

Exercise walk-through

11.3.2

Parameters by value and by reference, and variable scope

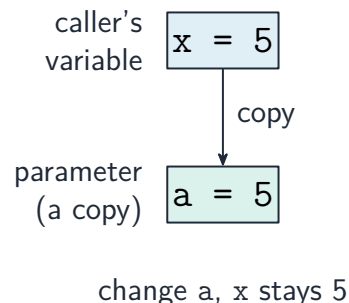
A-Level Computer Science

Parameters: by value and by reference

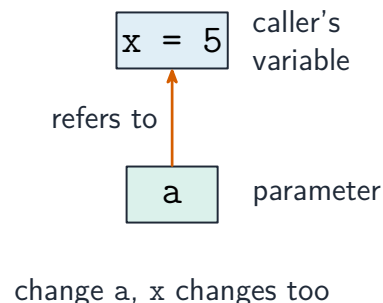
A **parameter** is a variable a subroutine declares to receive input; the values the caller supplies are **arguments**.

- **pass by value** : the routine gets a **copy**; changes inside it do not affect the caller. Use it for inputs the routine only reads.
- **pass by reference** : the routine gets a **reference** to the caller's variable; changes **do** affect the caller. Use it when the routine must update a parameter.

pass by value



pass by reference



Value copies; reference lets the routine change the caller's own variable

BYVAL and BYREF in the header

```
PROCEDURE Swap(BYREF a : INTEGER,  
               BYREF b : INTEGER)  
  DECLARE Temp : INTEGER  
  Temp ← a  
  a ← b  
  b ← Temp  
ENDPROCEDURE
```

Cambridge pseudocode writes the mode, BYVAL or BYREF, in the header **before each parameter**.

If neither is written, BYVAL is assumed. A routine that must change the caller's variable, such as Swap or a procedure that updates a running total, needs BYREF.

Worked example: what is output?

```
PROCEDURE Adjust(BYREF X : INTEGER,  
                 BYVAL Y : INTEGER)
```

```
    X ← X + Y
```

```
    Y ← Y * 2
```

```
ENDPROCEDURE
```

```
A ← 5
```

```
B ← 3
```

```
CALL Adjust(A, B)
```

```
OUTPUT A, B
```

X is a **reference** to A, so A becomes 8

Y is a **copy** of B, so doubling Y leaves B at 3

the output is 8, 3

Had the header said **BYVAL** X, A would still be 5.

Local and global variables

A **local variable** is declared inside a subroutine and exists only while it runs. A **global variable** is declared outside and is visible everywhere.

The region where a name is visible is its **scope**.

Prefer locals and parameters: heavy use of globals makes code hard to follow and test.

global scope: the whole program

Total

global: visible everywhere

local scope: procedure Calc

Temp

local: exists only
while Calc runs

Why local variables: the exam answer

The one-line difference: a global variable can be accessed from *anywhere* in the program, a local variable only *inside the subroutine that declares it*. Benefits of local variables:

1

the same identifier can be used in another subroutine **without a clash**

2

the value **cannot be changed accidentally** by other parts of the program

3

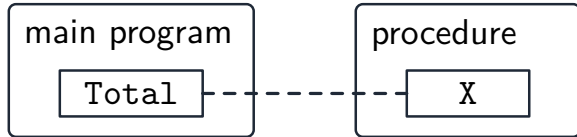
the memory is **released** when the subroutine ends

4

the subroutine is **self-contained**: tested on its own and reused in another program

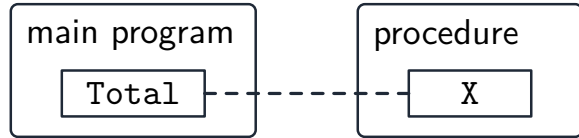
Exercise 2.1

2.1 State what is meant by passing a parameter **by value**, and what is meant by passing it **by reference**. [2]



Exercise 2.1

2.1 State what is meant by passing a parameter **by value**, and what is meant by passing it **by reference**. [2]

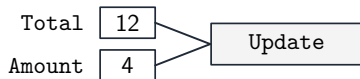


2.1

- **by value** – the subroutine receives a **copy** of the value, so changes it makes do not affect the caller's variable
- **by reference** – the parameter refers to the caller's **own** variable, so changes it makes do affect the caller

Exercises 2.2-2.3

2.2 For the `Update` procedure in the worked example, `Total` is 12 and `Amount` is 4 before the call. State the two values output, and give the reason for each. [2]

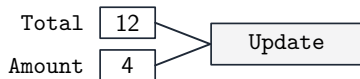


2.3 The header is changed so that **both** parameters are `BYVAL`. State the two values now output, and explain the change. [2]



Exercises 2.2-2.3

2.2 For the `Update` procedure in the worked example, `Total` is 12 and `Amount` is 4 before the call. State the two values output, and give the reason for each. [2]



2.2

- `Total` is **16**: `Count` is `BYREF`, so it is `Total`'s own box and the addition is written into it
- `Amount` is **4**: `Increment` is `BYVAL`, so setting it to 0 only empties the copy

2.3 The header is changed so that **both** parameters are `BYVAL`. State the two values now output, and explain the change. [2]



2.3

- `Total` is **12** and `Amount` is **4**
- both parameters are now copies, so the addition happens inside the procedure and nothing reaches the caller

Exercises 2.4-2.5

2.4 A procedure's header is changed from `BYVAL` to `BYREF` for one parameter. State whether the **interface** of the procedure has changed, and state whether the **argument** written at the call site has to change. [2]

PROCEDURE Add(BYVAL N : INTEGER)



PROCEDURE Add(BYREF N : INTEGER)

2.5 State what is meant by the **scope** of a variable. [1]

program

procedure

Exercises 2.4-2.5

2.4 A procedure's header is changed from `BYVAL` to `BYREF` for one parameter. State whether the **interface** of the procedure has changed, and state whether the **argument** written at the call site has to change. [2]

PROCEDURE Add(BYVAL N : INTEGER)

PROCEDURE Add(BYREF N : INTEGER)



2.4

- the **interface has changed**: the passing mode is part of the interface, which is everything a caller must know in order to use the procedure
- the **argument does not change**: it is still written as the variable's name at the call site, exactly as before – what changes is what the procedure may do to it

2.5 State what is meant by the **scope** of a variable. [1]

program

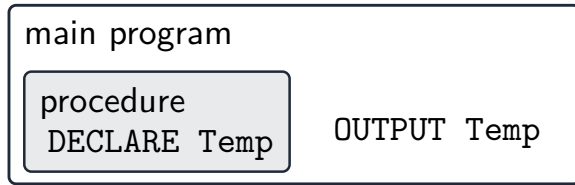
procedure

2.5

- the scope of a variable is the part of the program in which that name can be accessed

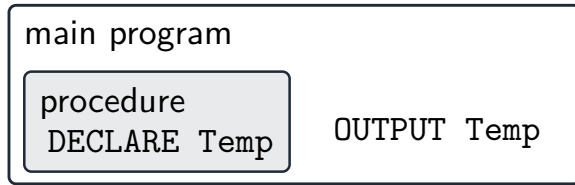
Exercise 2.6

2.6 A programmer declares a variable inside a procedure and then tries to use it in the main program. State what happens, and why. [2]



Exercise 2.6

2.6 A programmer declares a variable inside a procedure and then tries to use it in the main program. State what happens, and why. [2]

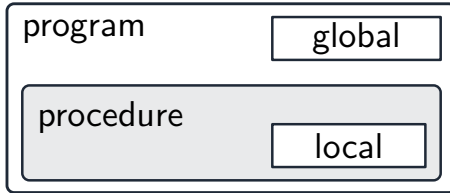


2.6

- the variable is **local** to the procedure, so it does not exist outside it
- the main program cannot access the name, so this is an error (and the value is destroyed when the procedure returns)

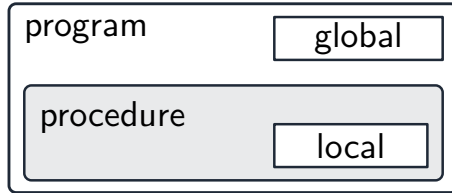
Exercise 2.7

2.7 State **two** advantages of using a local variable rather than a global variable. [2]



Exercise 2.7

2.7 State **two** advantages of using a local variable rather than a global variable. [2]



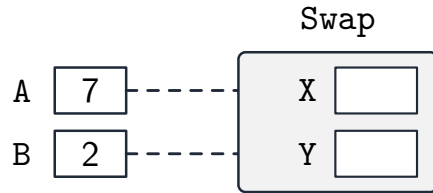
2.7

- any two: the same identifier may be used in another subroutine without a clash; the value cannot be changed accidentally by another part of the program; the memory is released when the subroutine ends; the subroutine is self-contained, so it can be tested on its own and reused

Exercise 3.1 (a)

3.1 A procedure `Swap()` is intended to exchange the values of two integer variables. Before the call, A is 7 and B is 2.

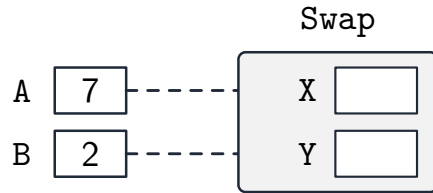
(a) The header is `PROCEDURE Swap(BYVAL X : INTEGER, BYVAL Y : INTEGER)`. State the values of A and B after the call, and explain your answer. [2]



Exercise 3.1 (a)

3.1 A procedure `Swap()` is intended to exchange the values of two integer variables. Before the call, A is 7 and B is 2.

(a) The header is `PROCEDURE Swap(BYVAL X : INTEGER, BYVAL Y : INTEGER)`. State the values of A and B after the call, and explain your answer. [2]



(a) A is still **7** and B is still **2**

- both parameters are copies, so the exchange happens inside the procedure and is lost when it returns

Exercise 3.1 (b)

3.1 A procedure `Swap()` is intended to exchange the values of two integer variables. Before the call, A is 7 and B is 2.

(b) Write a header that makes the procedure work as intended.

[2]

Exercise 3.1 (b)

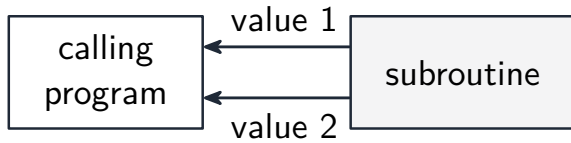
3.1 A procedure `Swap()` is intended to exchange the values of two integer variables. Before the call, `A` is 7 and `B` is 2.

(b) Write a header that makes the procedure work as intended. [2]

(b) both parameters must be passed by reference `PROCEDURE Swap(BYREF X : INTEGER, BYREF Y : INTEGER)`

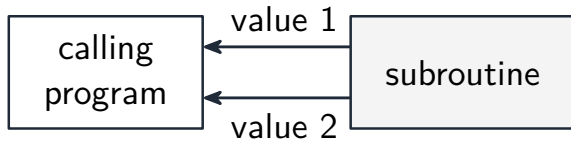
Exercise 3.2

3.2 A subroutine must give the calling program **two** values. Describe **two** different ways of doing this, and state one reason to prefer each. [4]



Exercise 3.2

3.2 A subroutine must give the calling program **two** values. Describe **two** different ways of doing this, and state one reason to prefer each. [4]



3.2

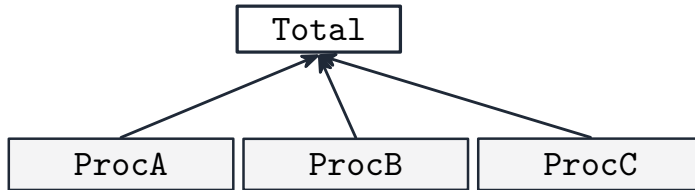
- **two BYREF parameters** – the procedure writes into the caller’s own variables
- prefer this when neither value is “the answer”: both are updates the caller asked for
- **a function returning one value, with the other as a BYREF parameter**
- prefer a function when one value is clearly the result, because the return value can be used directly in an expression

Exercise 3.3 (a)

3.3 A program stores a running total in a global variable that three different procedures update.

(a) State **two** problems this can cause.

[2]

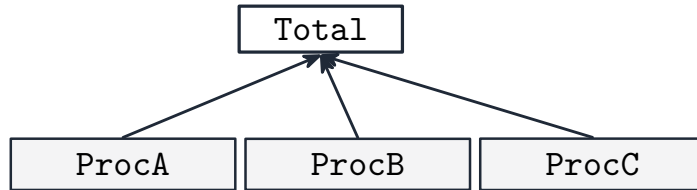


Exercise 3.3 (a)

3.3 A program stores a running total in a global variable that three different procedures update.

(a) State **two** problems this can cause.

[2]



(a) any two: any procedure can change it, so a fault is hard to trace to one place; a change made by one procedure can be overwritten by another; the procedures cannot be tested on their own; the name cannot be reused elsewhere

Exercise 3.3 (b)

3.3 A program stores a running total in a global variable that three different procedures update.

(b) Describe one change that removes those problems. [2]

Exercise 3.3 (b)

3.3 A program stores a running total in a global variable that three different procedures update.

(b) Describe one change that removes those problems. [2]

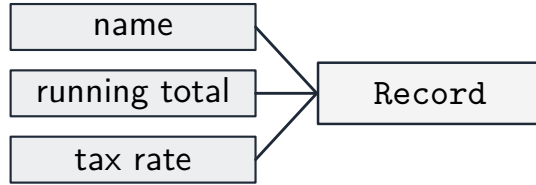
(b) make the total a **local** variable in the calling program and pass it to each procedure as a **BYREF** parameter

- each procedure is then self-contained and can be tested alone, and only a procedure actually given the total can change it

Exercise 3.4

3.4 A procedure `Record()` takes three parameters: a customer's name, which it only reads; a running total, which it must increase; and a tax rate, which it only reads.

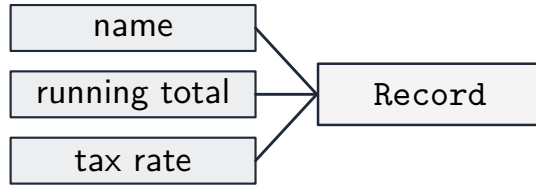
State, with a reason, whether each parameter should be passed **BYVAL** or **BYREF**. [4]



Exercise 3.4

3.4 A procedure `Record()` takes three parameters: a customer's name, which it only reads; a running total, which it must increase; and a tax rate, which it only reads.

State, with a reason, whether each parameter should be passed **BYVAL** or **BYREF**. [4]



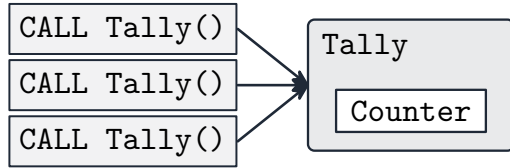
3.4

- the name: **BYVAL** – it is only read, and a copy protects the caller's variable
- the running total: **BYREF** – the procedure must increase it and the new value has to reach the caller
- the tax rate: **BYVAL**: it is only read, and nothing needs to go back to the caller

Exercise 3.5

3.5 A procedure declares a counter as a local variable, adds 1 to it and outputs it. The procedure is called three times.

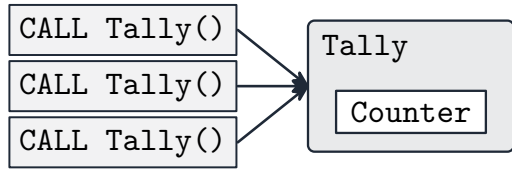
State the three values output, explain why, and name **two** ways to make the counter keep its value between calls. [3]



Exercise 3.5

3.5 A procedure declares a counter as a local variable, adds 1 to it and outputs it. The procedure is called three times.

State the three values output, explain why, and name **two** ways to make the counter keep its value between calls. [3]



3.5

- the outputs are **1, 1, 1**
- a local variable is created when the procedure is called and destroyed when it returns, so each call starts with a new, empty counter and the earlier value is gone
- to keep it: make the counter **global**, or pass it in as a **BYREF parameter** so the box belongs to the caller