

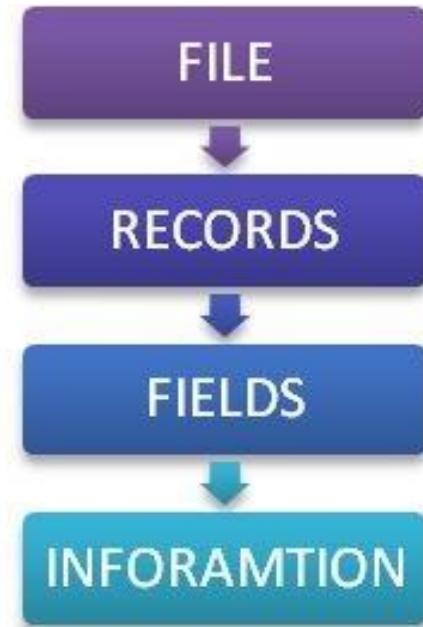
# Tutorial 2

COBOL programming

# More on COBOL

# Files

- COBOL is generally used to process record-based files.
- Each line in the files is a record.
- A record is a collection of fields.
  - Collection of items of information about a object
- Records will have same record structure.



# File Type

Sequential files	Indexed files
Read the records from top to bottom	Read the records based on the key
Can have duplicated data	Cannot have duplicated data
Not Sorted	Sorted by key
Stored in tape or disk	Stored in disk only
Frequently used	Rarely used

We use sequential files in the assignment.

# Sequential files

Declaration → Open → Process → Close

Three important things:

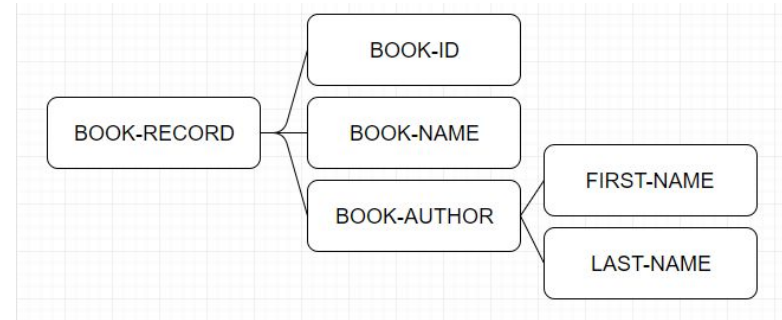
- Link the file identifier to an external file (INPUT-OUTPUT SECTION)
- Declare the file structure (FILE SECTION)
- Open, process, close the file in the program (PROCEDURE DIVISION)

# Sequential files

## Declare a record

- Specify the type and size of each field, as well as the hierarchy
- Recall:

```
01 BOOK-RECORD.  
  05 BOOK-ID PIC 9(5).  
  05 BOOK-NAME PIC X(20).  
  05 BOOK-AUTHOR.  
    10 FIRST-NAME PIC X(20).  
    10 LAST-NAME PIC X(20).
```



# Sequential files

The record type of the file is described in FILE SECTION.

- FD entry (File description)

```
000120 DATA DIVISION.
```

```
000130 FILE SECTION.
```

```
000140 FD INPUT-FILE.
```

```
000150 01 BOOK-RECORD.
```

```
000160 05 BOOK-ID PIC 9(5) .
```

```
000170 05 BOOK-NAME PIC X(20) .
```

```
000180 05 BOOK-AUTHOR.
```

```
000190 10 FIRST-NAME PIC X(20) .
```

```
000200 10 LAST-NAME PIC X(20) .
```

```
000210 FD ANOTHER-INPUT-FILE.
```

```
...
```

- File identifier

- Record type

No VALUE clause for all declarations in FILE SECTION.

# Sequential files

In previous example, we use INPUT-FILE to represent the input file.

However, on the disk, the input file is called “input.txt”, not INPUT-FILE.

We have to link the file and the variable in INPUT-OUTPUT SECTION.

Example:

```
000070 ENVIRONMENT DIVISION.  
000080 INPUT-OUTPUT SECTION.  
000090 FILE-CONTROL.  
000100         SELECT INPUT-FILE ASSIGN TO 'input.txt'  
000110         ORGANIZATION IS LINE SEQUENTIAL.
```



# Sequential files

Syntax:

SELECT [file identifier] ASSIGN TO [external file]

ORGANIZATION IS LINE SEQUENTIAL

FILE STATUS IS [variable].

Optional

- Default is 'Record sequential'. There is nothing in between two records.
- “Line sequential” means records are separated by '\r\n'.
- '\r\n' is appended or removed automatically when writing or reading in line sequential mode

- File status describes how a file operation completed.  
(OPEN, READ, WRITE, CLOSE)

# File status

File status indicates the cause of error. It contains two digits.

Define a variable for the file status of a file (in WORKING-STORAGE SECTION)

- 01 FS PIC 99.

Meaning:

- 00: no error
- 10: end-of-file
- ... See more from [here](#).

# File I/O - Opening & Closing

- File Opening
  - OPEN [mode] [file identifier]
  - Mode: INPUT, OUTPUT, I-O, EXTEND
- File Closing
  - CLOSE [file identifier]

# File I/O - Reading

- Suppose the file structure is:

```
10234Harry Potter
12345Twilight
23456Animal Fram
24896Brave New World
...
```

- Define the file description

```
FD IN-FILE.
01 BOOK-RECORD.
    05 BOOK-ID PIC 9(5).
    05 BOOK-NAME PIC X(20).
```

# File I/O - Reading

- To read one record of the file:
  - READ [file identifier]
  - **NOT allowed to use AT END clause**
- To read next record, execute again.
- Example:
  - READ IN-FILE
  - BOOK-ID contains book ID
  - BOOK-NAME contains the name of the book

# File I/O - Writing

- Suppose the file structure is:

```
10234Harry Potter
12345Twilight
23456Animal Fram
24896Brave New World
...
```

- Define the file description

```
FD OUT-FILE.
01 BOOK-RECORD.
    05 BOOK-ID PIC 9(5).
    05 BOOK-NAME PIC X(20).
```

# File I/O - Writing

- To write one record into the file:
  - WRITE [record name]
- Example:
  - MOVE 755 TO BOOK-ID.
  - MOVE 'Linux Manpage' TO BOOK-NAME.
  - WRITE BOOK-RECORD.

# Multiple record-type sequential files

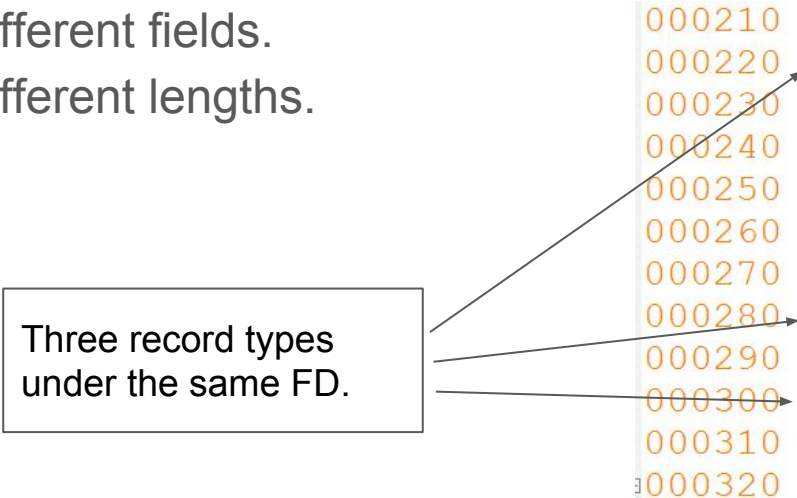
A file can have multiple record types.

Different record types can

- have different fields.
- have different lengths.

Three record types  
under the same FD.

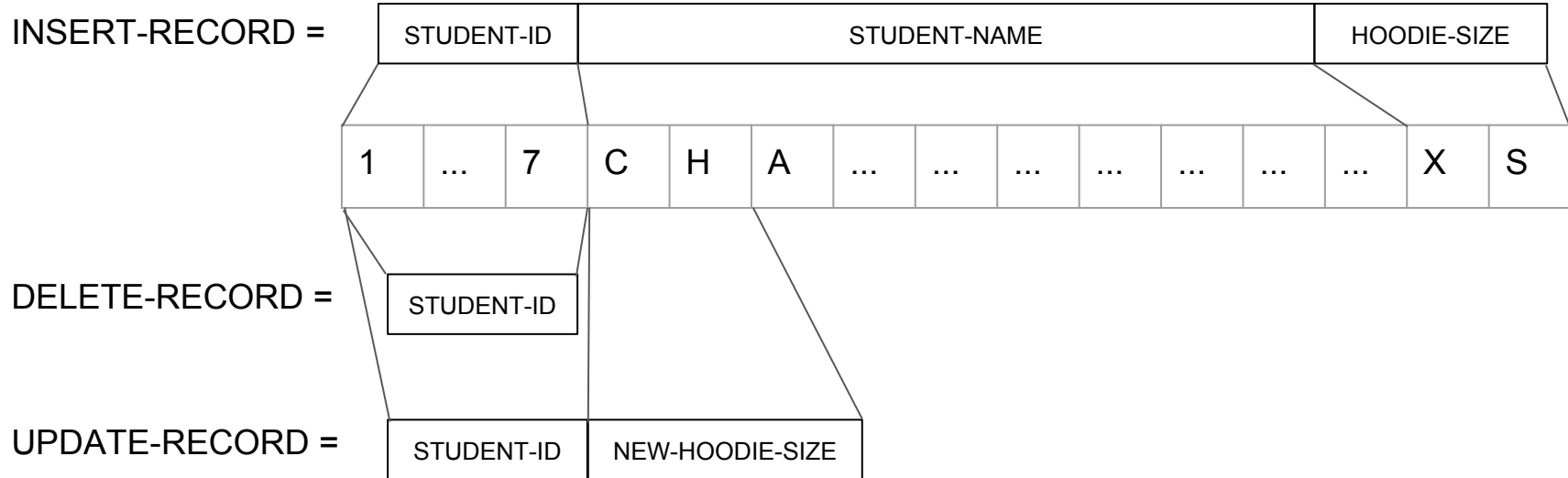
```
000210 FD UPDATE-FILE.  
000220 01 INSERT-RECORD.  
000230      05 STUDENT-ID PIC 9(10).  
000240      05 STUDENT-NAME.  
000250          10 FIRST-NAME PIC X(20).  
000260          10 LAST-NAME PIC X(20).  
000270      05 HOODIE-SIZE PIC XX.  
000280 01 DELETE-RECORD.  
000290      05 STUDENT-ID PIC 9(10).  
000300 01 UPDATE-RECORD.  
000310      05 STUDENT-ID PIC 9(10).  
000320      05 NEW-HOODIE-SIZE PIC XX.
```





# Multiple record-type sequential files

Suppose the input file looks like the following, then all record types represent,



# Reading with multiple record-type file

- When reading the file by
  - READ [file identifier]
  - e.g. READ UPDATE-FILE.
- Contents in all record types will be updated simultaneously.
  - Actually, only one record buffer is created for the file.
  - All record declarations of a file are mapped to the same record buffer.
  - When reading the file, contents in buffer change, hence contents in all record types change simultaneously

# Writing with multiple record-type file

- Choose a record type to write to the file
- Move the contents to the variable in that record type
- Write to the file by
  - WRITE [record name]

# Naming in multiple records

- COBOL allows same name for different variables under different records
- To let compiler know which variable you are referring to, you must quantify the variable with the record name.
- Syntax: [variable] OF/IN [record]
- Example:

```
01 REC-A.  
  05 NUM-A.  
    10 NUM PIC 9.  
  05 NUM-B.  
    10 NUM PIC 9.  
01 REC-B.  
  05 NUM-A.  
    10 NUM PIC 9.
```

- DISPLAY NUM OF NUM-A OF REC-A.
- DISPLAY NUM IN REC-B.
- DISPLAY NUM IN NUM-A OF REC-B.
- DISPLAY NUM IN NUM-B.

Or you may see  
error: 'NUM' is ambiguous; need quantification

# Filler

- When an area of storage must be declared but we don't care the name, we can use the special name FILLER.
- Example:

```
FD IN-FILE.  
01 BOOK-RECORD.  
    05 BOOK-ID PIC 9(5).  
    05 FILLER PIC X(20).
```

# Sorting

- COBOL has internal sorting function SORT
- Specific the input for sorting by INPUT PROCEDURE
- In INPUT PROCEDURE, choose the records that you want to sort by RELEASE
- Syntax:

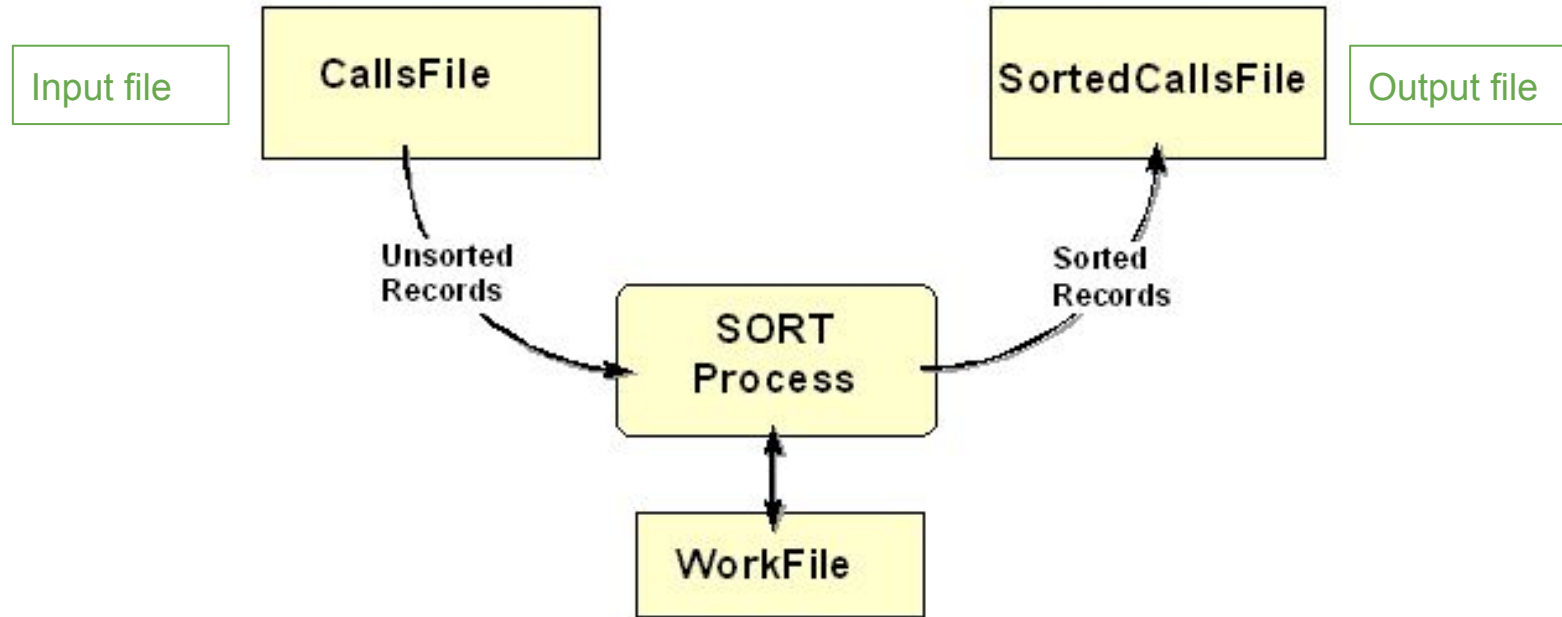
SORT [work file identifier]

ON ASCENDING/DESCENDING KEY [sort key identifier]

INPUT PROCEDURE IS [input procedure name]

GIVING [output file identifier]

# Sorting



# Sorting

- Link the work file and output file in INPUT-OUTPUT SECTION.
- Define a sort file description (SD) in FILE SECTION.
- Record in SD must contain the key field for sorting
- Example:

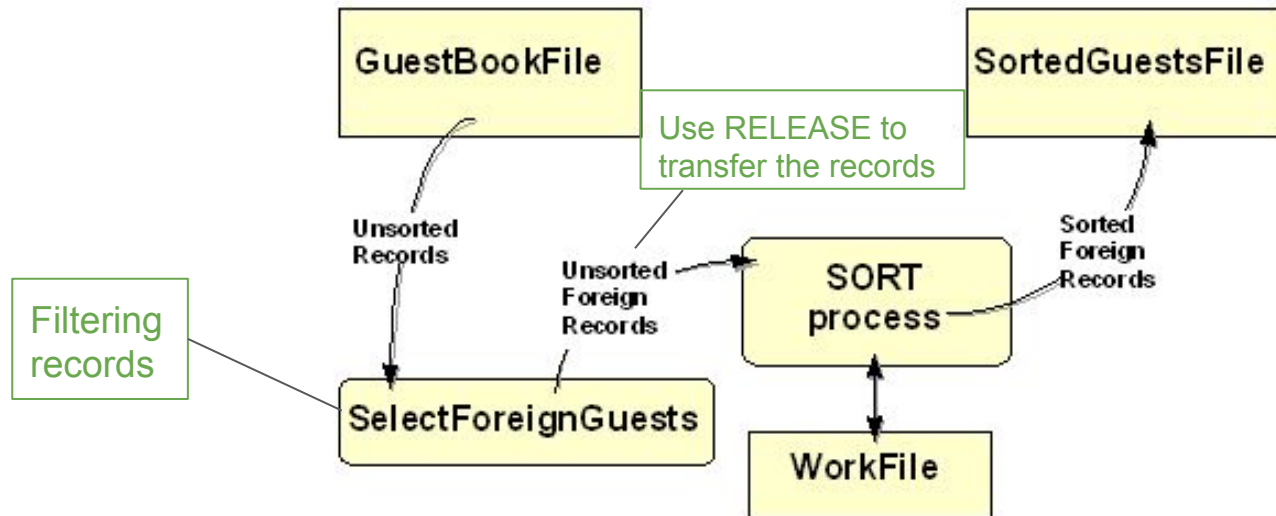
```
SD WORK-FILE.  
01 BOOK-RECORD.  
    05 BOOK-ID PIC 9(3).  
    05 BOOK-NAME PIC X(20).  
  
....  
  
    SORT WORK-FILE ON ASCENDING KEY BOOK-ID  
    ....
```



# Sorting

INPUT PROCEDURE is to select which records in input files to be sorted.

- Must contain at least one RELEASE statement to transfer the records



# Sorting

Syntax: RELEASE [SD record name]

Example:

```
GET-BOOK-STARTED-WITH-A.  
OPEN INPUT BOOK-FILE.  
READ BOOK-FILE.  
....  
PERFORM UNTIL END-OF-FILE  
  IF BOOK-NAME(1:1) = 'A'  
    RELEASE BOOK-RECORD  
  END-IF  
  READ BOOK-FILE  
....  
END-PERFORM.  
CLOSE BOOK-FILE.
```

Paragraph for INPUT PROCEDURE

**NOT** allowed to use PERFORM-UNTIL

Filtering

Read the whole file and select records

# Sorting

- After sorting, it will write the sorted records to a file
  - SORT ... GIVING [output file identifier]
- or, you may alter the structure of sorted records
  - SORT ... OUTPUT PROCEDURE IS [paragraph name]
- For our assignment, you should first generate a temporary sorted output file, then keep processing to generate the summary report.
- **Don't** use OUTPUT PROCEDURE
  - Since you will use 'RETURN...**AT END**...END-RETURN' to retrieve the sorted information

# String handling

- Extract substring from a string with
  - the starting position of the substring
  - the length of the substring
- Syntax: `str([pos] : [len])`
- Example:

```
01 STR PIC X(12) VALUE 'hello world!'.  
01 STR2 PIC X(4).  
...  
    MOVE STR(1:4) TO STR2.  
    DISPLAY STR2.
```

Output:  
hell

# String handling

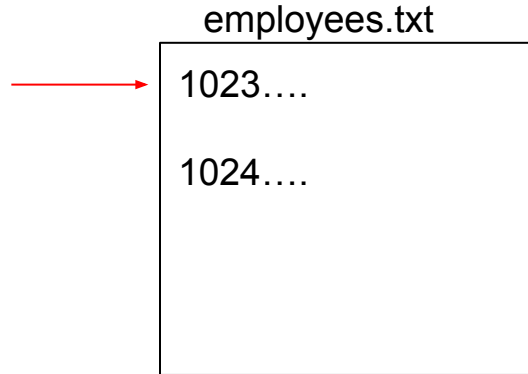
- String concatenation
- Syntax:

```
STRING [string 1] DELIMITED BY [value]  
      [string 2] DELIMITED BY [value]  
      INTO [destination string]  
END-STRING.
```

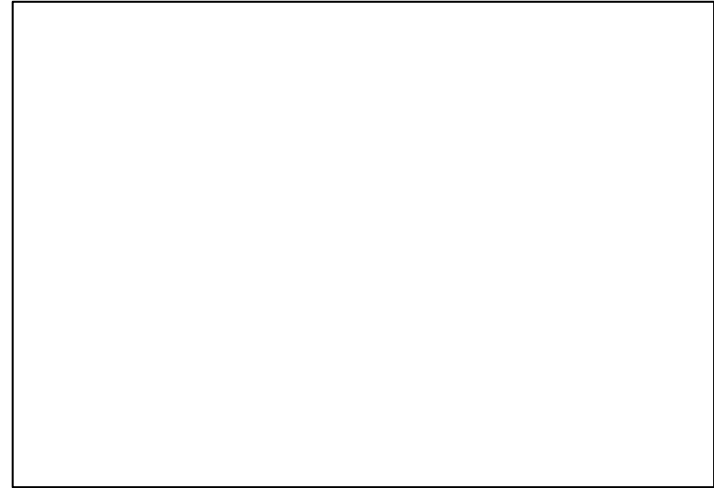
- String splitting: UNSTRING

# Hints on Assignment 1

# How to process attendance records?




`attendance.txt`



1. Read a employee record.


# How to process attendance records?

employees.txt



```
1023....  
1024....
```

attendance.txt

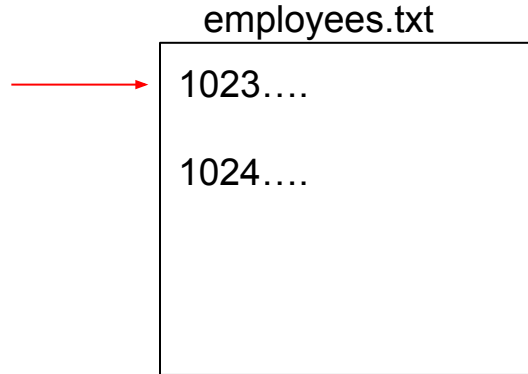


```
1023 .... 2019-01-07-09:28  
1024 .... 2019-01-07-09:57  
....  
1023 .... 2019-01-07-17:29  
1024 .... 2019-01-07-17:33  
....
```

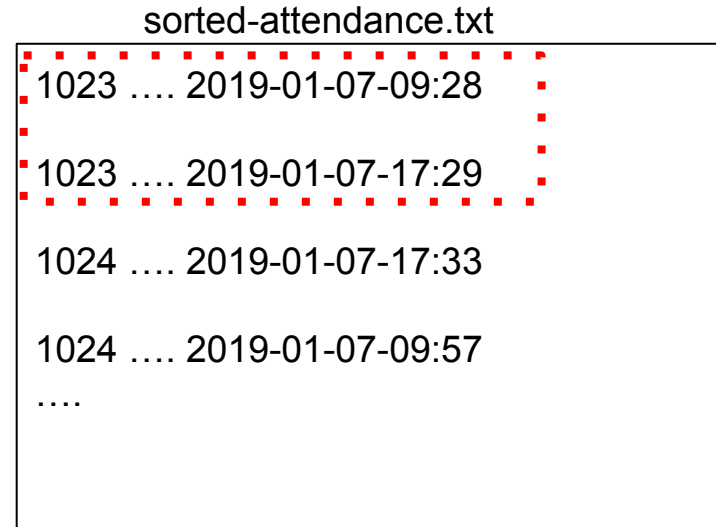
1. Read a employee record.
2. Search for corresponding attendance records.



# How to process attendance records?




1. Read a employee record.
2. Search for corresponding attendance records.



Better to sort it by staff ID,  
instead of timestamp


# How to process attendance records?

employees.txt



```
1023....  
1024....
```

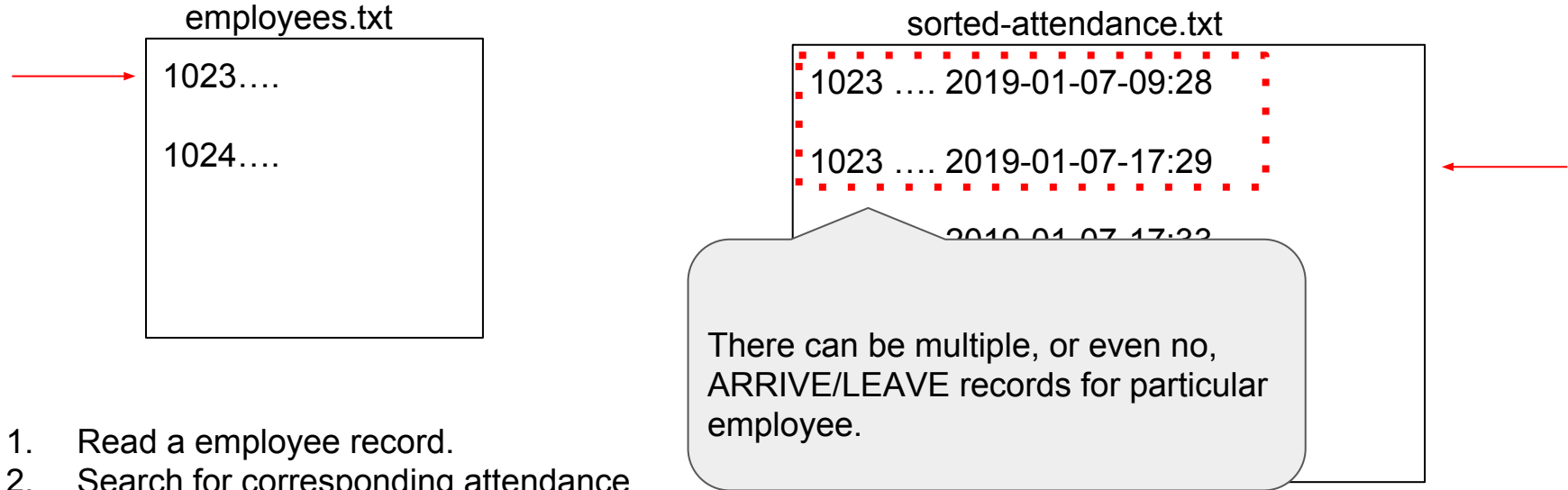
sorted-attendance.txt



```
1023 .... 2019-01-07-09:28  
1023 .... 2019-01-07-17:29  
1024 .... 2019-01-07-17:33  
1024 .... 2019-01-07-09:57  
....
```

1. Read a employee record.
2. Search for corresponding attendance records.
3. Do some calculation and write a line of summary


# How to process attendance records?



1. Read a employee record.
2. Search for corresponding attendance records.
3. Do some calculation and write a line of summary

# How to process attendance records?


employees.txt



```
1023....  
1024....
```

sorted-attendance.txt

```
1023 .... 2019-01-07-09:28  
1023 .... 2019-01-07-17:29  
1024 .... 2019-01-07-17:33  
1024 .... 2019-01-07-09:57  
....
```



1. Read a employee record.
2. Search for corresponding attendance records.
3. Do some calculation and write a line of summary
4. Handle next employee

# How to write a header?

- In FILE SECTION, no variable can be initialized with VALUE clause.
- Reason:
  - Records in the same FD entry share the same memory buffer
  - What if two records want to be initialized?
- Resolve:
  - Define the value in PROCEDURE DIVISION, or,
  - Copy the value in the variable from WORKING-STORAGE SECTION
  - WRITE [record in FD entry] FROM [record in working-storage section]

# No Leading Zero

- Recall: PIC [format]
- Edit the format to display
- Edited items cannot be used as operands in a computation
- Example:

Suppose, format of NUM is 9(5), and value is 01134.

MOVE NUM TO EDIT-NUM.

IF the format of EDIT-NUM is,

- 99,9(3) (value = 01,234)
- +\$\$,999 (value = +\$1,234)

- See more [here](#)

# Line separator

- `\r` : carriage return (CR)
- `\n` : line feed (LF)
- Windows : `\r\n`
- Unix, Mac : `\n`

Example of monthly-attendance.txt:

WRITE in COBOL

- `'\r\n'` is appended
- But, ...

The diagram illustrates the output of a COBOL WRITE statement. It shows four lines of data being written to a file. Each line is represented by a number (1-4) in a grey box, followed by the data string. The data strings are: '2018-09', '10230010000000', '10240000001000', and a blank line. The 'CRLF' characters are shown in black boxes at the end of each line. Red circles highlight the 'CRLF' characters on lines 1, 2, and 3, indicating that they are appended to the end of each line.

```
1 2018-09CRLF
2 10230010000000CRLF
3 10240000001000CRLF
4
```

# Line separator

- GnuCOBOL is run on a Windows simulator.
- Only ‘\n’ is appended for WRITE.
- You need to manually add ‘\r’.

(Google can help you.)



# Learning Resource

- Tutorialspoint
  - <https://www.tutorialspoint.com/cobol/>
- Mainframetechhelp
  - <http://www.mainframetechhelp.com/tutorials/cobol/>
- COBOL course
  - <http://www.csis.ul.ie/cobol/>

Enjoy your work : )