Algorithmic and Data Structure 2

Chapter 1

"Procedures & Functions"

Outline

Chapter 1: Procedures & Functions

1. Definitions

A sub-algorithm is a block that is part of an algorithm. It is declared in the header part (before the start of the algorithm "begin"), then called in the body of the algorithm.
Since it is a block in its own right, it possibly has a header, a series of instructions, and management of results just like the algorithm that contains it.

Notes:

- A sub-algorithm uses the variables declared in the algorithm (called global variables). It can also have its own variables (called local) declared in the space reserved for it; but which can only be used in this sub-algorithm and nowhere else because its scope (visibility) is limited to the block which contains it. The space of these local variables is only reserved when the sub-algorithm is called and is freed at the end of execution.
- A sub-algorithm is declared in a general way i.e. it can be called several times with different values using **arguments**. These latter, although they are optional, are called **parameters** and are clearly declared, if necessary, in the header of the sub-algorithm.
- A parameter is a value of the main block that the sub-algorithm needs to execute with **real data** the sequence of actions (instructions) that he is responsible for carrying out. There are two types of parameters:
    - → **The formal parameters**, definition of the number and type of values that the sub-algorithm must receive to start successfully. We declare the formal parameters during the declaration of the sub-algorithm.
    - → **The effective parameters** are real values (constant or variable) received by the sub-algorithm during the execution of the main block. They are defined independently at each call of the sub-algorithm in the main algorithm.
    - The execution of a sub-algorithm (procedure or function) is done by a call instruction (following sections). Applying this instruction generates a jump to the called sub-algorithm. The end of this sub-algorithm restarts the instruction sequence interrupted by the call.

2. Sub-algorithm types

A sub-algorithm can be in the form of a function or procedure.

A function is a sub-algorithm, which, from data, calculates and returns to the algorithm **One and Only one result**, whereas in general, a procedure displays the requested result(s).

2.1.Procedure

A procedure is a block of instructions named and declared in the header of the algorithm and called in its body whenever the programmer needs it.

Declaration of a procedure:

Procedure Procedure_Name (Parameter_Name: Prarameter_Type ;……);

Declaration

Variable_name: Variable_type; } **Local variables**

…

Begin

… Instructions; } **Body of the procedure**

END;

A procedure can be called by specifying, at the desired time, its name and possibly its parameters; this triggers the execution of the procedure instructions.

Example : Here is an algorithm using a procedure that calculates a sum of 100 numbers.

```
Algorithm Test;
Variables  i, S: integer;
Procedure Sum;
Begin  /*Start of Procedure*/
S ←0;
For i← 1 to 100 do
   S ← S + i
EndFor
Write (“The sum of the first 100 numbers is”, S);
```

```
End; /*End of Procedure*/
Begin /*Start of algorithm*/
Sum
END /*End of algorithm*/
```

2.2.Function

A function is a block of instructions that necessarily returns one and only one result value to the calling algorithm. A function never displays the response on the screen because it simply returns it to the calling algorithm.

Declaration of a function:

```
Function Function_Name (Parameter_Name: Prarameter_Type ;……): Function_Type;
Declaration
Variable_name: Variable_type;        } Local variables
…
Begin
… Instructions;                      } Body of the function
   Function_Name ← Result;
END;
```

Since the main purpose of a function is to return a value, it is therefore necessary to specify the type of the function which is in reality the type of this value.
A function call is an assignment expression so that the result is retrieved into a global variable

```
Global-variable-name ← Function_Name (parameters);
```

Example: The previous algorithm, which calculates a sum of N numbers, can use a function instead of a procedure.

```
Algorithm Test;
Variables  i, Som: integer;
Function  Sum : integer;
```

```
Variable S: integer;
Begin  /*Start of Function*/
S ←0;
For i← 1 to 100 do
   S ← S + i;
EndFor
  Sum ← S;
End; /*End of Function*/
Begin /*Start of algorithm*/
Som ←  Sum;
Write ("The sum of  100 first numbers is", Som);
END /*End of algorithm*/
```

Note: Just like a procedure, a function can call other sub-algorithms provided that they are defined before it or that they are declared in its header.

3. Parameter passing mode

A sub-algorithm with parameters is very useful because it allows you to repeat a series of complex operations for values that we do not know in advance. There are two types of parameter passing: by value and by variable (also called by reference or by address).

3.1.Passing parameters by value

This is the default transmission mode, there is a copy of the value. In this mode, the content of the effective parameters cannot be modified by the instructions of the function or procedure; because we are not working directly with the variable, but on a copy. At the end of the execution of the sub-algorithm the variable will retain its initial value. The parameters in this case are used as data.

Syntax:

```
Procedure procedure_name (param1 :type1 ; param2, param3 :type2);
Function <function_name> (param1 :type1 ; param2 :type2): Function_type;
```

Example : Consider the following algorithm.

```
Algorithm Pas-val;
Variable M: integer;
Procedure P1 (number: integer);
Begin
 If (number < 0) then
    number ← -number;
 Endif
   Write (number);
END;
Begin
   Read (M);
   P1(M);
   Write (M);
END
```

Let's run this algorithm for the value **(-6)**

Before the procedure call: the only variable declared is the global variable (M).

M	Screen
-6	

After the procedure call: the parameter variable "number" is declared and receives a copy of the value of M.

M	number	Screen
-6	-6	
-6	6	
-6	6	6

When returning to the algorithm (at the call level) only the global variable remains with its initial value.

M	Screen
-6	
-6	-6

3.2.Passing parameters by variable

Here, it is no longer a matter of simply using the value of the variable, but also its location in memory (hence the expression "by address"). In fact, the formal parameter replaces the effective parameter during the execution time of the sub-algorithm and at the exit it transmits its new value.

Such a parameter passing is done by using the keyword "**Var**" .

Syntax:

```
Procedure procedure_name (Var param1 :type1, param2, param3 :type2);
Function <function_name> (Var param1: type1, param2 : type2): Function_type;
```

Note: Parameters passed by value and by address can coexist within the same sub-algorithm. You just need to separate the two types of passage with a (;).

Syntax:

```
Procedure procedure_name (Var param1 :type1 ; param2, param3 :type2);
```

In this case **param1** is passed by **reference** while the other two are passed by **value.**

```
Function <function_name> (param1 :type1 ; Var param2 :type2): Function_type;
```

In this case **param1** is passed by **value** while the **second** is passed by **variable (reference).**

Example : Either the previous algorithm modified in the parameter passing mode.

```
Algorithm Pas-val;
Variable M: integer;
Procedure P1 (Var number: integer);
Begin
 If (number < 0) then
    number ← -number;
 End if
   Write (number);
END
Begin
   Read (M);
   P1(M);
   Write (M);
END
```

Let's run this algorithm for the value **(-6)**

Before the procedure call: the only variable declared is the global variable (M).

M	Screen
-6	

After the procedure call: the variable-parameter number replaces the variable M.

~~M~~ number	Screen
-6	
6	
6	6

When returning to the algorithm only the global variable remains with its new value.

M	Screen
-6 “input value”	
6	6

4. Application examples

Example 1 :

An algorithm that calculates and displays the absolute value of a value using a function.

```
Algorithm example1;
Variable a, b: integer;
Function abs (n: Integer): Integer;
Variable  AbsoluteValue: integer;
Begin
If (n >= 0) then
    AbsoluteValue ←  n;
else
    AbsoluteValue ←  - n;
Endif
    abs ←  AbsoluteValue ;
END
```

```
Begin
Write ("Enter an integer :");
Read (a);
b ← abs (a);
Write ("the absolute value of ", a, "is", b);
END
```

Example 2:

It is required to write an algorithm that asks the user to enter an integer value (X) then it tells the user if X is a positive or negative number.

```
Algorithm example2;
Variable X: integer;
Procedure positive_negative (n: Integer);
Begin
If (n >= 0) then
   Write ("positive");
else
   Write ("negative");
Endif
END
Begin
Write ("Enter an integer :");
Read (X);
Positive_negative (X);
END
```