데이타 값에 오류가 있는지 검사하는 과정
  - 점검 내용
    - 입력된 값이 올바른 범위 안에 있는가
    - 필요한 필드 값 존재 여부, 필드 값 타입 적절성, 필드 값 유효성, 관련 필드 값 유무
    - 합계의 일치, 계수
    - 숫자 타입 필드의 앞자리 공백은 0으로 채움, 영숫자나 텍스트를 적절한 코드로 채움, 누락된 데이타 값 삽입


???
수정된
데이타
편집
오류
보고서
트랜잭션
트랜잭션
데이타
입력 장치
미편집
트랜잭션
편 집
편집된
트랜잭션
정 렬
편집 완료된
정렬 트랜잭션

◆ 편집과 정렬 작업을 위한 범용 유틸리티의 기능

1. 한 저장 장치에서 다른 저장 장치로 순차 화일을 복사
2. 같은 저장 장치에서 하나의 순차 화일을 다른 순차 화일로 복사
3. 레코드에 대한 간단한 편집 수행
4. 레코드에 대해 간단한 재구성 수행
5. 주어진 필드 값에 따라 오름차순 또는 내림차순으로 정렬
6. 여러 개의 정렬된 화일을 주어진 필드 값에 따라 오름차순 또는 내림차순으로 하나의 정렬된 화일로 합병

❖ 순차 화일의 갱신

◆ 순차 화일에서의 검색

- 레코드의 저장 순서에 따라 연속적으로 검색
- 레코드를 검색하고자 하는 순서에 따라 레코드 입력 순서를 결정

◆ 순차 화일에 대한 질의

- 화일 구조상 연속적인 검색의 경우에 효율적
  - 예) 사원 봉급의 평균과 표준 편차는 얼마인가?
  - 예) 네 개의 의료 보험에 각각 몇 명의 사원들이 가입되어 있는가?
- 화일의 질의 적중 비율(inquiry hit ratio)

$$= \frac{\text{질의에 응답하기 위해 접근해야 할 레코드 수}}{\text{화일 전체의 레코드 수}}$$

  - 질의 적중 비율이 높을수록 입력 순차 화일 구조에 더 적합

▶ 키 순차 화일의 갱신

- **키 순차 화일에 대한 레코드 삽입**
  - 키 값에 따라 오름차순/내림차순을 유지해야 하므로 복잡
    1. 두 기존 레코드 사이에 삽입 위치를 검색
    2. 이 삽입점 앞에 있는 모든 레코드를 새로운 화일에 복사하고 새로운 레코드를 삽입
    3. 삽입점 뒤에 있는 나머지 레코드들을 새로운 화일로 복사
- **키 순차 화일에서의 레코드 삭제**
  - 삽입과 거의 같은 단계를 거친다.
- **키 순차 화일에 있는 기존 레코드 수정**
  - 삽입, 삭제와 거의 같은 단계를 거친다.
  - 직접 접근 저장 장치를 사용하는 프로그래밍 언어
    - 순차 화일의 임의접근이 가능하므로, 레코드를 수정해서 화일에 있는 기존 레코드 위에 수정된 레코드를 기록할 수 있다.

▶ 순차 마스터 화일의 갱신

- 갱신 트랜잭션을 트랜잭션 화일에 모아서 일괄 처리
  - 트랜잭션 화일의 트랜잭션들
    - 마스터 화일과 똑같은 키로 정렬
    - 갱신 프로그램에 의해 마스터 화일에 적용
    - 각 트랜잭션은 대응하는 마스터 레코드의 키 값과 갱신 타입을 나타내는 갱신코드(update code)를 가지고 있다.
  - 마스터 화일에 적용하는 갱신
    - 새 레코드의 삽입(I)
    - 기존 레코드의 삭제(D)
    - 기존 레코드의 수정(M)

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

갱신을 위한 트랜잭션 레코드

- **트랜잭션의 삽입**
  - 트랜잭션 코드, 키값은 반드시 있어야 한다.
  - 삽입할 다른 데이터 필드 값
- **트랜잭션의 삭제**
  - 보통 해당 마스터 레코드의 키 값만을 지정해도 된다.
- **트랜잭션의 수정**
  - 키 값과 수정될 필드들과 해당 값만 명세
  - 일반적으로 수정하지 않을 필드는 공백으로 남겨 놓는다.

- **화일 갱신**
  - 이미 저장된 레코드의 삽입, 저장되지 않은 레코드 삭제 혹은 수정 등 여러 가지 오류를 고려
  - 갱신 프로그램 : 오류 보고서를 생성하여 수행하지 못한 모든 트랜잭션의 내용과 그 이유를 명시

▶ 마스터 화일 갱신 빈도수

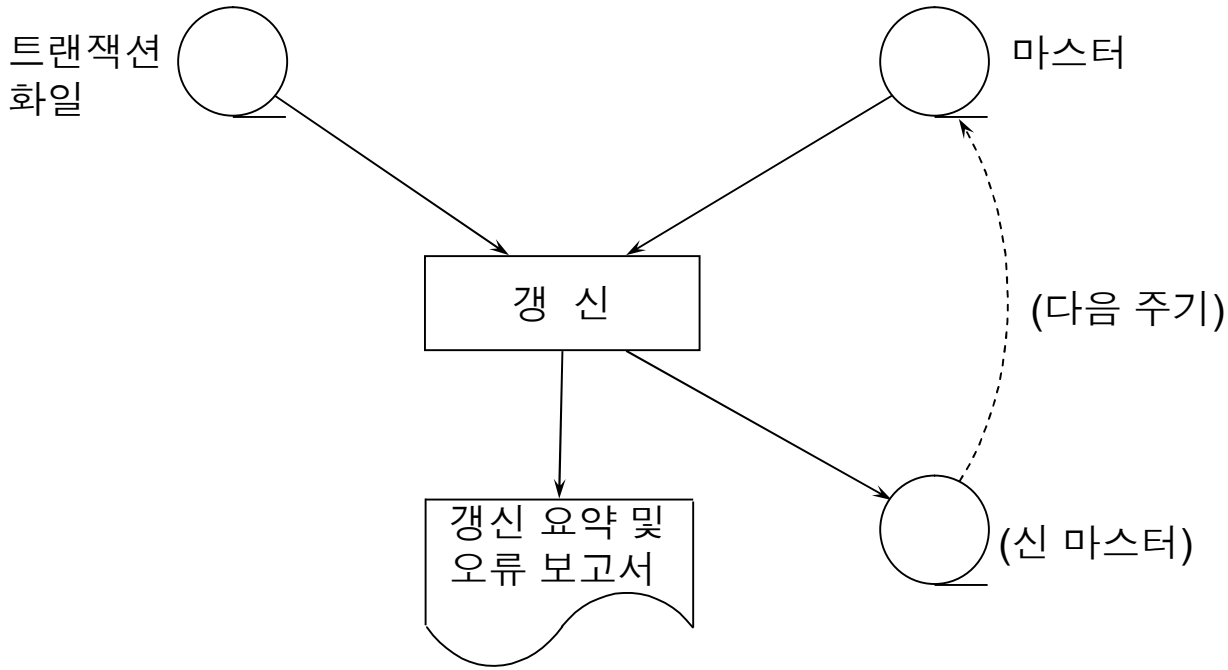
- 갱신 빈도수를 결정하는 요인
  - 데이타 변경율
  - 마스터 화일의 크기
  - 마스터 화일의 최신 데이타 요구
  - 화일 활동 비율

- **화일 활동 비율(file activity ratio)**

$$= \frac{\text{일련의 트랜잭션에 의해 영향을 받는 마스터 화일의 레코드 수}}{\text{마스터 화일의 총 레코드 수}}$$

▶ 순차 화일의 갱신 작업

- 마스터 화일 / 신마스터 화일
  - 화일 활동 비율이 낮을수록 신마스터 화일로 단순히 복사하는 레코드 수가 증가
  - 갱신 작업 종류 후 실행 과정에 일어난 여러 가지 오류와 갱신 요약을 보고서로 생성



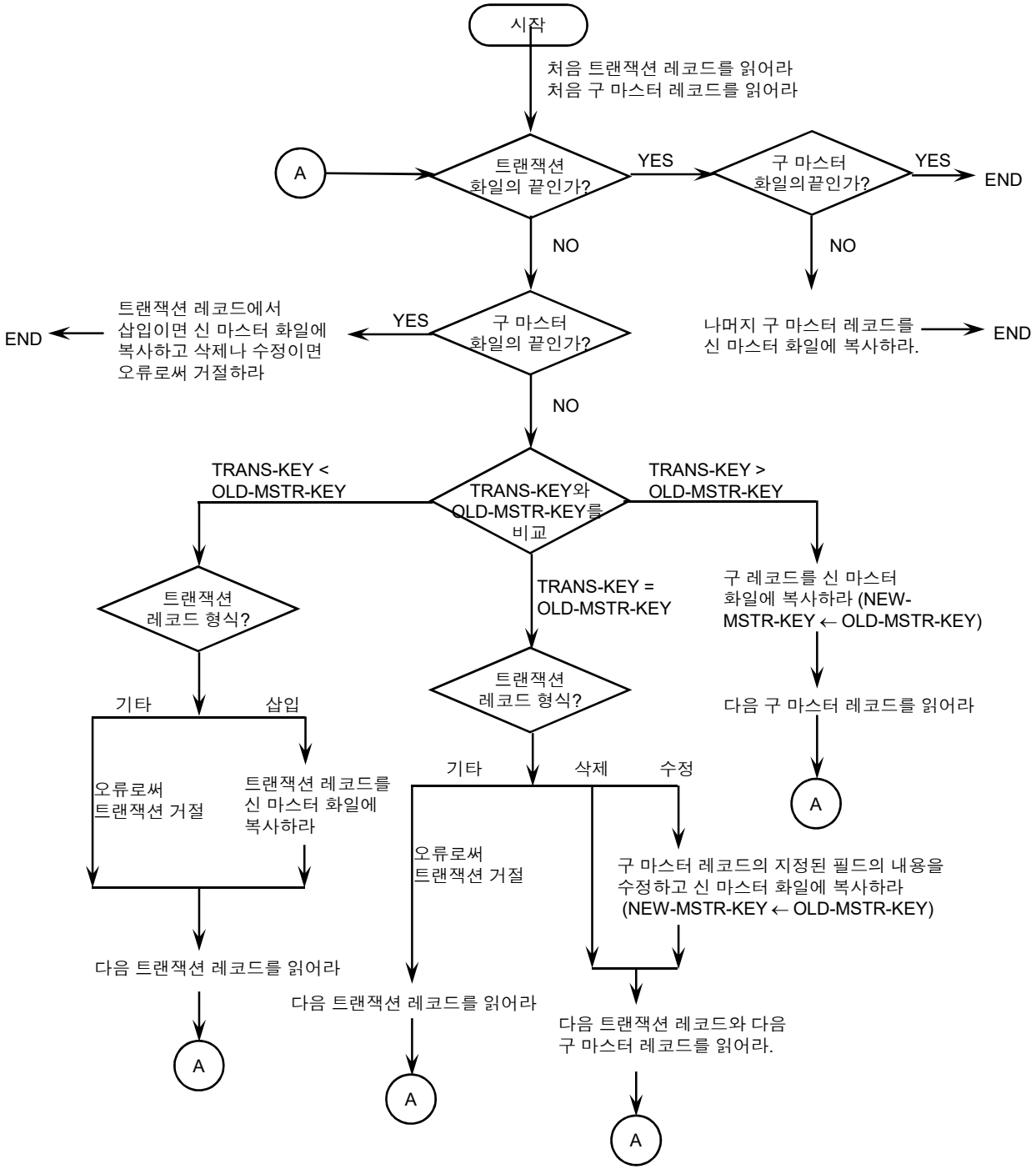
▶ 키 순차 마스터 화일의 갱신 알고리즘

- 가정
  - 트랜젝션 및 마스터 화일이 같은 정렬 키를 기초로 정렬됨
  - 각 마스터 레코드 마다 단 하나의 트랜젝션 레코드만 존재함
    (마스터 화일만 검사하면 트랜젝션 레코드를 처리할 수 있음)

- 갱신 알고리즘
  - 마스터 화일과 트랜잭션 화일을 비교
  - 어느 한 키 값이 두 화일에서 일치하면 갱신 프로그램은 갱신 코드에 따라 레코드를 수정 또는 삭제
  - 트랜잭션 레코드 키 값이 마스터 화일의 어떤 레코드의 것과도 일치하지 않으면, 새로 삽입할 레코드로 간주하고 마스터 화일에 삽입

- 오류 처리
  - 마스터 화일에 있는 키 값을 가진 레코드를 삽입
  - 마스터 화일에 없는 키 값을 가진 레코드를 삭제
  - 마스터 화일에 없는 키 값을 가진 레코드를 수정

- **transKey와 masterKey의 비교**
  - 갱신 알고리즘 중 핵심이 되는 부분
  - transKey : 트랜잭션 화일의 레코드 키
  - masterKey : 마스터 화일의 레코드 키
  - 가정 : 화일의 끝을 나타내는 EOF는 어떤 레코드 키 값보다 크다.


시작
처음 트랜잭션 레코드를 읽어라
처음 구 마스터 레코드를 읽어라
A
트랜잭션 화일의 끝인가?
YES
구 마스터 화일의끝인가?
YES
END
NO
NO
나머지 구 마스터 레코드를 신 마스터 화일에 복사하라.
END
YES
구 마스터 화일의 끝인가?
트랜잭션 레코드에서 삽입이면 신 마스터 화일에 복사하고 삭제나 수정이면 오류로써 거절하라
END
NO
TRANS-KEY < OLD-MSTR-KEY
TRANS-KEY와 OLD-MSTR-KEY를 비교
TRANS-KEY > OLD-MSTR-KEY
TRANS-KEY = OLD-MSTR-KEY
트랜잭션 레코드 형식?
구 레코드를 신 마스터 화일에 복사하라 (NEW-MSTR-KEY ← OLD-MSTR-KEY)
다음 구 마스터 레코드를 읽어라
A
기타
삽입
오류로써 트랜잭션 거절
트랜잭션 레코드를 신 마스터 화일에 복사하라
다음 트랜잭션 레코드를 읽어라
A
트랜잭션 레코드 형식?
기타
삭제
수정
오류로써 트랜잭션 거절
다음 트랜잭션 레코드를 읽어라
A
구 마스터 레코드의 지정된 필드의 내용을 수정하고 신 마스터 화일에 복사하라 (NEW-MSTR-KEY ← OLD-MSTR-KEY)
다음 트랜잭션 레코드와 다음 구 마스터 레코드를 읽어라.
A

- masterKey < transKey
  - 마스터 레코드에 적용할 트랜잭션 레코드가 없는 경우
  - 마스터 레코드를 새로운 마스터 화일로 복사만 하고, 다음 마스터 레코드를 읽어 온다.

- masterKey = transKey
  - 트랜잭션 레코드의 갱신코드에 따라 수행 내용이 다르다.

  - 수정(M)인 경우 : 레코드를 변경하여 새로운 마스터 화일에 삽입하고, 다음 트랜잭션 레코드를 읽어 온다.
  - 삭제(D)인 경우 : 마스터 레코드는 삭제, 즉 무시된다.
  - 삽입(I)인 경우 : 마스터 화일에 이미 같은 키 값을 가진 레코드가 있으므로 중복 레코드라는 오류 메시지를 프린트하고 다음 트랜잭션 레코드를 읽어 온다.

- masterKey > transKey
  - 트랜잭션 레코드에 일치하는 마스터 레코드가 없는 경우
  - 해당 트랜잭션 레코드는 삽입할 레코드이거나 오류
  - 갱신 코드가 삽입(I)인 경우 : 레코드를 구성해서 새로운 마스터 화일에 삽입
  - 그 이외의 경우 : 오류가 되어 적절한 오류 메시지를 프린트하고, 다음 트랜잭션 레코드를 읽어 온다.

- 하나의 마스터 레코드에 대해 적용할 트랜잭션이 다수
  - 트랜잭션들을 발생된 시간 순서에 따라 적용
    - 1차 키는 transKey, 2차 키는 트랜잭션 발생 시간을 기준으로 정렬한 뒤에 갱신 작업을 시작해야 한다.
    - 갱신된 레코드를 새로운 마스터 화일에 출력하기 전에 관련 트랜잭션들이 모두 처리되었는지 확인해야 한다.

❖ 순차 화일과 임의 접근

- 순차 화일의 저장
  - 순차 접근 저장 장치(자기 테이프)
  - 임의 접근 저장 장치(자기 디스크)에도 저장할 수 있다.

- 임의 갱신(**random update**)
  - 해당 마스터 레코드의 위치를 찾고, 이 마스터 레코드에 트랜잭션을 적용하고, 갱신된 마스터 레코드를 원래의 위치에 다시 기록
  - 효율적 : 해당 마스터 레코드만 읽고, 원 위치에 재기록
  - 임의 접근이 가능해야 한다.
  - 레코드 삭제 : 물리적 제거보다 해당 레코드 위에 “deleted” 표시를 기록하여 논리적 삭제로 대신하는 게 낫다.