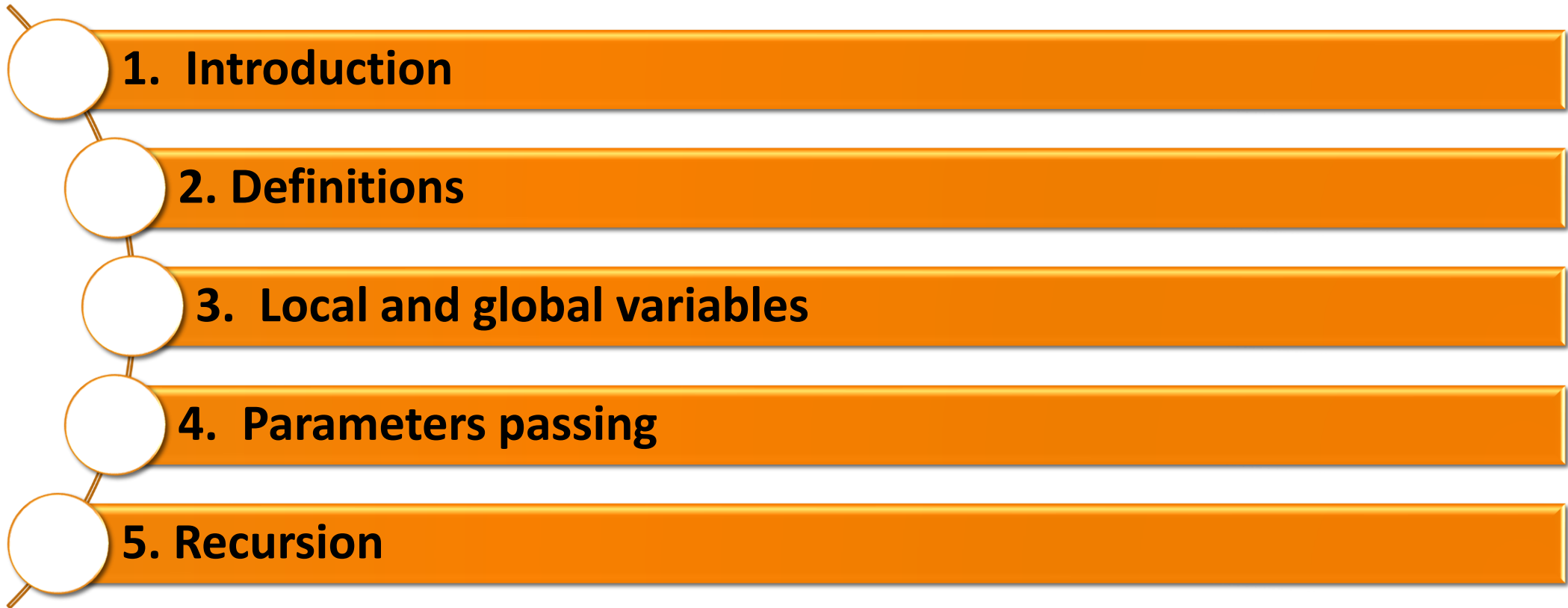


Sub-programs: Procedures and Functions

CHAPTER 6

Chapter 6 : Sub-programs

- 
- 1. Introduction**
 - 2. Definitions**
 - 3. Local and global variables**
 - 4. Parameters passing**
 - 5. Recursion**

Introduction : Illustrative example

Consider the following problem:

Write an algorithm that uses 2 numerical values to perform addition (+), subtraction (-), multiplication (*) or division, depending on the user's choice.

Two solutions can be proposed:

```
Algorithm Calculator;  
Var x,y, res: real; op: character;  
Begin  
  Read (x, y);  
  Read (op);  
  case op of  
    '+': begin  
      res  $\leftarrow$  x+y;  
      Write (res);  
      End;  
    '-': begin  
      res  $\leftarrow$  x-y;  
      Write (res);  
      End;  
    '*': begin  
      res  $\leftarrow$  x*y;
```

```
      Write (res);  
      End;  
    '/' : If y $\neq$ 0 then  
      Begin  
        res  $\leftarrow$  x/y;  
        Write (res);  
        End;  
      else write ('indefinite');  
    else : write ('unknown operation');  
  end;  
End.
```

```
Algorithm Calculator;  
Var x,y, res: real; op: character;  
procedure Addition (a, b: real,  
var c: real);  
Begin  
c ← a+b;  
Write(c);  
End;  
procedure Subtraction (a, b: real,  
var c: real);  
Begin  
c ← a-b;  
Write(c);  
End;  
procedure Multiplication (a, b:  
real, var c: real);
```

```
Begin  
c ← a*b;  
Write(c);  
End;  
procedure Division (a, b: real,  
var c: real);  
Begin  
If b ≠ 0 then  
    Begin  
        c ← a/b;  
        Write(c);  
    End  
else write ('indefinite');  
End;
```

```
Begin  
Read (x, y);  
Read (op);  
case op in  
    '+': addition(x, y, res);  
    '-' : subtraction(x, y, res);  
    '*': multiplication(x, y, res);  
    '/' : division(x, y, res) ;  
    else: write ('unknown  
operation');  
end;  
End.
```

Introduction

- All the problems (and their algorithms) we have studied so far have been implemented as a single processing block.
- As the problems become more complex, the size of the algorithm increases accordingly, as does the difficulty of managing and understanding it.
- Addressing a practical problem requires a complex and time-consuming approach. Solving this type of problem in a single phase can lead
 - to failure
 - or to a complicated and unreadable algorithm, which is quite difficult to write.

Introduction

- Decomposing a complex problem into smaller, manageable parts is key to solving it efficiently and reliably.
- The overall solution is then constructed by combining these individual solutions.
- A sub-program can be either a procedure or a function.

Definitions: Sub-programs

- A sub-program is the solution to a sub-problem.
- It is a block of instructions that enables a simple task to be carried out.
- A task is a well-defined action of varying complexity that is performed on one or more objects at a given time.
- A sub-program is declared in the header and then called in the body of the algorithm.

Definitions: Sub-programs

Sub-programs have the following properties:

- A sub-program is designed to perform a well-defined and well-scoped operation, ideally independent of the specific context of the calling algorithm
- Sub-programs can be nested.
- An algorithm calls a sub-program, transferring control of processing execution to that sub-program for a limited period.
 - The calling program is suspended during execution of the called subprogram.
 - Control always returns to the caller when the called subprogram's execution terminates.
- A sub-program can call another one.

Definitions: Sub-programs

A sub-program provide several advantages:

- Each sub-program can be implemented and tested separately.
 - This facilitates updating and error correction.
- If the same processing is required at several points in the program, the corresponding sub-program can simply be called several times instead of repeating the same block of instructions.
 - This improves the clarity of algorithms and optimises the number of instructions.

Definitions: Sub-programs

For example, a sub-program may :

- Swap the contents of the integers A and B.
- Divide the integer A by the integer B to obtain the result Q and the remainder R.
- Raise A to the power of B and store the result in C.
- Sort the sequence of N elements contained in an array T.
- ...

Definitions: Procedures

- A procedure is a sub-program that performs a simple task, identified by a *name* and made up of a *set of instructions*.
- A procedure accepts *arguments* and returns *results* (zero, one or more) *of any type*.
- A procedure can also be defined as a sub-program which solves a given problem and which can have several results based on one or more given arguments.

Definitions: Procedures

- The declaration of a procedure defines a sub-program and associates it with an identifier through which it will be called.
- It occurs in the algorithm's header and involves:
 - Assigning a name to the procedure in the header.
 - Defining its variable declaration part specific to it.
 - Defining its processing part, which constitutes the body of the procedure.
- Syntactically, the declaration is done in the algorithm's header, as follows:

Definitions: Procedures

- The declaration of a procedure defines a sub-program and associates it with an identifier through which it will be called.
- It occurs in the algorithm's header and involves:
 - Assigning a name to the procedure in the header.
 - Defining its variable declaration part specific to it.

Formal parameters

```
Procedure procedure_name (parameter1, parameter2,...,parameterN);  
⟨Local_Object_Declarations⟩;  
Begin  
⟨Instruction_Block⟩;  
End ;
```

The body of the procedure.

Definitions: Procedures

Notes :

- The parameters parameter1, parameter2, ..., parameterN are called Formal Parameters. Each formal parameter is defined by an identifier, a type and a transmission mode, as following :

transmission mode parameter_identifier : parameter_type

- The name and type of the parameter is defined in the same way as for variables.
- The transmission mode specifies whether the parameter is an input or output.

Procedure call

- Once a subprogram has been declared, it can be used.
- To use a procedure (i.e. to execute its instructions), an algorithm calls (or invokes) this procedure using the procedure call instruction with the real parameters or arguments, also referred to as **actual parameters**.

procedure_name (parameter1, parameter2,...,parameterN);

Procedure call

- Calling a parameterized procedure involves the following steps:
 1. The address of the instruction following the call is saved as the **return address**.
 2. Each formal parameter is matched with its corresponding actual parameter.
 3. The statements of the called procedure are executed.
 4. At the end of the called procedure:
 - a) All local objects are deleted,
 - b) The caller (sub-program/main algorithm) is resumed and execution continues from the return address.

Formal and actual parameters

- The parameters are used to allow the procedure to be executed several times, with different values (arguments).
 - For example, The procedure `division(a, b, Q, R)` can be called with different arguments, such as `division(7, 2, X, Y)` or `division(24, 5, Q, R)` or `division(n, m, x, y)`.
- Their declaration in the header of a procedure P describes the data (their numbers, types and order) expected by P to be executed.
- **Formal parameters** are used to **declare** the procedure,
- **Actual parameters** are used to **call** the procedure.

Formal and actual parameters

```
Algorithm Power_Calculation ;
```

```
  Var uneVal, R : Real ;
```

```
    nbPuissances: integer ;
```

```
  Procedure Power (X : Real, N : integer, Var NB : real) ;
```

```
  begin
```

```
    ...
```

```
  End ;
```

```
    ...
```

```
  begin
```

```
    ...
```

```
    Power(unVal, nbPuissances, R) ;
```

```
    ...
```

```
  end.
```

Formal parameters of the procedure

Procedure call

Actual parameters of
the procedure

Application Exercises

1. Write an algorithm that input a positive non-zero integer **N** and displays whether it is perfect or not.
2. Write an algorithm to enter a real number **X** and an integer **n** , and then calculate and display **X^n** , regardless of the value of **n** .
3. Write an algorithm to enter an array of 10 characters, then reverse it, and finally display it.

Definitions : Functions

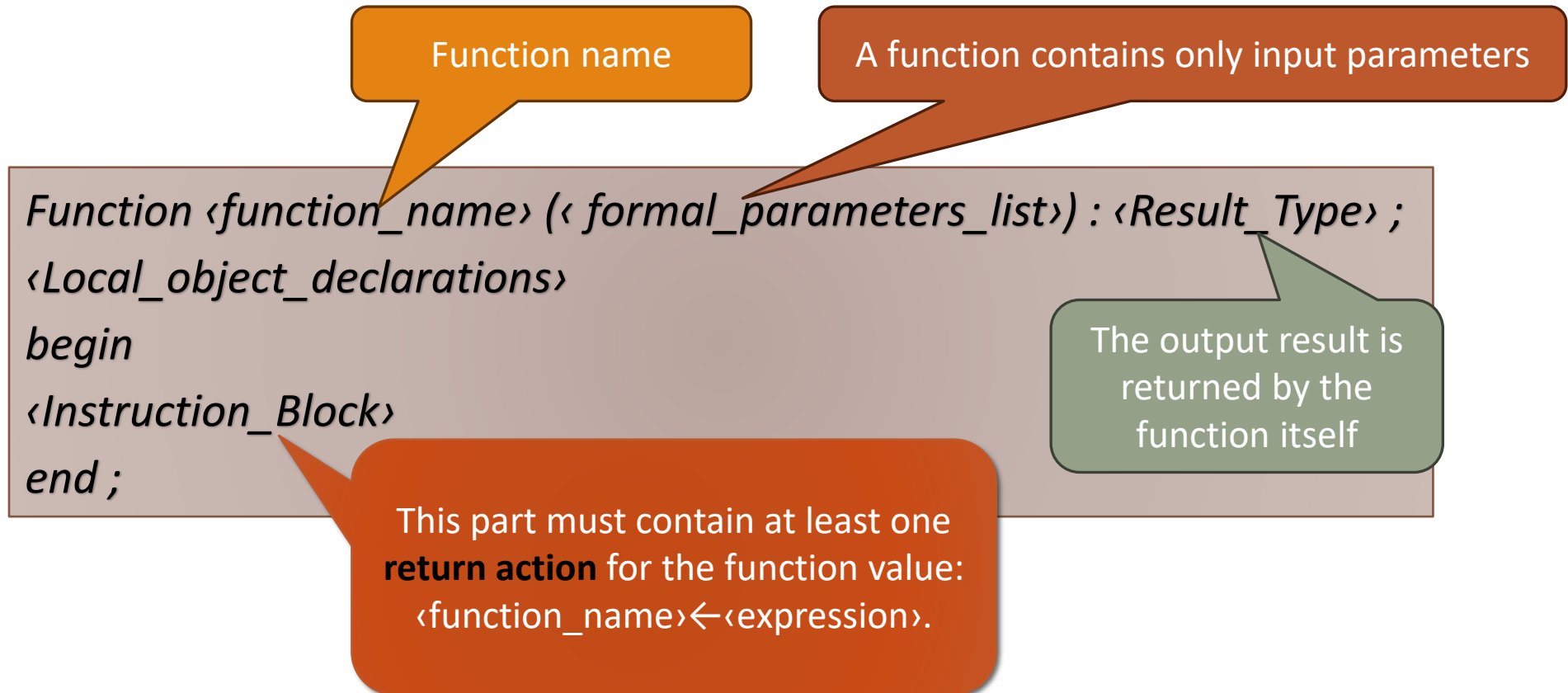
- Procedures accept input data (variables or constants) and may return results (via parameters passed by reference).
- Functions are also sub-programs that:
 - accept parameters as input,
 - return only a single scalar result.
- This result is returned via the function name.

Definitions: Functions

- A function can be defined as a sub-program that accepts arguments (parameters) and must return a single result (a value) of scalar type.
- Similar to a procedure, a function must be declared before it is used, which is typically done in the algorithm's header, involving:
 - Assigning a name to the function in the header.
 - Defining its variable declaration part specific to it.
 - Defining its processing part, which constitutes the body of the function.

Definitions: Functions

- Syntactically, the declaration occurs in the algorithm's header, as follows:



Function call

- The function is called by its name, followed by a list of parameters, as part of an expression to be calculated.
 - Case 1 :
⟨variable_name⟩ ← ⟨function_name⟩ (parameter1, parameter2,...,parameterN);
 - Case 2 :
If (⟨function_name⟩ (parameter1, parameter2,...,parameterN)=⟨variable_name⟩) then....
 - Case 3 :
Write (⟨function_name⟩ (parameter1, parameter2,...,parameterN));

Note

Formal parameters and local objects are subject to the same rules as procedures.

Application Exercises

1. Write an algorithm that input a positive non-zero integer **N** and displays whether it is perfect or not.
2. Write an algorithm to enter a real number **X** and an integer **n** , and then calculate and display **X^n** , regardless of the value of **n** .
3. Write an algorithm to enter a positive non-zero integer **N** and calculate the following sum:
$$1! + 2! + 3! + \dots + N!$$

Local and global variables

- **Local object:** An object declared within a sub-program (the called) is only accessible inside that sub-program. It is said to be **local** and **not visible** elsewhere in the program.
- **Global object:** An object declared in the main block (main program or sub-program) is accessible to all nested sub-programs. It is said to be **global** and **visible** throughout the block where it is declared.

```
Procedure P3 ;  
  Var k : integer ;  
    Procedure P2 ;  
      Var j : integer ;  
        Procedure P ;  
          Var i : integer ;  
          begin  
            ...  
          end ;  
        ...  
      end ;  
    ...  
  end ;  
...  
end ;
```

Local and global variables

- A global variable and a local variable can have **the same name**. In this case, **locality masks globality**: the variable used in the sub-program is always the local variable.
- An object **B** declared in a sub-program **P** is global to any sub-program declared in **P**.

```
Algorithm GlobLoc_Var;  
Var  X, Y : Integer;  
    Procedure display;  
    Var X: Integer;  
    Begin  
    X ← 1;  
    Write(X, Y);  
    Y ← 100;  
    Write(X, Y);  
    End;  
Begin  
X ← 20;  
Y ← 10;  
display;  
Write(X, Y);  
END.
```

Parameters transmission

- Formal variables are defined either by **value** or by **reference**
- A parameter is said to be "**by value**" when only its value is useful. It cannot be modified directly in the procedure, but its value can be used to perform calculations.
- In other words,
 - The procedure receives a copy of the value of the actual parameter,
 - The original value of the effective parameter is retained,
 - Even if the procedure assigns a new value to the corresponding formal parameter

Parameters transmission

- A parameter passed by **variable** (**reference/ address**) allows both the use and modification of the value of the original variable in the called procedure.
- These variables are preceded by **VAR** keyword in the procedure header.
- The called procedure has access to the memory address of the original variable. This means that changes made to the variable in the called procedure also affect the original variable in the caller.

Example

Algorithm example1 ;

Var a: integer;

 Procedure clear (x: integer) ;

 Begin

$x \leftarrow 0$;

 end ;

Begin

$a \leftarrow 5$;

clear (a) ;

write(a) ;

End.

Algorithm example2 ;

Var a: integer;

 Procedure clear (**Var** x: integer) ;

 Begin

$x \leftarrow 0$;

 end ;

Begin

$a \leftarrow 5$;

clear (a) ;

write(a) ;

End.

Parameter substitution rules

- 1. Actual/formal count match:** The number of formal parameters must correspond to the number of effective parameters when the function is called.
- 2. Actual/formal parameter substitution:** Each actual parameter is substituted for the corresponding formal parameter, following the order of declaration.
- 3. Type matching:** The type of each effective parameter must be compatible with the type of the corresponding formal parameter.

Parameter substitution rules

- 4. Pass-by value:** A pass-by-value parameter creates a local variable whose initial value is the current value of the effective parameter. Modifying this variable within the procedure does not affect the value of the effective parameter.
- 5. Pass by reference:** The formal parameter and the actual parameter must be variables. Modifying the formal parameter within the procedure also modifies the value of the actual parameter.
- 6. Result parameters:** A result parameter must be declared by reference to receive the return value.