

10.3 Files

Name: _____ Class: _____ Date: _____

Total: 31 marks**KEY WORDS** file 文件 • text file 文本文件 • end of file 文件结束 • record 记录

Objective

Build the skills to answer exam questions on **text files** 文本文件 — why files are needed, and reading and writing them in pseudocode.

You must be able to:

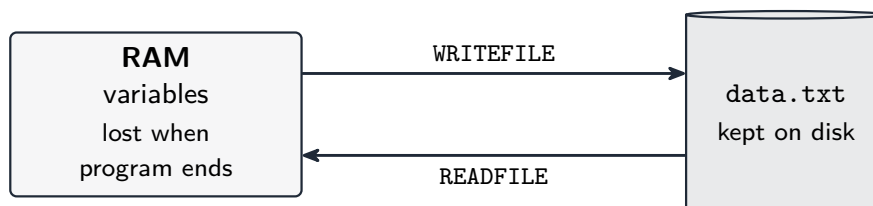
- explain **why** a program uses files
- write pseudocode to **read** a text file line by line using **EOF**
- write pseudocode to **write** or **append** lines to a text file

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ 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.



■ 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"
```

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

■ 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"
```

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

2 Practice

Now apply the methods above.

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

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

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]

3 Exam-style questions

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

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

3.5 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]

(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]

(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]

(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]

Solutions

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

2.3

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

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

3.1

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

3.2

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

3.3

```
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
```

3.5 (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.

(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

```

(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. 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.

(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.