

Exercise walk-through

11.3

Structured Programming

A-Level Computer Science

Procedures

Structured programming builds a program from small named **subroutines**, each with one job.

```
PROCEDURE Greet(Name : STRING)
    OUTPUT "Hello, ", Name
ENDPROCEDURE

CALL Greet("Ada")
```

A **procedure** is a named block that **does an action**. It may take **parameters**, but it does **not return a value**.

CALL Greet("Ada")



prints Hello, Ada

returns no value

Worked example: IsValid, each character

```
FOR Index ← 1 TO 6
  Ch ← MID(Code, Index, 1)
  IF Index ≤ 2 THEN
    IF Ch < "A" OR Ch > "Z" THEN
      RETURN FALSE
    ENDIF
  ELSE
    IF Ch < "0" OR Ch > "9" THEN
      RETURN FALSE
    ENDIF
  ENDIF
NEXT Index
RETURN TRUE
ENDFUNCTION
```

Validation like this
returns a BOOLEAN, so the
caller decides what to do:

```
IF IsValid(Entry) THEN
  CALL Save(Entry)
ELSE
  OUTPUT "Invalid"
ENDIF
```

The **caller** outputs the
message: a function
calculates, a procedure acts.

From design to code

Turn a design, a **flowchart** (a **program flowchart**) or **structured English**, into **pseudocode** and then into a real language:

1 find the **variables** and their **data types**

2 input/output boxes become **INPUT** / **OUTPUT**

3 diamonds become **IF ... ENDIF** or **CASE**

4 loop arrows become **WHILE**, **REPEAT** or **FOR**

5 process boxes become assignments

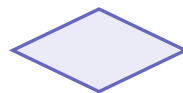
6 trace a small input to check

flowchart

pseudocode



INPUT / **OUTPUT**



IF ... THEN or **CASE**



$x \leftarrow \text{expression}$



WHILE / **FOR** / **REPEAT**

Exercise 2.1

2.1 Complete the sentences. Use words from the list. Each word is used once. [4]

```
PROCEDURE Greet(Name : STRING)
```

```
CALL Greet("Ada")
```

argument · called · defined · parameter

A procedure is once, and it can then be many times. A variable in the header that receives a value is a The value that the call gives to it is an

Exercise 2.1

2.1 Complete the sentences. Use words from the list. Each word is used once. [4]

```
PROCEDURE Greet(Name : STRING)
```

```
CALL Greet("Ada")
```

argument · called · defined · parameter

A procedure is once, and it can then be many times. A variable in the header that receives a value is a The value that the call gives to it is an

2.1

- a procedure is **defined** once and **called** many times
- the variable in the header is a **parameter**, and the value given in the call is an **argument**

Exercise 2.2

2.2 Study this pseudocode.

```
PROCEDURE Welcome()  
    OUTPUT "Welcome"  
    OUTPUT "Please log in"  
ENDPROCEDURE  
  
OUTPUT "Start"  
CALL Welcome()  
OUTPUT "End"
```



State everything that is output, in the correct order.

[2]

Exercise 2.2

2.2 Study this pseudocode.

```
PROCEDURE Welcome()  
    OUTPUT "Welcome"  
    OUTPUT "Please log in"  
ENDPROCEDURE  
  
OUTPUT "Start"  
CALL Welcome()  
OUTPUT "End"
```



State everything that is output, in the correct order.

[2]

2.2

- the program starts at the first statement outside the definition, so **"Start"** comes first
- the call runs both lines of the procedure, and the program then carries on: **Start, Welcome, Please log in, End**

Exercise 2.3

2.3 Define a procedure `Greet(Name)` that outputs "Hello, " followed by the name. [2]



Exercise 2.3

2.3 Define a procedure `Greet(Name)` that outputs "Hello, " followed by the name. [2]



2.3

- PROCEDURE ... ENDPROCEDURE with a parameter

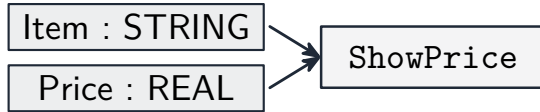
```
PROCEDURE Greet(Name : STRING)
    OUTPUT "Hello, ", Name
ENDPROCEDURE
```

Exercise 2.4 (a)

2.4 A procedure has the header `PROCEDURE ShowPrice(Item : STRING, Price : REAL)`

State whether each call is correct. If it is not correct, give the reason. [3]

(a) `CALL ShowPrice("Pen", 1.20)`



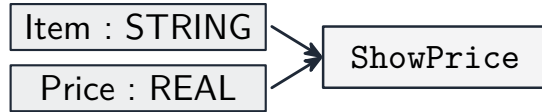
Exercise 2.4 (a)

2.4 A procedure has the header `PROCEDURE ShowPrice(Item : STRING, Price : REAL)`

State whether each call is correct. If it is not correct, give the reason.

[3]

(a) `CALL ShowPrice("Pen", 1.20)`



(a) **correct**: a string and a real number, in the order of the header

Exercise 2.4 (b)

2.4 A procedure has the header `PROCEDURE ShowPrice(Item : STRING, Price : REAL)`

State whether each call is correct. If it is not correct, give the reason.

[3]

(b) `CALL ShowPrice(1.20, "Pen")`

Exercise 2.4 (b)

2.4 A procedure has the header `PROCEDURE ShowPrice(Item : STRING, Price : REAL)`

State whether each call is correct. If it is not correct, give the reason. [3]

(b) `CALL ShowPrice(1.20, "Pen")`

(b) **not correct**: the arguments are in the wrong order, so a number would go into the `STRING` parameter `Item`

Exercise 2.4 (c)

2.4 A procedure has the header `PROCEDURE ShowPrice(Item : STRING, Price : REAL)`

State whether each call is correct. If it is not correct, give the reason. [3]

(c) `CALL ShowPrice("Pen")`

Exercise 2.4 (c)

2.4 A procedure has the header `PROCEDURE ShowPrice(Item : STRING, Price : REAL)`

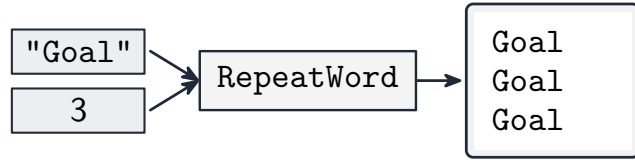
State whether each call is correct. If it is not correct, give the reason. [3]

(c) `CALL ShowPrice("Pen")`

(c) **not correct:** the header has two parameters, but the call gives only one argument

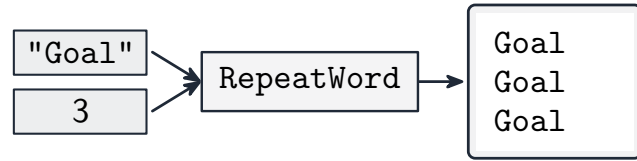
Exercise 2.5

2.5 Write a procedure `RepeatWord` that takes a string `Word` and an integer `Times`, and outputs the word that many times, each on a new line. Then write the call that outputs "Goal" 3 times. [4]



Exercise 2.5

2.5 Write a procedure `RepeatWord` that takes a string `Word` and an integer `Times`, and outputs the word that many times, each on a new line. Then write the call that outputs "Goal" 3 times. [4]



2.5

- a header with the two parameters and their types

Full solution 2.5

2.5

- a header with the two parameters and their types
- a FOR loop from 1 to Times, with a local counter
- OUTPUT Word inside the loop, and ENDPROCEDURE
- the call: CALL RepeatWord("Goal", 3)

```
PROCEDURE RepeatWord(Word : STRING, Times : INTEGER)
    DECLARE Count : INTEGER
    FOR Count ← 1 TO Times
        OUTPUT Word
    NEXT Count
ENDPROCEDURE

CALL RepeatWord("Goal", 3)
```

Exercise 2.6

2.6 State **two** benefits of using procedures in a program.

[2]

```
PROCEDURE Show()  
    _____
```

```
ENDPROCEDURE
```

Exercise 2.6

2.6 State **two** benefits of using procedures in a program.

[2]

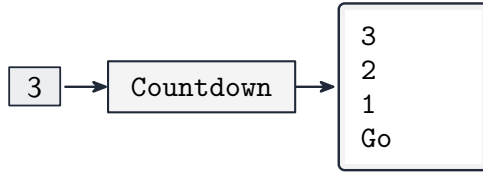
```
PROCEDURE Show()  
    _____  
ENDPROCEDURE
```

2.6 any **two** of:

- the statements are written once and called many times, so the program is shorter
- a change is made in one place only
- a procedure can be tested on its own
- the program is easier to read

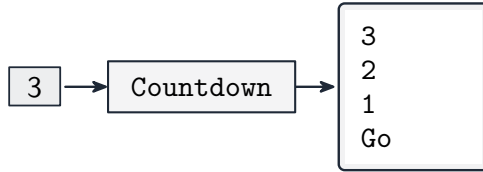
Exercise 3.1

3.1 Write a **procedure** `Countdown()` that takes a positive integer `N` as a parameter, outputs the numbers from `N` down to 1 on separate lines, and then outputs "Go". [5]



Exercise 3.1

3.1 Write a **procedure** `Countdown()` that takes a positive integer `N` as a parameter, outputs the numbers from `N` down to 1 on separate lines, and then outputs "Go". [5]



3.1

- PROCEDURE header with the parameter, and ENDPROCEDURE

Full solution on next slide

Full solution 3.1

3.1

- PROCEDURE header with the parameter, and ENDPROCEDURE
- a loop that counts **down** from N to 1
- output each number inside the loop
- output "Go" once, **after** the loop

```
PROCEDURE Countdown(N : INTEGER)
  DECLARE Count : INTEGER
  FOR Count ← N TO 1 STEP -1
    OUTPUT Count
  NEXT Count
  OUTPUT "Go"
ENDPROCEDURE
```

- a WHILE loop that decreases a copy of N earns the same marks

Setup for exercise 3.2

3.2 Study this pseudocode.

```
PROCEDURE Box(Width : INTEGER, Symbol : CHAR)
    DECLARE Count : INTEGER
    DECLARE Line : STRING
    Line ← ""
    FOR Count ← 1 TO Width
        Line ← Line & Symbol
    NEXT Count
    OUTPUT Line
ENDPROCEDURE

CALL Box(4, '#')
CALL Box(2, '+')
```

Exercise 3.2 (a)

(a) State what is output.

[2]



Exercise 3.2 (a)

(a) State what is output.

[2]



(a) the first call joins four # characters, and the second call joins two + characters

- the output is #### and then ++

Exercise 3.2 (b)

(b) Identify the **parameters** of the procedure, and the **arguments** of the first call. [2]

Exercise 3.2 (b)

(b) Identify the **parameters** of the procedure, and the **arguments** of the first call. [2]

(b) the parameters are `Width` and `Symbol`

- the arguments of the first call are `4` and `'#'`

Exercise 3.2 (c)

(c) State the name of the type of variable that `Count` and `Line` are, and state when they exist. [2]

Exercise 3.2 (c)

(c) State the name of the type of variable that **Count** and **Line** are, and state when they exist. [2]

(c) they are **local variables**

- they exist only while the procedure is running

Exercise 3.3 (a)

3.3 A program for a game outputs the same three lines of instructions at the start of the game, after each level, and when the player asks for help.

(a) **Explain** why a procedure is appropriate here.

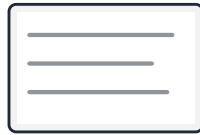
[2]



start



after a level



help

Exercise 3.3 (a)

3.3 A program for a game outputs the same three lines of instructions at the start of the game, after each level, and when the player asks for help.

(a) **Explain** why a procedure is appropriate here.

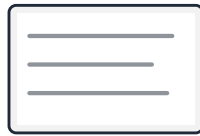
[2]



start



after a level



help

(a) the same group of statements is needed at three different points of the algorithm

- the task only outputs text and gives no value back, so a procedure that is written once and called three times is appropriate

Exercise 3.3 (b)

3.3 A program for a game outputs the same three lines of instructions at the start of the game, after each level, and when the player asks for help.

(b) The instructions are changed in a new version of the game. **Explain** the benefit of having used a procedure. [2]

Exercise 3.3 (b)

3.3 A program for a game outputs the same three lines of instructions at the start of the game, after each level, and when the player asks for help.

(b) The instructions are changed in a new version of the game. **Explain** the benefit of having used a procedure. [2]

(b) the instructions are written in one place only, inside the procedure

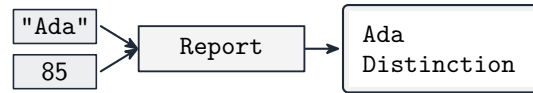
- so one change updates all three places, and no copy can be forgotten or changed differently

Setup for exercise 3.4

3.4 A teacher wants a program that outputs a report for each student: the student's name, then a line of text that depends on the mark ("**Distinction**" for 80 or more, "**Pass**" for 50 or more, otherwise "**Not yet**").

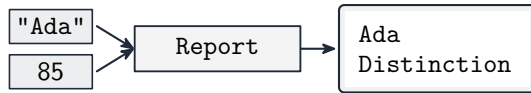
Exercise 3.4 (a)

- (a) Write a procedure **Report** that takes a student's name and mark as parameters, and outputs the two lines. [6]



Exercise 3.4 (a)

- (a) Write a procedure **Report** that takes a student's name and mark as parameters, and outputs the two lines. [6]



A large, empty rectangular box with a thin black border, intended for the student to write the code for the **Report** procedure.

- (a) a header with two parameters and their types
- the name is output

Full solution 3.4 (a)

(a) a header with two parameters and their types

- the name is output
- a first test for 80 or more
- a nested IF for 50 or more, with the ELSE for "Not yet"
- the three messages in the correct paths
- every construct closed, and ENDPROCEDURE

```
PROCEDURE Report(Name : STRING, Mark : INTEGER)
  OUTPUT Name
  IF Mark >= 80 THEN
    OUTPUT "Distinction"
  ELSE
    IF Mark >= 50 THEN
      OUTPUT "Pass"
    ELSE
      OUTPUT "Not yet"
    ENDIF
  ENDIF
ENDPROCEDURE
```

Exercise 3.4 (b)

3.4 A teacher wants a program that outputs a report for each student: the student's name, then a line of text that depends on the mark ("Distinction" for 80 or more, "Pass" for 50 or more, otherwise "Not yet").

(b) Write a main program that inputs the name and the mark of each of 30 students, and calls **Report** for each one. Declare your variables. [5]

Exercise 3.4 (b)

3.4 A teacher wants a program that outputs a report for each student: the student's name, then a line of text that depends on the mark ("Distinction" for 80 or more, "Pass" for 50 or more, otherwise "Not yet").

(b) Write a main program that inputs the name and the mark of each of 30 students, and calls **Report** for each one. Declare your variables. [5]

(b) the variables declared

- a FOR loop that runs 30 times
- the name and the mark input inside the loop
- **CALL Report(...)** inside the loop
- the arguments in the same order as the parameters

```
DECLARE Student : INTEGER
DECLARE StudentName : STRING
DECLARE StudentMark : INTEGER
FOR Student ← 1 TO 30
    INPUT StudentName
    INPUT StudentMark
    CALL Report(StudentName, StudentMark)
NEXT Student
```