Exercise walk-through

File Processing and Exception Handling ▪ 20.2

eXercise

You must be able to

- write pseudocode for **file** read, write, append and search
- explain what an **exception** is and why **handling** it matters
- write a `TRY ... EXCEPT` block and **raise** an exception

Method — File processing

`OPENFILE … FOR READ | WRITE | APPEND` (WRITE overwrites; APPEND adds to the end), `READFILE`, `WRITEFILE`, `CLOSEFILE`, `EOF`. Search a file, stopping when found:

```
found ← FALSE
OPENFILE "people.txt" FOR READ
WHILE NOT EOF("people.txt") AND NOT found DO
    READFILE "people.txt", Line
    IF Line = Target THEN found ← TRUE
ENDWHILE
CLOSEFILE "people.txt"
```

Method — File processing

To **edit in place**, copy to a temp file (changing the right lines), then replace the original.

Random (direct-access) files store fixed-size **records** you can jump straight to by position: `OPENFILE … FOR RANDOM`, then `SEEK` to a record number and `GETRECORD` / `PUTRECORD` to read or write that record — without reading the whole file.

Method — Exception handling

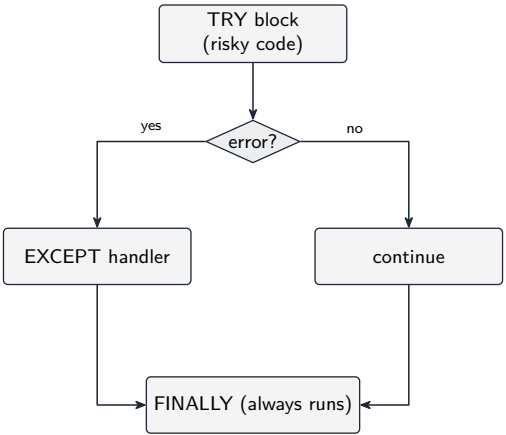
An **exception** is an error during execution (file not found, divide by zero). Handling it lets the program respond **gracefully** instead of crashing:

```
TRY
    OPENFILE "data.txt" FOR READ
    READFILE "data.txt", Line
    CLOSEFILE "data.txt"
EXCEPT FileNotFound
    OUTPUT "Sorry, the file does not exist."
ENDTRY
```

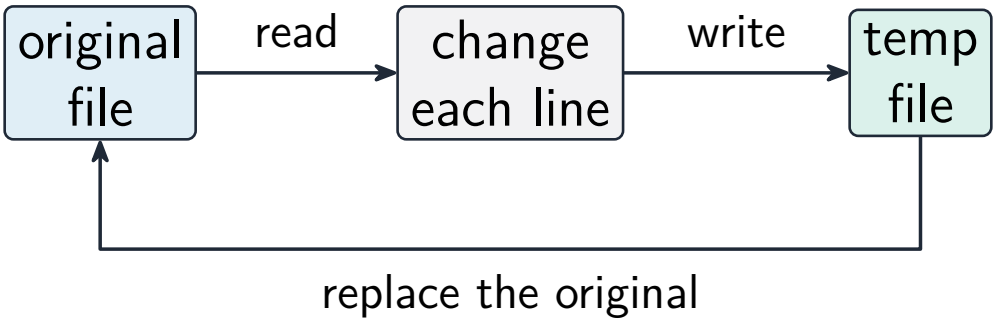
Method — Exception handling

A subroutine can **raise** an exception for the caller to handle: `IF b = 0 THEN RAISE DivideByZero`.

Method — Exception handling



Method — Exception handling



Updating a file record with exception handling

Practice 2.1 ▪ 1 mark

State what `OPENFILE … FOR APPEND` does that `FOR WRITE` does not.

Solution 2.1

Method: File processing

Worked steps

`APPEND` **keeps** the existing contents and adds after them; `WRITE` **overwrites** (erases) the file B1.

Practice 2.2 ▪ 1 mark

State what an **exception** is.

Solution 2.2

Worked steps

An **error or unexpected condition** that occurs **during execution** (e.g. file not found, divide by zero) B1.

Practice 2.3 ▪ 1 mark

State the purpose of a `FINALLY` block.

Solution 2.3

Method: Exception handling

Worked steps

Code that **always runs** (whether or not an exception happened) — used for **cleanup** such as closing files B1.

Practice 2.4 ▪ 1 mark

State **one** pitfall of file processing.

Solution 2.4

Method: File processing

Worked steps

Any one: forgetting to **close** a file (data lost); opening `FOR WRITE` instead of `APPEND` (overwrites); reading past `EOF`; hard-coded paths B1.

Exam-style 3.1 ▪ 4 marks

Write pseudocode that searches `"people.txt"` for a `Target` name, **stopping** as soon as it is found, and outputs whether it was found.

Solution 3.1

Method: File processing

Worked steps

```
found ← FALSE
OPENFILE "people.txt" FOR READ
WHILE NOT EOF("people.txt") AND NOT found DO
    READFILE "people.txt", Line
    IF Line = Target THEN found ← TRUE
ENDWHILE
CLOSEFILE "people.txt"
IF found THEN OUTPUT "Found" ELSE OUTPUT "Not found"
```

Solution 3.1

M1 M1 M1 A1 *OPEN + loop, 'WHILE NOT EOF ... AND NOT found', compare + set flag, CLOSE + report*

Exam-style 3.2 ▪ 3 marks

Write a `TRY … EXCEPT` block that opens and reads `"data.txt"` and handles the case where the **file does not exist**.

Solution 3.2

> Method: Exception handling

Worked steps

```
TRY
    OPENFILE "data.txt" FOR READ
    READFILE "data.txt", Line
    CLOSEFILE "data.txt"
EXCEPT FileNotFound
    OUTPUT "Sorry, the file does not exist."
ENDTRY
```

Solution 3.2

M1 M1 A1 *'TRY ... ENDTRY', file read inside TRY, 'EXCEPT' handler with message*

Exam-style 3.3 ▪ 2 marks

Explain **why** exception handling is important in real programs.

Solution 3.3

Method: Exception handling

Worked steps

Programs face errors that **cannot be prevented up front** (missing files, bad input) B1; handling them lets the program **recover/inform the user** instead of **crashing**, and keeps the normal code clean B1.

Exam-style 3.4 ▪ 2 marks

Write pseudocode for a function `Divide(a, b)` that **raises** an exception when `b` is `0`, otherwise returns `a DIV b`.

Solution 3.4

Method: Exception handling

Worked steps

```
FUNCTION Divide(a : INTEGER, b : INTEGER) RETURNS INTEGER
    IF b = 0 THEN
        RAISE DivideByZero
    ENDIF
    RETURN a DIV b
ENDFUNCTION
```

Solution 3.4

M1 A1 *test ‘b = 0‘ + ‘RAISE‘, ‘RETURN a DIV b‘ otherwise*

Exam-style 3.5 ▪ 2 marks

A file of fixed-length records is opened `FOR RANDOM`. State what `SEEK` and `PUTRECORD` are each used for.

Solution 3.5

Method: File processing

Worked steps

`SEEK` moves to a given **record number / position** in the file B1; `PUTRECORD` **writes a record** at that position B1.