CSE 206 - 파일 처리론
(File Processing)

1. 파일의 개념

Contents

- **1. 파일의 개념**
  - **1.1. 파일의 논리적 구조 및 물리적 저장 위치**
  - **1.2. 파일의 분류**
  - **1.3. 파일의 연산**
  - **1.4. 파일 구조 설계 요소**

1.1. 파일의 논리적 구조 및 물리적 저장 위치

파일이란

- **파일 (file)**
  - 어떤 공통적인 목적을 위해 함께 저장된 데이터 집합.

게임의 세이브 파일

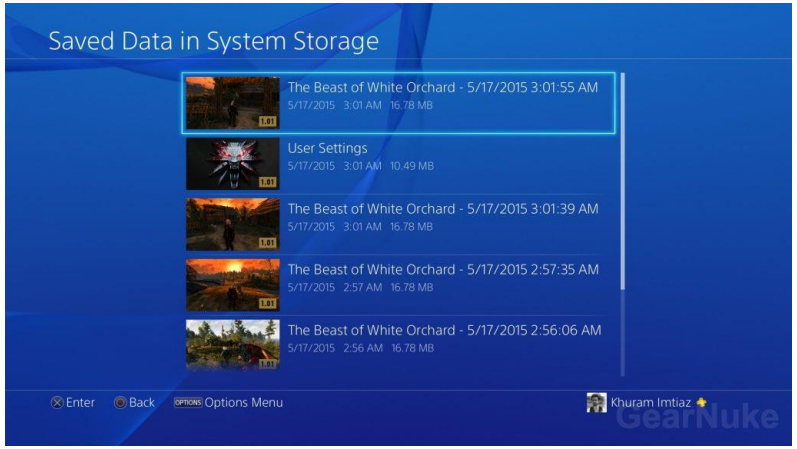
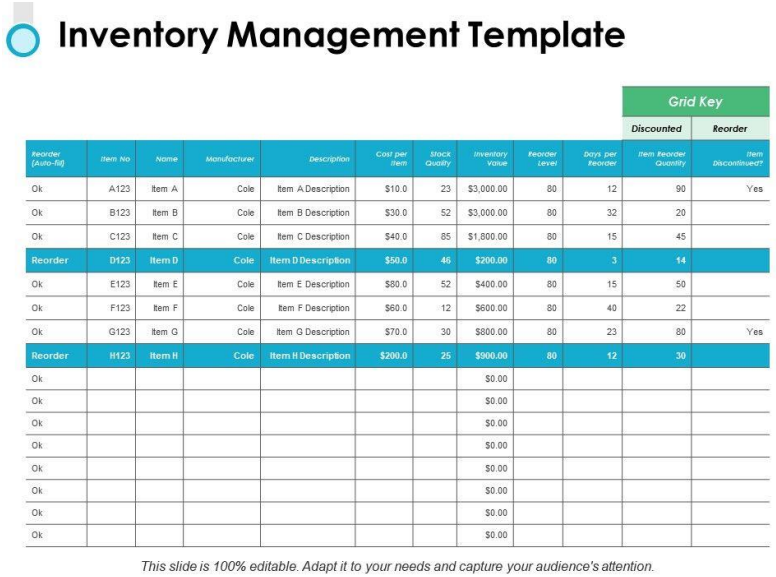


Image source : https://gearnuke.com/witcher-3-uses-neat-way-save-game-ps4-huge-file-size-can-easily-max-ps-cloud-storage/

재고 관리 파일



생각해 보기: 위 파일들에 저장되는 데이터는 무엇이고, 이들의 목적은?

파일: Data의 논리적 구조

- **파일 (file)**
  - 파일이 사람의 일처리를 위해 만들어지므로, 사람이 다루기 쉽도록 어느정도 **구조화된 데이터가 파일에 저장**되는 것이 일반적이다.

비구조화 된 데이터 예 : Text 데이터

첫째 과일은 사과이고 3000 원인데 전부 300 개 있었으나 100개 팔았다. 재고가 150개인 바나나는 사과와 같은 물량 들여왔고 가격은 천 원이다. 그리고 딸기는..

구조화된 데이터 예 (미리 정의된 방식으로 정리된 정보)

상품번호	이름	가격 (원)	판매한 개수	재고 수
과일_1	사과	3,000	100	200
과일_2	바나나	1,000	150	150
과일_3	딸기	5,000	80	120
과일_4	감	2,500	90	110

생각해 보기: 우리가 주로 접하는 Structured Data를 다루기 위한 프로그램은 무엇인가?

Structured Data (구조화된 데이터)

- **Structured Data**
  - (1) 미리 정의된 Data Model을 따르는 Data.
  - (2) Data 전체를 테이블 형태로 나타낼 수 있다.
    - 테이블의 한 Column : **Data field, Data Attribute, Data Item**라 한다.
      - 이름을 가진 논리적 데이터의 최소 단위
      - 특정 객체(object, entity)의 한 성질을 표현
    - 테이블의 모든 Colum을 합쳐서 **Record Type**이라 한다.
      - 논리적으로 서로 연관된 데이터 필드(항목)들의 집합으로 이들이 하나의 Data Object (객체)를 나타내는 정보이다.
    - 테이블의 Row :
      - 어떤 특정 객체의 Instance
      - 보통 레코드(record)라고 함

Structured Data (Cont'd)

(2) Data 전체를 테이블 형태로 나타낼 수 있다.

테이블의 한 Column : **Data field, Data Attribute, Data Item**라 한다.

이름을 가진 논리적 데이터의 최소 단위

특정 객체(object, entity)의 한 성질을 표현

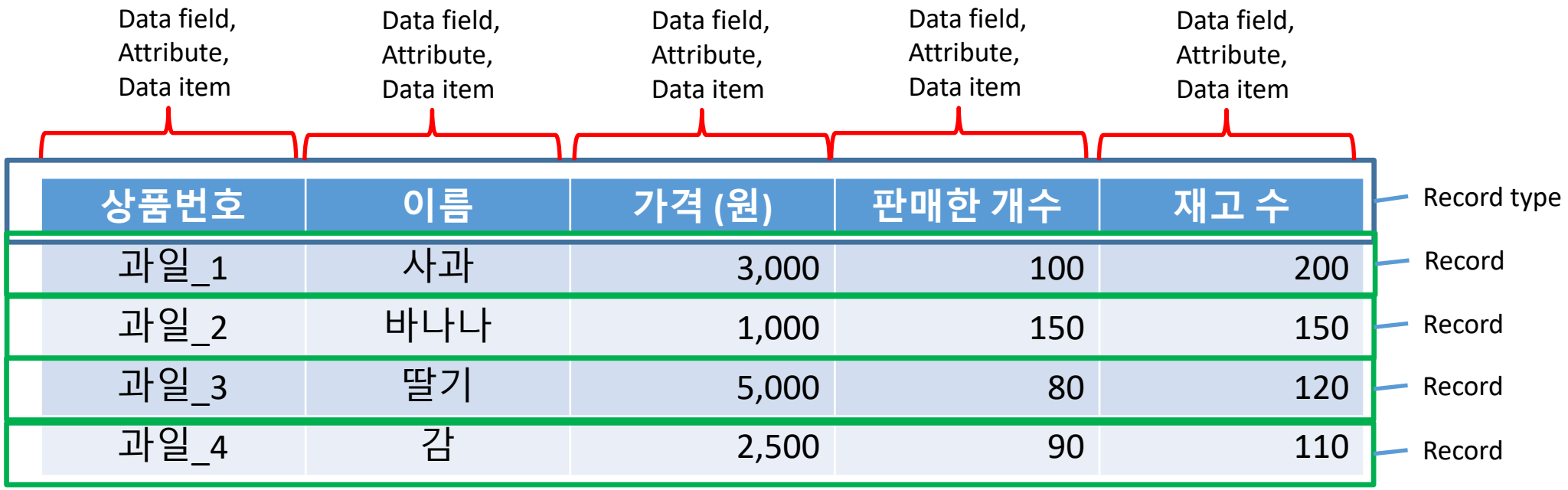
테이블의 모든 Colum을 합쳐서 **Record Type**이라 한다.

논리적으로 서로 연관된 데이터 필드(항목)들의 집합으로 이들이 하나의 Data Object (객체)를 나타내는 정보이다.

테이블의 Row :

어떤 특정 객체의 Instance

보통 **Record**라고 함



Structured Data : Data Model

- **Structured Data**
  - **(1) 미리 정의된 Data Model을 따르는 Data.**
- **Data Model**
  - Data 를 저장, 처리 및 접근하는 방법에 대한 정의
  - Data model이 정의 되어 있기 때문에
    - Data의 각 Field가 독립적으로 존재하고,
    - 이들 Field에 대해 단독 혹은 여러 개 동시에 접근하여 Data를 처리할 수 있다.
    - 예) 평균, 표준편차, 최소값, 최대값, 범위 연산 등.

Structured Data : Data Model (Cont'd)

생각해 보기:

1. [Data Model] 아래 재고 관리 파일에서 Attribute 왜 아래와 같이 정의 되었는가?
   - 아래 5개 Attribute는 재고 관리를 위해 충분한 데이터가 정의되어 있는가?
2. [Structured Data] 재고 관리에서 다음 정보가 필요하다고 할 때, Structured Data로 Data가 저장되어 있으면 좋은 점은? (Unstructured data에서 같은 일을 하려면?)
   - 재고가 가장 많이 남아 있는 상품은?
   - 각 상품에 대한 매출액은?

Structured Data:

상품번호	이름	가격 (원)	판매한 개수	재고 수
과일_1	사과	3,000	100	200
과일_2	바나나	1,000	150	150
과일_3	딸기	5,000	80	120
과일_4	감	2,500	90	110

Unstructured Data:

오늘 사과를 하나 3000원에 100개 팔아서 200개 남았다. 바나나는 150개 남았는데, 오늘 판 개수가 150이고 두개에 3000원에 팔았다. …

파일: 물리적 저장 위치 - 컴퓨터의 구조

입력장치 (Input Device):
외부의 정보(주위 환경 등) 및 사람의 명령을 컴퓨터에 알려준다.

1 + 1=?

CPU (Central Processing Unit):
명령에 따라 정보를 처리하여 결과를 생성한다. 정보 처리를 위해 메모리에 저장된 정보를 읽고 생성된 결과를 메모리에 저장한다.

1차 저장장치(Memory)
: 계산에 필요한 정보를 저장해 두는 장치. 빠르게 접근할 수 있으나 용량이 상대적으로 작다. 전원이 꺼지면 정보는 모두 사라진다.
(책상위의 메모지, 연습장)

2차 저장장치(HDD)
:계산에 필요한 정보를 저장해 두는 장치.
용량이 상대적으로 크나 접근 속도가 느리다.
전원이 꺼져도 정보는 계속 저장된다.
(책장에 꽂아 둔, 책 혹은 잘 정리된 노트)

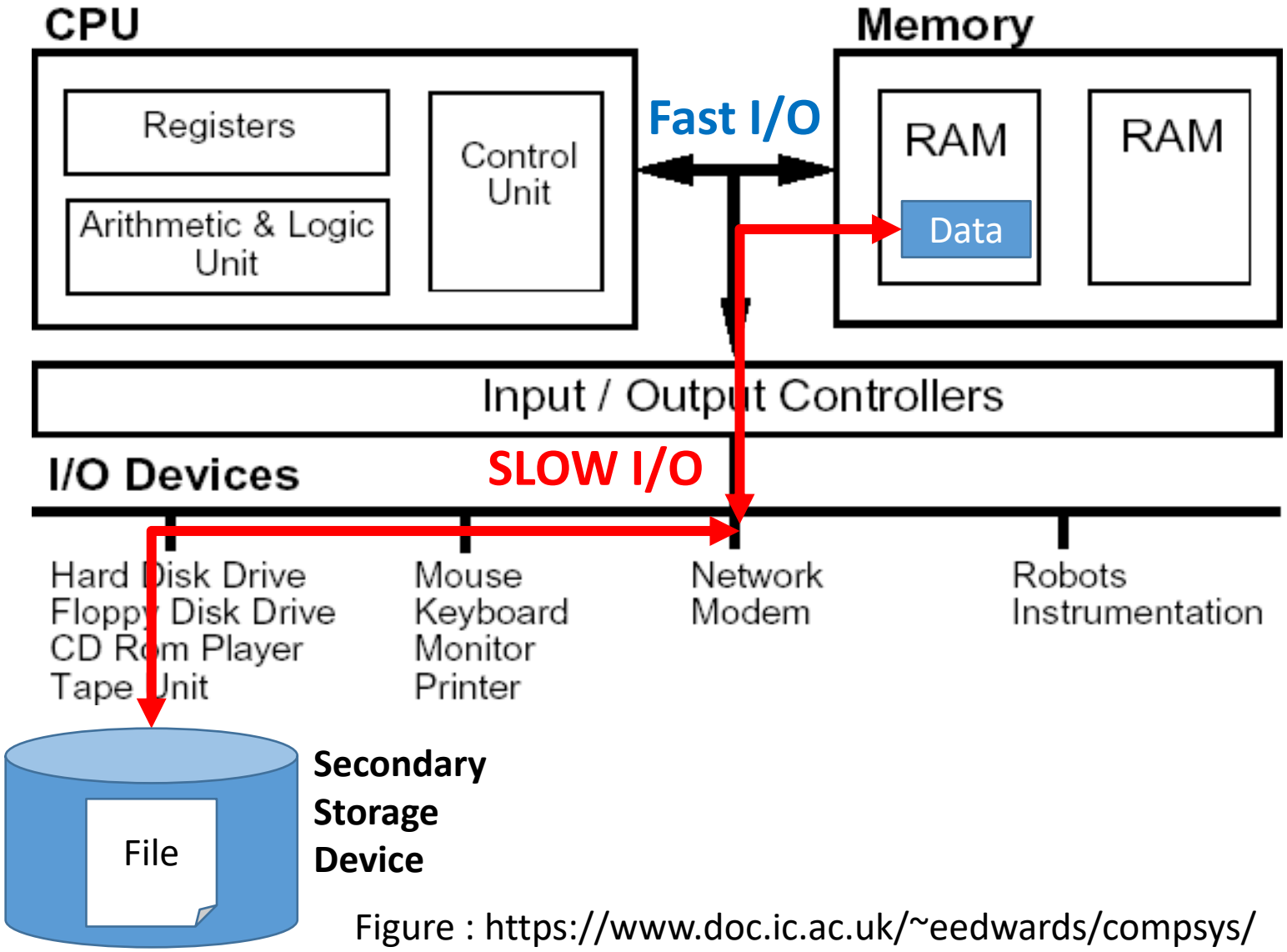
2

출력 장치 (Output Device):
연산(계산) 결과를 명령을 사람에게 알려준다.

파일: 물리적 저장 위치

- **파일은 2차 저장장치에 저장한다.**
  - Hard Disk, CD, SSD
- **데이터를 파일로 만들어 2차 저장장치에 저장해 두는 이유**
  - 시스템의 메인 메모리에 전부 로드하기에는 **Data 양이 너무 많다**.
  - 프로그램은 **특정시점에 파일에 저장되어 있는 Data의 일부만 접근**한다.
    - 파일에 저장되어 있는 Data 전부를 한번에 메인 메모리에 읽어 올 필요가 없다.
    - 예) Blue ray에 저장된 영화, 게임
      - 왜 Game에서 특정 시점에서 Load를 하는가?
  - Data를 특정 프로그램과 분리시켜 **Data의 독립성(independency)을 유지**하기 위해.
    - 여러 다른 프로그램이 동일 파일에 접근하거나 내용을 갱신할 수 있다.
  - 대부분의 파일은 **오랜 기간동안 저장**해야 한다.
    - 전원이 나가도 Data가 유지되어야 한다.

파일 처리론이 필요한 이유: 속도, 속도, 그리고 속도..



- 파일이 저장되는 2차 저장장치의 I/O는 Main memory I/O에 비해 느리다.

Figure : https://www.doc.ic.ac.uk/~eedwards/compsys/

Data 전송 속도를 높이려면?

- (1) **Application의 데이터 접근 Pattern을 이해**하고,
- (2) **2차 저장장치의 물리적 특성을 이해**하여,
- (3) 가능한한 **느린 I/O의 사용 횟수를 줄이면서, Application이 필요로 하는 데이터를 Main memory에** 가져 올 수 있도록 하여야 한다.
- 어떻게?
  - 파일 처리론은 이를 위해 발전되어 왔다.

Summary

- **파일의 논리적 구조**
  - 어느정도 구조화된 데이터(Structured Data)가 파일에 저장
    - 사람이 다루기 쉽도록
  - **Structured Data**
    - (1) 미리 정의된 Data Model을 따르는 Data.
    - (2) Data 전체를 테이블 형태로 나타낼 수 있다.
  - **Data Model**
    - Data 를 저장, 처리 및 접근하는 방법에 대한 정의
- 물리적 저장 위치
  - Main Memory에 비해 느린 2차 저장장치에 저장된다.
  - 느린 I/O의 사용 횟수를 줄이면서, Application이 필요로 하는 데이터를 Main memory에 가져올 수 있도록 하여, Data 처리 속도를 높여야 한다 => 파일 처리론의 목적

1.2. 파일의 분류

파일의 분류

파일을 용도에 따라 분류할 수 있으며 대표적으로는 다음과 같이 분류할 수 있다.

1. 파일이 만들어진 목적에 따라

- 마스터 파일 (master file), 트랜잭션 파일 (transaction file), 보고서 파일 (report file), ...

2. 프로그램이 파일에 접근하는 목적에 따라

- 입력 파일 (Input file), 출력 파일 (Output file), 입/출력 파일 (Input/output file)

1. 생성 목적에 따른 분류

파일이 만들어진 목적에 따라.

마스터 파일 (master file)

트랜잭션 파일 (transaction file)

보고서 파일 (report file)

작업 파일 (work file)

프로그램 파일 (program file)

텍스트 파일 (text file)

Master File : 원본 데이터

- **Master File :** 데이터의 **원본 데이터**
  - 특정 시점에서 어떤 **업무에 관한 정적인 상태를 나타내는 데이터의 집합**
  - 예(제조 회사) : 상품 재고 master file, 급여 master file, 고객 master file, …
  - 데이터를 저장하는 여러 다른 버전의 파일들이 Consistency가 다를 경우 Master 파일을 따른다.
    - 생각해 보기: 어떤 경우?
- 데이터의 Insertion (삽입), Deletion (삭제), Modification (갱신)을 통해 데이터를 변경하고 데이터는 비교적 영구적으로 저장/관리됨.
- 현재성(currency)을 정확히 유지함으로써 현실 세계에 대한 정확한 정보제공
- 일반적으로 파일이라 하면 Master File을 의미

Transaction File : 데이터에 대한 변경 작업 명세

- **Master file에 적용할 Transaction들을 모아 저장한 파일**

- **Transaction (트랜잭션)**
  - 논리적인 작업 단위
  - 하나의 건수로 처리되어야 하는 분리될 수 없는 연산 그룹

- **Transaction의 내용**
  - 새로운 레코드 삽입(insert)
  - 기존 레코드 삭제(delete)
  - 현존 레코드의 내용 수정(modification)

Transaction File : 데이터에 대한 변경 작업 명세 (Cont'd)

- **생각해 보기:**
- 하나의 Transaction : 하나의 건수로 처리되어야 하는 분리될 수 없는 연산 그룹

- - 다음 Master File에서 사과를 10개 판매했다면 어떤 데이터가 어떻게 변경되어야 하는지 (2 가지 변경) 생각해 보아라.
- - 2 가지를 하나의 Transaction으로 관리하지 않을 경우 생길 수 있는 문제점을 생각해 보라.

상품번호	이름	가격 (원)	판매한 개수	재고 수
과일_1	사과	3,000	100	200
과일_2	바나나	1,000	150	150
과일_3	딸기	5,000	80	120
과일_4	감	2,500	90	110

보고서 파일 (Report File) : Master File Data의 발췌

- 데이터에 대한 검색의 결과를 보여주기 위해 **검색한 데이터를 일정한 형식으로 정리**해서 저장해 놓은 파일
  - 하드 카피(hard copy) 보고서 출력 , 모니터에 디스플레이 등

상품번호	이름	가격 (원)	판매한 개수	재고 수
과일_1	사과	3,000	100	200
과일_2	바나나	1,000	150	150
과일_3	딸기	5,000	80	120
과일_4	감	2,500	90	110

3,000원 이상의 과일 이름과 가격을 보여 주세요.

이름	가격 (원)
사과	3,000
딸기	5,000

작업 파일 (work file)

- 어느 한 프로그램에서 생성된 출력 데이터를 다른 프로그램의 입력 데이터로 사용하기 위해 만드는 **임시 파일** (temporary file)

- 최종 목표를 달성하는 과정에서 만들어지는 **중간 결과를 저장**하는 파일
  - 예. Merge sort에서 Merge 이전의 부분 sorting을 한 결과를 저장하는 파일.

- 수학 문제 풀기 위한 연습장과 같은 개념.
  - 수학 답을 풀기위해서는 답을 계산하는 중간 계산들이 필요하다. 이를 연습장 (work file)에 쓴다.
  - 그러므로 임시로 만들어 사용한 후 삭제한다

프로그램 파일 (program file)

- **데이터를 처리하기 위한 명령어들을 저장**하고 있는 파일
  - 고급언어(C, JAVA) 혹은 저급 언어 (어셈블리어)로 작성
  - object code, execution code를 말함.

텍스트 파일 (Text file)

- **문자, 숫자, 그래픽 데이터를 포함하고 있는 파일**로서 텍스트 편집기의 입력과 출력으로 사용
  - 여러 텍스트 (이미지) 편집기에 의해 처리될 수 있음

프로그램의 파일 접근 목적에 따른 분류

- **input file (입력 파일)**
  - 프로그램이 읽기(Read) 위해 접근하는 파일
  - 예) MP3 파일, 동영상 파일. 프로그램 실행 파일

- **output file (출력 파일)**
  - 프로그램이 기록(Write)하기 위해 접근하는 파일
  - 예) 로그 파일

- **input/output file (입/출력 파일)**
  - 프로그램의 실행 중 읽기도 하고 쓰기도 하기 위해 접근하는 파일
  - 예) 텍스트 파일, 읽고 쓰기가 가능한 Master File.
  - 입출력 파일에는 어떤 파일들이 있을까?

Summary

파일은

- **만들어진 목적에 따라**
  - **Master file, Transaction file, Report file, Work file, program file, text file**

- **프로그램의 파일 접근 목적에 따라**
  - **input file, output file, input/output file 로 분류된다.**

- 로 분류된다.

1.3. 파일의 연산

파일에 대한 연산 (File operations)

- File creation: Data 저장 및 접근 방식 결정 및 공간 확보
- File open: 파일 연산을 위해 버퍼 할당
- File write: 레코드 갱신/삽입/삭제
- File read: 파일 이름, 블록 명세, 레코드 읽기
- File close: 파일 연산을 위해 할당한 버퍼의 반환
- File deletion : 파일과 관련된 모든 데이터 삭제

File creation

- 파일을 새로 만드는 것
  - Program 작성할 때 File Creation은 fopen() 등의 함수에서 Creation까지 처리.
    - File이 존재하지 않으면 Creation, File이 존재하면 Open.

File Creation (Cont'd)

- File Creation시 실제로 어떤 일이 일어나는가?
  - Data 저장 및 접근 방식 결정 및 결정된 저장 및 접근 방식에 따라 필요로 하는 정보를 OS에 알려줌.
    - 데이터 골격 (skeleton)의 설계
      - 데이터 접근은 Random Access를 지원할 것인가? VS Sequential Access를 지원할 것인가?
      - 저장하는 Data는 Binary Data인가? Text Data인가?
      - 등등..
  - Data Loading을 위한 준비
    - 디스크 공간 할당 요청 (OS 에)
    - Data를 한번에 저장할 것인가? vs. 나누어 순차적으로 저장할 것인가?
    - Directory에 File 정보 등록
      - 저장장치에서 파일의 Data가 저장된 물리적인 위치
      - 파일 크기
      - 등등

File Open

- 파일에 Data를 저장하거나 파일에서 Data를 읽어올 수 있도록 준비.
  - C의 fopen()
  - OS에서 Disk에서 필요한 Data를 읽어오는데 필요한 정보를 읽어서 Main memory에 Load.
    - 파일 저장 형태 (Text vs Binary)
    - 저장장치에서 파일의 Data가 저장된 물리적인 위치
    - 파일 크기
  - Main Memory <-> Disk 간의 파일 데이터 전송을 위한 Buffer 할당 등.

File write

- 파일에 내용을 기록 하거나 출력 한다 (output)
  - C의 fwrite()
    - i) 기존 Record 내용의 Update
    - ii) 새로운 Record의 Insert
    - iii) 기존 Record 삭제 Delete

Remind: 아래 테이블에서 Record 하나의 Update, Insert, Delete를 해보라.

상품번호	이름	가격 (원)	판매한 개수	재고 수
과일_1	사과	3,000	100	200
과일_2	바나나	1,000	150	150
과일_3	딸기	5,000	80	120
과일_4	감	2,500	90	110

File read

- 파일의 내용을 읽어 와서 응용의 입력으로 사용한다.
  - C의 fread()

File close

- 파일의 닫기
  - 파일 사용 종료
    - 메인 메모리에 파일 전송 버퍼의 출력 데이터를 디스크에 기록 후
    - 파일 관련 자원 (전송 버퍼 등) 할당 해제

File deletion

- 파일의 삭제
  - 디렉터리로부터 파일 위치 검색
  - 할당된 디스크 공간 반환, 디렉터리 에서 파일 정보 삭제

1.4. 파일 구조 설계 요소

파일 구조 설계 요소

- OS나 프로그래밍 언어 Library로 이미 제공되는 file operation API를 사용하지 않고,
- OS, 2차 저장장치 구조까지 고려하여
- **빠르고** 효과적으로
- Data를 2차 저장장치에 저장하거나 Main memory로 읽어오는 파일을 설계하려면 어떤 점을 고려해야 할까?

파일 구조 설계 요소 – 속도를 어떻게 평가하는가?

- 파일 구조 설계의 목표 : 속도, 속도, 그리고 속도..

- 속도 평가
  - 2차 저장장치에 저장되어 있는 데이터를 읽어서 사용하거나 (File Read), 계산한 데이터를 2차 저장장치에 저장하는 것 (File Write)이 파일 연산이다.

  - **[생각해 보기]** File Read, File Write에 관여하는 저장장치를 2개 생각해 보아라

파일 구조 설계 요소 – 속도를 어떻게 평가하는가?

- **메인 메모리 속도 평가**
  - 데이터 접근 (I/O) 시간은 모두 일정한 것으로 가정
    - 왜?
      - Main Memory와 Disk의 접근 속도 차이.
      - Memory Address 지정에 의한 직접 접근.
  - 메인 메모리의 데이터 구조는 **비교 연산 횟수로 평가**
    - 왜 비교 연산일까?

파일 구조 설계 요소 – 속도를 어떻게 평가하는가? (Cont'd)

- **2차 저장장치(보조 저장장치)의 속도 평가**
  - 2차 저장장치 데이터 접근 시간이 메인 메모리에 비해 상당히 길다.
  - 2차 저장장치의 **접근 횟수**가 파일 구조 평가의 주요 요소
    - 왜 Main memory는 비교 연산인데, 2차 저장장치는 접근 횟수인가?
- 파일 구조 설계의 방향성
  - 만드려는 파일에서 가장 많이 사용되는 사용 형태 및 파일 연산(Operation)에 대해
  - **2차 저장 장치의 접근 횟수를 가장 작게 만들 수 있는 파일 구조** 설계가 중요
    - 왜? Main memory는 고려 안해도 되는가?

파일 구조 설계 요소

- 파일 구조 설계 시 고려할 요소
  - **가변성**
  - **활동성**
  - **사용 빈도수**
  - **응답 시간**
  - **파일 크기**
  - **파일 접근 유형**

가변성(volatility)

- **파일 내용이 얼마나 자주 변하는가**
- **파일의 성격**
  - 내용이 변하지 않는 정적 (static) 파일 (과거의 기록)
    - 예) 시스템 로그 파일, 2018년도 인구 조사 기록
  - 내용이 자주 변하는 동적 (dynamic) 파일 (현재의 상황 데이터)
    - 예) 상품 판매 및 재고 데이터

활동성(activity)

- **파일의 활동성**
  - 주어진 기간 동안에 파일의 총 레코드 수에 대해 접근한 레코드 수의 비율
  - 활동성이 높으면 순차 파일 구조가 유리할 수 있다.
    - 활동성이 높다 = 파일의 거의 모든 레코드에 접근한다. = 순차 파일 구조를 사용해서 처음 레코드 부터 끝 레코드까지 읽는 것이 접근 속도 면에서 더 나을 수 있다.

총 레코드 수 10인 데이터가 파일에 저장되어 있고 주어진 기간동안
활동성이 0.1 인 경우는 몇 개의 레코드에 접근한 경우인가?

활동성이 1.0 인 경우는 어떤 경우인가?

사용 빈도수 (frequency of use)

- **파일의 사용 빈도수**
  - 일정 기간 동안의 파일의 사용 빈도수 (=“Open – read or write – Close” 되는 횟수)
  - 가변성과 활동성에 밀접히 관련

응답 시간(response time)

- **응답 시간**
  - 응답시간: 검색이나 갱신 요청을 완료할 때까지 걸리는 시간
    - 요청을 주었을 때부터 결과가 나올 때 까지의 시간
      - 레코드 3 검색 요청을 19:00 에 했는데 레코드 3의 내용이 19:03에 나오면 응답시간은?
        - 1) 1 분　　2) 3 분　　3) 5분　　4) 9 분
  - 일반적으로, 빠른 응답 시간이 필요한 데이터를 저장하기 위해서는 임의 접근(random access) 방법을 선택하는 것이 좋다.
    - 필요한 일부 레코드에 접근하기 위해 순차 접근으로 파일 안의 거의 모든 레코드를 읽는 것은 비효율적이다.

파일 크기(file size)

- **파일 크기**
  - 레코드 수와 각 레코드 길이(record length)가 파일 크기 결정
    - 파일_크기 ∝ 레코드_길이 × 레코드_수
  - 일반적으로 시간이 지남에 따라 파일 크기는 성장

    (레코드 길이 확장, 레코드 수 증가)
  - 파일 크기의 성장을 유연하게 수용할 수 있는 구조가 필요

레코드 길이

상품번호	이름	가격 (원)	판매한 개수	재고 수
과일_1	사과	3,000	100	200
과일_2	바나나	1,000	150	150
과일_3	딸기	5,000	80	120
과일_4	감	2,500	90	110

레코드 수

파일 접근 유형

- **파일 접근 유형**
  - 파일 연산의 유형과 접근 형식에 따라 파일 구조 결정

ex) 1. 파일 접근이 읽기 위주 접근 ? 쓰기 위주 접근 ?

2. 파일 레코드들을 순차 접근이 주도 ? 임의 접근이 주도 ?

Summary: **파일 구조 설계 요소**

- 파일 구조 설계 시 고려할 요소
  - **가변성**
  - **활동성**
  - **사용 빈도수**
  - **응답 시간**
  - **파일 크기**
  - **파일 접근 유형**

Appendix

파일 접근 형태 – Sequential access Vs. Random access

- **Sequential access**
  - 읽을 Data Record의 순서가 항상 일정하고 이 순서대로 순차적으로 읽어가는 접근.
  - Record를 순차 접근의 순서대로 (물리적으로) 2차 저장장치에 저장하면 다음에 필요한 Record가 현재 Record 바로 뒤에 있으므로 데이터 처리의 시간을 단축 시킬 수 있다.

상품번호	이름	가격 (원)	판매한 개수	재고 수
과일_1	사과	3,000	100	200
과일_2	바나나	1,000	150	150
과일_3	딸기	5,000	80	120
과일_4	감	2,500	90	110

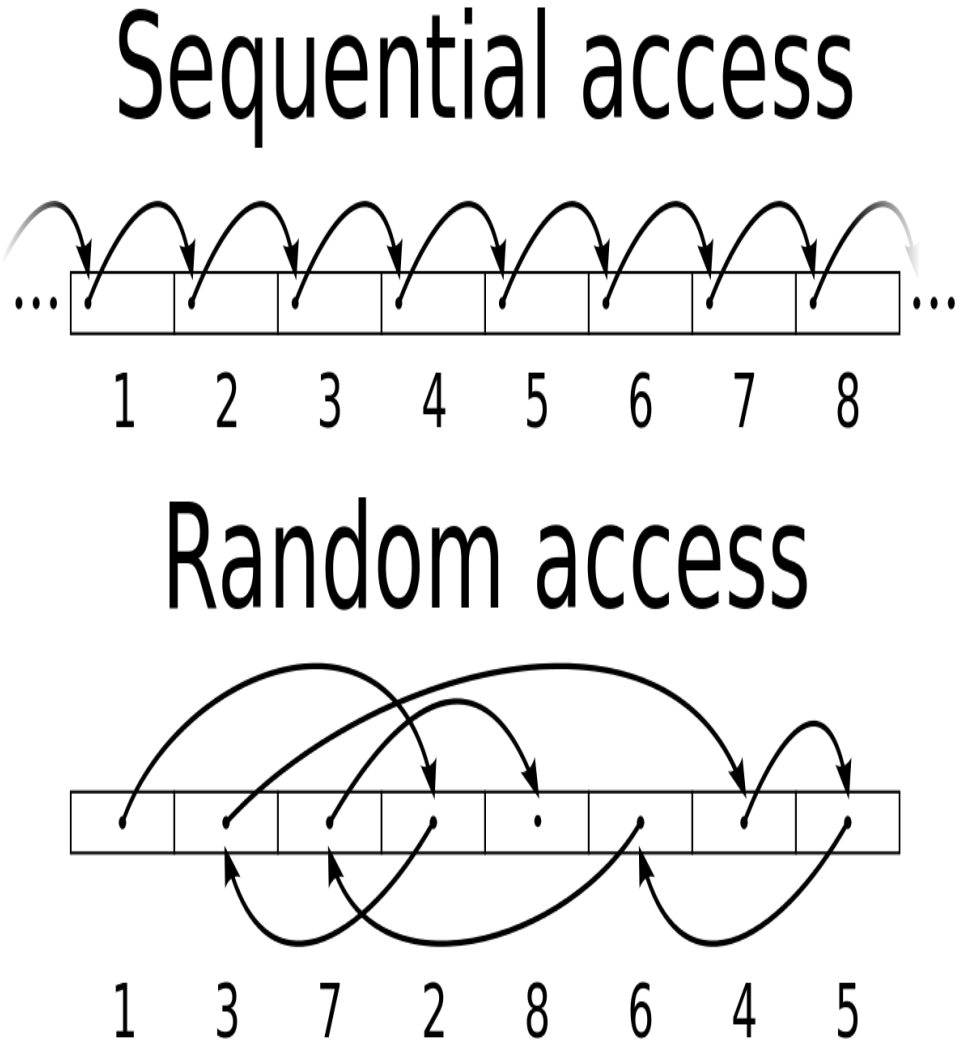


Figure :https://en.wikipedia.org/wiki/Sequential_access#/media/File:Random_vs_sequential_access.svg

파일 접근 형태 – Sequential access Vs. Random access (Cont'd)

- **Random access (임의 접근)**
  - 읽을 Data Record의 순서가 일정하지 않고 임의의 Data Record 에 접근하여 읽는 방식.
  - 임의의 데이터에 접근하는 시간이 접근하는 데이터에 관계없이 거의 일정하면서 충분히 빠른 구조가 필요.

상품번호	이름	가격 (원)	판매한 개수	재고 수
과일_1	사과	3,000	100	200
과일_2	바나나	1,000	150	150
과일_3	딸기	5,000	80	120
과일_4	감	2,500	90	110

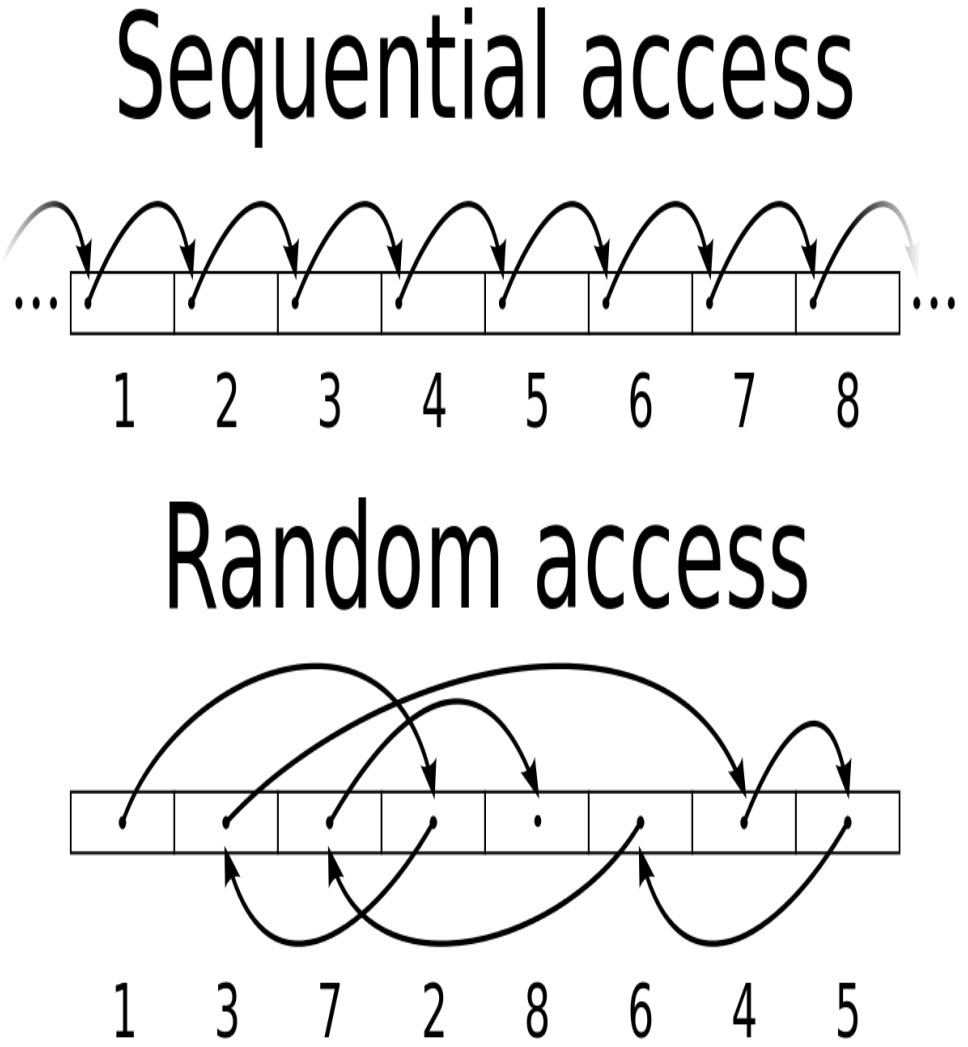


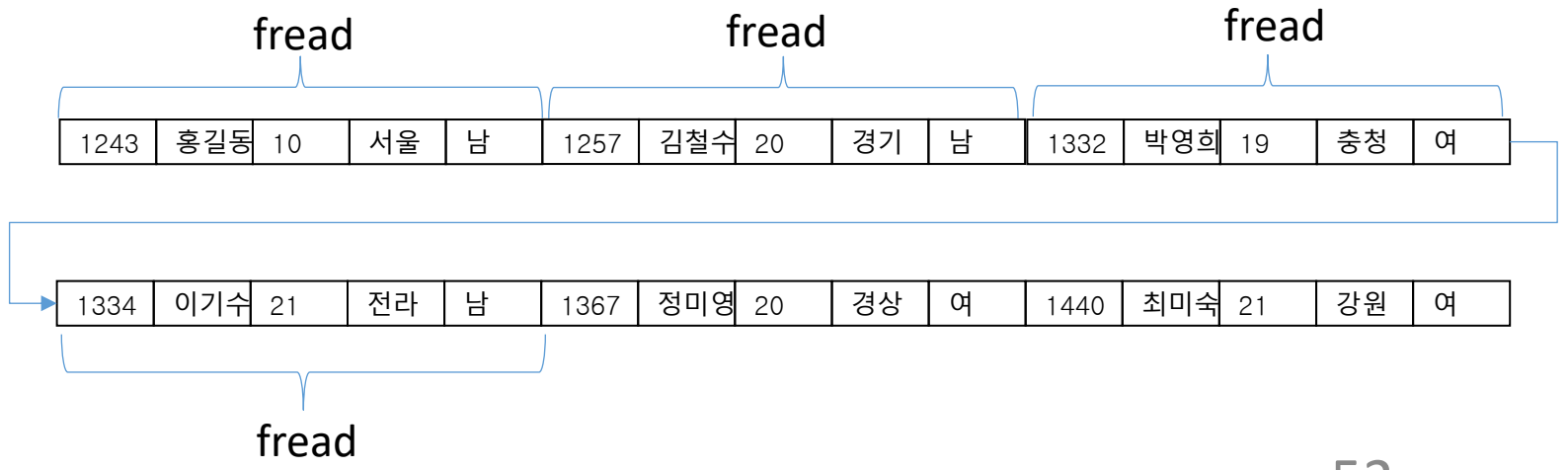
Figure :https://en.wikipedia.org/wiki/Sequential_access#/media/File:Random_vs_sequential_access.svg

스트림 파일(stream file)

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

Program A

```
int main() {
  ..
  ..
  ..
   char my_buf[RECORD_SIZE + 1];
   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);
   printf("%s", my_buf)
   ...
}
```



스트림 파일(stream file)

- **종류**
  - **Sequential Access Stream File (순차 접근 스트림 파일)**
    - 순차 접근만 허용
    - **open, read, write, close 함수만 제공**된다.
  - **Random Access Stream File (임의 접근 스트림 파일)**
    - 임의 접근이 허용
    - 위에 더해 **임의 위치에 접근 할 수 있는 seek 함수**도 제공
- **파일 접근 모드(access mode)**
  - 파일 열기 시 수행하려는 연산을 명세: 읽기(read), 쓰기(write), 붙이기(append) 등