Text files

Ref As Pseudocode

Opening and closing a file

Variables lose their values when a program ends. A **text file** keeps data on storage, so the next run can read it again. A text file is a list of **lines**, and each line is a string. Lines are read or written **in order**, one after another.

OPENFILE

Before you use a file, you must open it in one **file mode** :

```
OPENFILE <file identifier> FOR <file mode>
```

Mode	What it does
`READ`	read lines from the file, starting with the first line
`WRITE`	create a **new**, empty file and write to it; any old data in the file is **lost**
`APPEND`	add lines to the **end** of the file; the old data is kept

The file identifier is the file's name, as a string such as `"Scores.txt"`, or a `STRING` variable that holds the name.

CLOSEFILE

```
CLOSEFILE <file identifier>
```

Close every file when you have finished with it. A file is open in **one mode at a time**: to read a file you have just written, close it and open it again `FOR READ`.

```
DECLARE FileName : STRING
FileName ← "Log.txt"
OPENFILE FileName FOR WRITE
WRITEFILE FileName, "Program started"
CLOSEFILE FileName
OUTPUT "Written to ", FileName
```

Common mistakes

- Using a file without opening it first.
- Forgetting `CLOSEFILE`, often worth a mark in Paper 2.
- Opening with `WRITE` when you wanted to keep the old lines. `WRITE` empties the file; use `APPEND`.

Writing and appending

`WRITEFILE` writes **one line** to a file that is open for `WRITE` or `APPEND`.

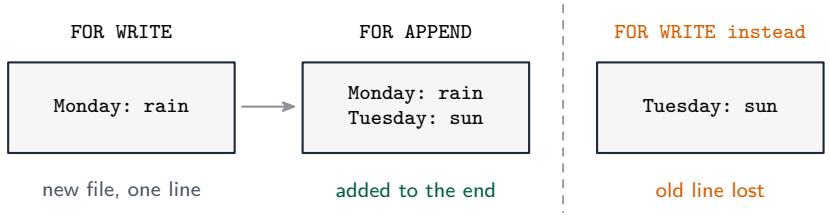
```
WRITEFILE <file identifier>, <data>
```

```
DECLARE Line : STRING
OPENFILE "Diary.txt" FOR WRITE
WRITEFILE "Diary.txt", "Monday: rain"
CLOSEFILE "Diary.txt"

OPENFILE "Diary.txt" FOR APPEND
WRITEFILE "Diary.txt", "Tuesday: sun"
CLOSEFILE "Diary.txt"

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

Both lines are printed, because `APPEND` kept the first one. Change `APPEND` to `WRITE` and run it again: only Tuesday is left.



WRITE starts an empty file; APPEND adds to the end

exercises.lol exercises.com.cn

Several values on one line

A line is one string. To store several values, join them into one string with a separator separator , such as a comma:

```
DECLARE Name : STRING
DECLARE Score : INTEGER
Name ← "Ravi"
Score ← 87
OPENFILE "Results.txt" FOR WRITE
WRITEFILE "Results.txt", Name & "," & NUM_TO_STR(Score)
CLOSEFILE "Results.txt"
OUTPUT "Saved: ", Name & "," & NUM_TO_STR(Score)
```

Saving a whole array

Write one element per line inside a loop:

```
DECLARE Colours : ARRAY[1:3] OF STRING
DECLARE i : INTEGER
Colours[1] ← "red"
Colours[2] ← "green"
Colours[3] ← "blue"
OPENFILE "Colours.txt" FOR WRITE
FOR i ← 1 TO 3
   WRITEFILE "Colours.txt", Colours[i]
NEXT i
CLOSEFILE "Colours.txt"
OUTPUT "3 lines written"
```

Keep `OPENFILE` and `CLOSEFILE` **outside** the loop. Opening a file `FOR WRITE` inside the loop would empty it on every pass.

Reading to the end of a file

`READFILE` reads the **next line** of a file that is open `FOR READ`, and stores it in a variable. The variable should be a `STRING`.

```
READFILE <file identifier>, <variable>
```

The function `EOF` tells you when there is nothing left to read. `EOF` stands for **end of file** :

```
EOF(<file identifier>)
```

`EOF` returns `TRUE` when there are no more lines (also when the file is empty), and `FALSE` otherwise.

exercises.lol exercises.com.cn

The standard reading loop

You usually do not know how many lines a file has, and it may be empty. So use a `WHILE` loop that tests `EOF` **before** each read:

```
DECLARE Line : STRING
DECLARE Count : INTEGER
OPENFILE "Names.txt" FOR WRITE
WRITEFILE "Names.txt", "Amir"
WRITEFILE "Names.txt", ""
WRITEFILE "Names.txt", "Bea"
CLOSEFILE "Names.txt"

Count ← 0
OPENFILE "Names.txt" FOR READ
WHILE NOT EOF("Names.txt")
   READFILE "Names.txt", Line
   IF Line <> "" THEN
      Count ← Count + 1
      OUTPUT Line
   ENDIF
ENDWHILE
CLOSEFILE "Names.txt"
OUTPUT Count, " names read"
```

The first part only creates the file, so that the example can run. The second part is the pattern to learn: **open, loop while not EOF, read, process, close**.

Copying one file to another

This is the guide's own example: copy every line, but replace blank lines with dashes.

```
DECLARE LineOfText : STRING
OPENFILE "FileA.txt" FOR READ
OPENFILE "FileB.txt" FOR WRITE
WHILE NOT EOF("FileA.txt")
   READFILE "FileA.txt", LineOfText
   IF LineOfText = "" THEN
      WRITEFILE "FileB.txt", "-------------"
   ELSE
      WRITEFILE "FileB.txt", LineOfText
   ENDIF
ENDWHILE
CLOSEFILE "FileA.txt"
CLOSEFILE "FileB.txt"
```

Two files can be open at the same time, each in its own mode.

Practice

1. A file `Marks.txt` has one whole-number mark per line. Write pseudocode that reads it and outputs the average mark.

<details> <summary>Model answer</summary>

```
DECLARE Line : STRING
DECLARE Total : INTEGER
DECLARE Count : INTEGER
// Create a sample file so that the program can run
OPENFILE "Marks.txt" FOR WRITE
WRITEFILE "Marks.txt", "60"
WRITEFILE "Marks.txt", "75"
WRITEFILE "Marks.txt", "90"
CLOSEFILE "Marks.txt"

Total ← 0
Count ← 0
OPENFILE "Marks.txt" FOR READ
WHILE NOT EOF("Marks.txt")
   READFILE "Marks.txt", Line
   Total ← Total + STR_TO_NUM(Line)
   Count ← Count + 1
ENDWHILE
CLOSEFILE "Marks.txt"
IF Count > 0 THEN
   OUTPUT "Average: ", Total / Count
ELSE
   OUTPUT "The file is empty"
ENDIF
```

Each line is a string, so `STR_TO_NUM` turns it into a number. The `IF Count > 0` stops a division by zero when the file is empty.

</details>

Common mistakes

- Using `REPEAT … UNTIL EOF(…)`. It reads once **before** testing, so it fails on an empty file.
- Using a `FOR` loop with a guessed number of lines.
- Doing arithmetic on the line you read without converting it first.
- Reading a file that is open `FOR WRITE`, or writing to one that is open `FOR READ`.

exercises.lol exercises.com.cn