Chapter: Subprograms (Procedures and Functions)

1. Introduction:

- As one progresses in the design of an algorithm, it may grow in size and complexity. Similarly, sequences of instructions may be repeated in multiple places.
- An algorithm written in this manner becomes challenging to understand and manage, especially when it extends beyond four pages. ⇒ The solution is to break down the algorithm into smaller parts. These parts are known as sub-algorithms.
- The sub-algorithm is written separately from the main body of the algorithm and will be called by it when necessary.
- There are two types of sub-algorithms (programs): procedures and functions.

2. Procedures:

- A procedure is a sub-program identified by a name.
- It can be called in another program or at various locations within the same program or even in another sub-program.
- It allows for the execution of actions through
- a simple call, similar to an instruction, using different data.
- A procedure can return multiple values (by one) or no value.

2.1 Procedure Declaration:

```
Procedure proc_name(parameter list and declarations);
Variables identifiers: type;
  Begin
    Instructions;
  End;
```

2.1.1 **Similar structure to a program** A procedure has a structure similar to that of a program, often composed of a sequence of instructions that perform a specific task.

2.1.2 Parameter list: After the name of the procedure, we specify the list of parameters within parentheses, if any. These parameters are also called formal parameters or arguments. Each parameter is accompanied by its respective type, which indicates the type of data the procedure can accept.

2.1.3 **Formal parameters:** The parameters specified in the declaration of the procedure are formal parameters. Their value is not known at the time of the creation of the procedure. They act as placeholders for the values that will be provided when the procedure is called.

2.1.4 "**Parameter values:** The values of formal parameters are provided when the procedure is called. These values are then assigned to the formal parameters of the procedure and used in the execution of instructions within the procedure.

Note: In summary, a procedure in programming is an entity that groups a set of instructions to perform a specific task. It can accept formal parameters whose values are provided when it's called, and its structure is similar to that of a program, with parameter specification within parentheses after the procedure name.

2.2 Calling a Procedure

- To trigger the execution of a procedure within a program, simply call it.
- Procedure call is written by putting the procedure name, followed by the parameter list, separated by commas.
- When calling a procedure, the program interrupts its normal flow, executes the instructions of the procedure, then returns to the calling program and executes the next instruction

Syntax: Procedure_name(parameter list);

Note:

- The parameters used when calling a procedure are called actual parameters.
- These parameters will provide their values to the formal parameters.
- To execute an algorithm that contains procedures and functions, you need to start execution from the main part (main algorithm).

2.3 Notions of Global and Local Variables In computer programming:

- A global variable is a variable declared outside the body of any function or procedure and can therefore be used anywhere in the program.
- A local variable is a variable that can only be used within the function or block where it is defined.
- The local variable contrasts with the global variable, which can be used throughout the program.

2.4 Parameter Passing

- Information exchange between a procedure and the calling sub-algorithm is done through parameters.
- There are two main types of parameter passing that allow for different uses.

2.4.1 Pass by Value (or Default Value) Parameter Passing:

- When parameters are passed
- by value, a copy of the parameter's value is passed to the called procedure.
- This means that any modifications made to these parameters inside the procedure will not affect the original variables in the calling program.
- This type of parameter passing is typically used when the procedure needs to manipulate the values of parameters without affecting the original variables.

2.4.2 Pass by Reference (or by Address) Parameter Passing:

- When parameters are passed by reference, it's the memory addresses of the original variables that are passed to the called procedure.
- This allows the procedure to directly access and modify the values of the original variables in the calling program.
- Changes made to the parameters inside the procedure will thus directly affect the original variables.
- This type of parameter passing is often used when the procedure needs to modify the values of the original variables or when it's necessary to minimize memory overhead by avoiding data copying.

Note: These two methods offer different approaches to exchanging information between procedures and parts of the calling program, and the choice between them depends on the specific requirements of your application.

Example 01 : Consider the following algorithm:

```
ALGORITHM Pass_by_value;
Variables N: integer;

Procedure P1(A: integer) //* Declaration of procedure P1
Begin
    A <- A * 2;
    write(A);
EndProc

Begin //* Main algorithm
    N <- 5;
    P1(N);
    write(N);
End
```

This algorithm defines a procedure P1 that takes an integer parameter A, multiplies it by 2, and then prints the result. In the main algorithm, it initializes variable N with the value 5, calls procedure P1 with N as an argument, and then prints the value of N.

To trace the execution of this algorithm, let's examine step by step what happens:

1. We declare a variable N of type integer and initialize it to 5.
2. We call the procedure P1 with N as an argument.
3. Inside the P1 procedure, the value of A is multiplied by 2.
4. The resulting value of A (after multiplication) is displayed.
5. Then, we return to the main algorithm and display the value of N.

Now, let's look at the execution trace:

- Initially, N is initialized to 5.
- By calling P1(N), we pass the value of N (which is 5) to the P1 procedure as a parameter by value.
- Inside the P1 procedure, the value of A is multiplied by 2, resulting in 10.
- The value of A (10) is displayed inside the P1 procedure.
- Back in the main algorithm, the value of N remains unchanged at 5 and is displayed.

- What we notice here is that even though we modified the value of A inside the P1 procedure, it does not affect the value of N in the main algorithm. This confirms that parameter passing to the P1 procedure is done by value, meaning only a copy of the value of N is passed to P1, not the reference to the variable N itself.

Example 2: Let's revisit the previous example.

```
ALGORITHM Pass_by_reference;
Variables N: integer;
//* Declaration of procedure P1
Procedure P1 (Var A: integer);
Begin
    A <- A * 2;
    write(A);
EndProc
//* Main Algorithm
Begin
    N <- 5;
    P1(N);
    write(N);
End ;
```

- In this algorithm, we define a procedure P1 that takes an integer parameter A by reference using the keyword Var. Inside P1, the value of A is multiplied by 2 and then printed.

- In the main algorithm, we initialize variable N with the value 5, call procedure P1 with N as an argument, and then print the value of N. Since N is passed by reference to P1, any modification made to A inside P1 will directly affect the value of N in the main algorithm

To trace the execution of this algorithm, let's examine step by step what happens:

1. We declare a variable N of type integer and initialize it to 5.
2. We call the procedure P1 with N as an argument.
3. Inside the P1 procedure, the value of A is multiplied by 2.
4. The resulting value of A (after multiplication) is displayed.
5. Then, we return to the main algorithm and display the value of N.

Now, let's look at the execution trace:

- Initially, N is initialized to 5.
- By calling P1(N), we pass the reference of N to the P1 procedure.
- Inside the P1 procedure, the value of A is multiplied by 2, resulting in 10.
- The value of A (10) is displayed inside the P1 procedure.
- Back in the main algorithm, the value of N is also modified to 10, as N was passed by reference to P1.

What we notice here is that modifying the value of A inside the P1 procedure also affected the value of N in the main algorithm. This confirms that parameter passing to the P1 procedure is done by reference, meaning that the variable N in the main algorithm and the parameter A in the P1 procedure share the same memory address, and any modification to one will affect the other.

Important Note: When there are multiple parameters in the definition of a procedure, it is essential that the same number of parameters are provided at the call, and their order is respected.
Example :

```
PROCEDURE sum(x, y: integer);
Variables S: integer;
 Begin
S <- x + y;
    End;
```

To call the `sum` procedure in this example, we would write:

- `sum(2.5, 7)` → false (the type of actual parameters must be integer).
- `sum(2, 5, 9)` → false (the number of actual parameters must be 2 and not 3).
- `sum(2, 5)` → correct (the type and number of parameters are appropriate).

3. FUNCTIONS

3. FUNCTIONS : A function in programming is a sub-routine or sub-algorithm that performs a specific task and returns a result. Here is a more detailed definition:

- Function Declaration: A function is typically defined by a name, a list of parameters (possibly empty), and a return type, which specifies the type of data the function will return.
- Function Parameters: Parameters can be used to pass values to the function so that it can perform operations on those values. These parameters can be optional or required, depending on the design of the function.
- Function Body: The body of the function contains the instructions that perform the work of the function. These instructions can include calculations, loops, conditions, calls to other functions, etc.
- Return Value: A function can return a single result to the place where it was called. This result is defined by the return type specified in the function declaration. that can appear in an expression, in a comparison, to the right of an assignment, etc.

Here's an example of a function declaration in pseudo-code:

```
Function calculateSum(a, b: integer):
integer;
Begin
    sum = a + b;
    return sum;
EndFunction ;
```

In this example, the calculateSum function takes two integers as input, calculates their sum, and returns the result as an integer.

Example 02 : Define a function that returns the greater of two different numbers:

```
Function greatest(x, y: integer): integer;
Begin
    If x > y Then
        Return x
    Else
        Return y;
    EndIf
EndFunction ;
```

In this function, we compare the values of x and y. If x is greater than y, we return x; otherwise, we return y.
You can then call this function in your main algorithm to find the greater of two numbers:

```
ALGORITHM MyAlgorithm
Variables
    a, b, greatestNumber: integer
Begin
    a <- 10
    b <- 7
    // Call the greatest function with arguments a and b
    greatestNumber <- greatest(a, b)
    Write("The greatest number between ", a, " and ", b,
" is ", greatestNumber)
End
```

In this example, **greatest(a, b)** is the call to the **greatest** function with arguments **a** and **b**. The result returned by the function is stored in the variable **greatestNumber** and displayed in the main algorithm.

3.1 Calling a Function in the Main Algorithm

- We use a function in an expression or a procedure call, just like a variable, but by indicating its parameters in parentheses and separated by commas to return a single result.
- To return the calculated value to the main algorithm or program, you need to write the following instruction:

Value of the Result ← FunctionName;

Example: Write an algorithm calling and using the Max function from the previous

```
Algorithm Call_Max_Function
Variables A, B, M : real
// * Declaration of the Max function
Function Max(X: real, Y: real) : real
BEGIN
    If X > Y Then
        Max(X,Y) ← X
    Else
        Max(X,Y) ← Y
    EndIf
END;
// * Main Algorithm
BEGIN
    Write ("Enter the value of A:");
    Read(A);
    Write ("Enter the value of B:");
    Read(B);
    // * Call to the Max function
    M ← Max(A,B);
    Write ("The greater of these two numbers is: ", M);
END.
```

4. Scope of Variables :

- The scope of a variable refers to the visibility domain of that variable.
- A variable can have a global or local scope.
- Global scope means that the variable is accessible from anywhere in the program, meaning it can be used both in the main block of the program as well as in functions or procedures defined within that program. It exists throughout the lifetime of the program.
- Local scope means that the variable is only accessible within a specific part of the program A variable declared inside a procedure (or a function), typically limited to the function or procedure in which it is declared. Local variables cannot be used outside of their local scope.The lifetime of a local variable is limited to the execution duration of its procedure or function.
- it is possible to declare an identifier (variable) in a procedure that is used in an enclosing level. In this case, locality masks globality.

Exercice : Given the following procedure schema:

```
Procedure A
      Procedure B
         Procedure C

          End C;
      End B;
    Procedure D

    End D;
End A.
```

Questions :

1. Say, justifying which procedures can be called by procedures : A, B, C, and D?
2. If there are variables declared within procedure B, can they be accessed by the procedure: A, D and C?

5. Advantages of Procedures and Functions:

- Procedures or functions help avoid repeating the same sequence of instructions multiple times within a program (algorithm).
- They structure an algorithm into modules and enhance its readability and comprehensibility.
- They facilitate program debugging (i.e., compilation and error detection will be faster when using procedures and/or functions).
- They can even be implemented outside the context of the program; in other words, they can be stored in a library of tools and used by any other programs.