Exercise walk-through

10.3 Files

A-Level Computer Science

Sheet 10.3 – what you need to know

You must be able to:

- open, read, write, append and close a text file
- read to the end of a file with EOF
- design a line format with separators, and convert types

What the exam asks

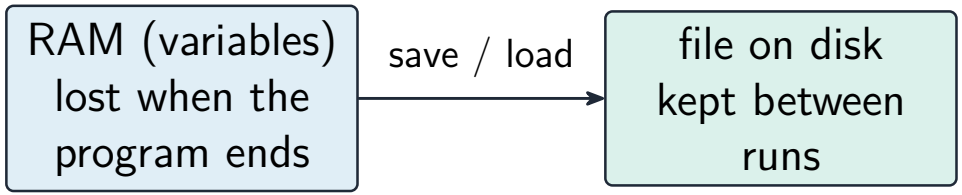
"Complete the following numbered steps: Step 1: open the file ... in ..." **[6]**

"Write pseudocode for module AddNewCustomers()." **[8]**

"Explain the additional file design changes needed to extract each item after the line is read." **[3]**

Practise on exercise sheet 10.3, then try the real exam questions on past-paper sheet 10.3.

Files



A **file** keeps data on disk between runs (RAM is lost at exit).

- `OPENFILE` for READ/WRITE/APPEND
- `EOF` tests the **end of file**
- always `CLOSEFILE` – or buffered writes are lost

Reading and appending a text file

```
OPENFILE "Stock.txt" FOR READ
Count ← 0
WHILE NOT EOF("Stock.txt")
    READFILE "Stock.txt", Line
    Count ← Count + 1
ENDWHILE
CLOSEFILE "Stock.txt"

OPENFILE "Log.txt" FOR APPEND
Line ← NUM_TO_STR(ID) & "," & Name
WRITEFILE "Log.txt", Line
CLOSEFILE "Log.txt"
```

- WRITE replaces the file; APPEND adds to its end
- a file holds strings: NUM_TO_STR before writing, STR_TO_NUM after reading
- fields on one line need a separator that cannot occur in the data; for encrypted text, store each field's length first
- serial: in the order added; sequential: in key order

Exercises 2.1·2.2

2.1 State **one** reason a program needs to use a file rather than only variables. [1]

variables file

2.2 State when the function `EOF` returns `TRUE`. [1]

———

Exercises 2.1·2.2

2.1 State **one** reason a program needs to use a file rather than only variables. [1]

variables file

2.1

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

2.2 State when the function `EOF` returns `TRUE`. [1]

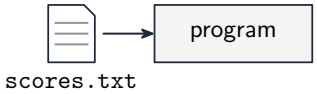
———

2.2

- when the **end of the file** has been reached – there is nothing left to read

Exercises 2.3·2.4

2.3 Write pseudocode to **open** `"scores.txt"` for reading and then **close** it. [2]

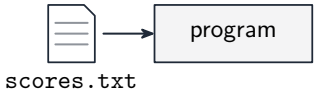


2.4 State **one** reason you must always **close** a file. [1]

open file

Exercises 2.3·2.4

2.3 Write pseudocode to **open** `"scores.txt"` for reading and then **close** it. [2]



2.3

- open for read, then close

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

2.4 State **one** reason you must always **close** a file. [1]

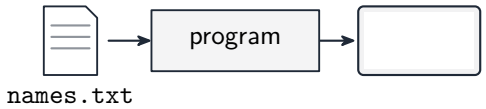
open file

2.4

- any one: so **buffered writes are saved**; so the file is not left **locked** for other programs

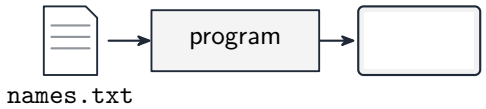
Exercise 3.1

3.1 Write pseudocode to read **every line** of `"names.txt"` and output each line. [4]



Exercise 3.1

3.1 Write pseudocode to read **every line** of `"names.txt"` and output each line. [4]



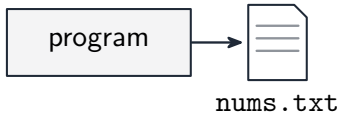
3.1

- open, loop until EOF, read and output, then close

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

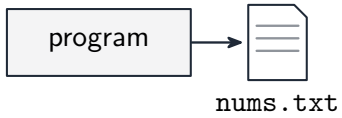
Exercise 3.2

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



Exercise 3.2

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



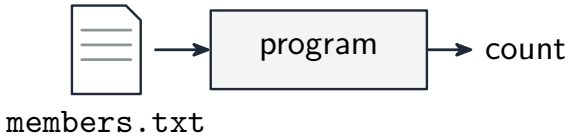
3.2

- open for write, loop 1 to 50, then close

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

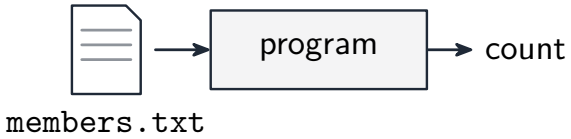
Exercise 3.3

3.3 A file `"members.txt"` holds one name per line. Write pseudocode to **count** how many names are in the file and output the total. [4]



Exercise 3.3

3.3 A file `"members.txt"` holds one name per line. Write pseudocode to **count** how many names are in the file and output the total. [4]



3.3

- start `Count` at 0, then read every record

Full solution on next slide

Full solution 3.3

3.3

- start `Count` at 0, then read every record

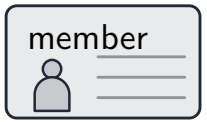
```
Count ← 0
OPENFILE "members.txt" FOR READ
WHILE NOT EOF("members.txt") DO
    READFILE "members.txt", Name
    Count ← Count + 1
ENDWHILE
CLOSEFILE "members.txt"
OUTPUT Count
```

- increment inside the loop; close and output

Exercise 3.4 (a)

3.4 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
```

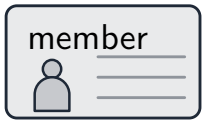


(a) **Explain** why a **separator** character is needed, and **state** one property the chosen character must have. [3]

Exercise 3.4 (a)

3.4 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. [3]

(a) the three items are of different lengths, so without a marker the program cannot tell where one field ends and the next begins

- 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

Exercise 3.4 (b)

3.4 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
```



(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`. [6]

Exercise 3.4 (b)

3.4 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
```



(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`. [6]

(b)

```
PROCEDURE CountClass(Wanted : STRING)
    DECLARE Line, ThisClass : STRING
    DECLARE Count : INTEGER
    Count ← 0
    OPENFILE "members.txt" FOR READ
```

Full solution on next slide

Full solution 3.4 (b)

(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"
    OUTPUT Count
ENDPROCEDURE
```

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

Exercise 3.4 (c)

3.4 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
```



(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. [3]

Exercise 3.4 (c)

3.4 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
```



(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. [3]

(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

- encode the encrypted values in a form that cannot contain `|` (hex or Base64), or use fixed-length fields instead of a separator

Exercise 3.4 (d)

3.4 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
```



(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. [3]

Exercise 3.4 (d)

3.4 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
```



(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. [3]

(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

- 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

- 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