Exercise walk-through

Files ▪ 10.3

eXercise

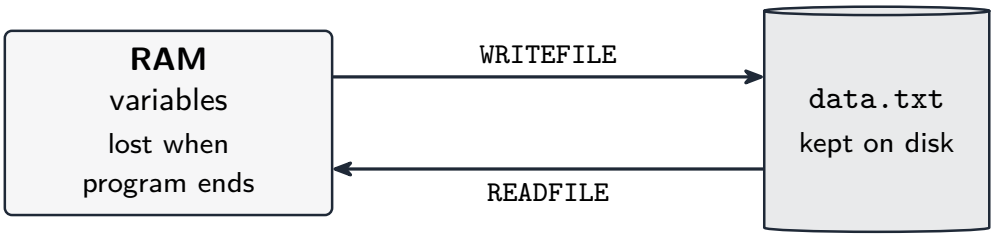
You must be able to

- explain **why** a program uses files
- write pseudocode to **read** a text file line by line using `EOF`
- write pseudocode to **write** or **append** lines to a text file

Method — Why files

Variables live in **RAM** and are lost when the program ends. A **file** on disk is **kept** between runs — so it stores data permanently (high scores, records) and lets programs share data.

Method — Why files



Method — Reading a text file

Open it, read each line until **end of file**, then **close** it:

```
OPENFILE "data.txt" FOR READ
WHILE NOT EOF("data.txt") DO
    READFILE "data.txt", LineString
    OUTPUT LineString
ENDWHILE
CLOSEFILE "data.txt"
```

Method — Reading a text file

`EOF` returns `TRUE` when there is nothing left to read — test it **before** each `READFILE`.

Method — Writing and appending

`FOR WRITE` creates a new file (or **overwrites** an old one); `FOR APPEND` keeps the existing lines and adds after them:

```
OPENFILE "log.txt" FOR WRITE
FOR i ← 1 TO 100
    WRITEFILE "log.txt", "Event " & i
NEXT i
CLOSEFILE "log.txt"
```

Method — Writing and appending

Always **close** a file — otherwise buffered writes can be lost and the file may stay locked.

Practice 2.1 ▪ 1 mark

State **one** reason a program needs to use a file rather than only variables.

Solution 2.1

Method: Why files

Worked steps

Any one: data is **kept** after the program ends; it stores large amounts permanently; it lets programs share data / restart from a saved state B1.

Practice 2.2 ▪ 1 mark

State when the function `EOF` returns `TRUE`.

Solution 2.2

Method: Reading a text file

Worked steps

When the **end of the file** has been reached — there is nothing left to read B1.

Practice 2.3 ▪ 2 marks

Write pseudocode to **open** `"scores.txt"` for reading and then **close** it.

Solution 2.3

> Method: Reading a text file

Worked steps

```
OPENFILE "scores.txt" FOR READ
CLOSEFILE "scores.txt"
```

Solution 2.3

M1 A1 *'OPENFILE ... FOR READ'*, *'CLOSEFILE'*

Practice 2.4 ▪ 1 mark

State **one** reason you must always **close** a file.

Solution 2.4

Method: Writing and appending

Worked steps

Any one: so **buffered writes are saved**; so the file is not left **locked** for other programs B1.

Exam-style 3.1 ▪ 4 marks

Write pseudocode to read **every line** of `"names.txt"` and output each line.

Solution 3.1

> Method: Reading a text file

Worked steps

```
OPENFILE "names.txt" FOR READ
WHILE NOT EOF("names.txt") DO
    READFILE "names.txt", Line
    OUTPUT Line
ENDWHILE
CLOSEFILE "names.txt"
```

Solution 3.1

M1 M1 M1 A1 *'OPENFILE ... FOR READ'*, *'WHILE NOT EOF'*, *'READFILE'* + *'OUTPUT'*, *'CLOSEFILE'*

Exam-style 3.2 ▪ 3 marks

Write pseudocode to write the numbers **1 to 50**, one per line, to a new file `"nums.txt"`.

Solution 3.2

Worked steps

```
OPENFILE "nums.txt" FOR WRITE
FOR i ← 1 TO 50
    WRITEFILE "nums.txt", i
NEXT i
CLOSEFILE "nums.txt"
```

Solution 3.2

M1 M1 A1 *'OPENFILE ... FOR WRITE', loop 1 to 50 with 'WRITEFILE', 'CLOSEFILE'*

Exam-style 3.3 ▪ 4 marks

A file `"members.txt"` holds one name per line. Write pseudocode to **count** how many names are in the file and output the total.

Solution 3.3

> Method: Reading a text file

Worked steps

```
Count ← 0
OPENFILE "members.txt" FOR READ
WHILE NOT EOF("members.txt") DO
    READFILE "members.txt", Name
    Count ← Count + 1
ENDWHILE
CLOSEFILE "members.txt"
OUTPUT Count
```

Solution 3.3

M1 M1 M1 A1 *'Count ← 0'* + *'OPENFILE'*, *'WHILE NOT EOF'* + *'READFILE'*, *'Count ← Count + 1'*, *'CLOSEFILE'* + *'OUTPUT Count'*

Exam-style 3.5 ▪ 3 marks

A sports club stores one line per member in a text file `members.txt`. Each line holds the membership number, the surname and the class of membership, separated by the character `|`.

```
40118|Chen|Full
40119|Osei|Junior
```

(a) **Explain** why a **separator** character is needed, and **state** one property the chosen character must have.

Exam-style 3.5 ▪ 3 marks

(b) **Write** pseudocode for a procedure `CountClass(Wanted)` that reads the whole file and outputs how many members are in the class given by `Wanted`.
(c) The club adds two new items for each member: a phone number and an emergency contact, **both encrypted**. **Explain** one problem this could cause for the file format, and **describe** how you would solve it.
(d) **Compare** storing this data in a **serial** file with storing it in a **sequential** file ordered by membership number, for the task of finding one member.

Solution 3.5

Method: Writing and appending

Worked steps

(a) The three items are of different lengths, so without a marker the program cannot tell where one field ends and the next begins B1 B1. The separator must be a character that can **never appear inside the data itself** — a surname could contain a space or a hyphen, so | is safer than either B1.

(b)

```
PROCEDURE CountClass(Wanted : STRING)
    DECLARE Line, ThisClass : STRING
    DECLARE Count : INTEGER
    Count ← 0
    OPENFILE "members.txt" FOR READ
    WHILE NOT EOF("members.txt")
        READFILE "members.txt", Line
        ThisClass ← the text after the second "|" in Line
        IF ThisClass = Wanted THEN Count ← Count + 1 ENDIF
    ENDWHILE
    CLOSEFILE "members.txt"
```

Solution 3.5

B1 B1 B1 B1 B1 B1 *header with parameter; counter initialised; 'OPENFILE … FOR READ'; 'WHILE NOT EOF' with 'READFILE'; the field extracted and compared; 'CLOSEFILE' and output*

(c) Encrypted data is arbitrary bytes, so it may **contain the separator character itself**, which would split one field into two and corrupt every field after it on that line B1 B1. Solve it by encoding the encrypted values in a form that cannot contain | — for example converting them to hexadecimal or Base64 before writing — or by using fixed-length fields instead of a separator B1.

(d) In a **serial** file the records are in the order they were added, so a search must read from the start until it finds the member: on average half the file B1. A **sequential** file is ordered by membership number, so the search can stop as soon as it passes the wanted key, and the file can be searched far faster B1. The cost is that inserting a new member into a sequential file means rewriting the file to keep the order, whereas a serial file just appends B1.