

Contents

Procedures and functions	2
Defining and calling procedures	2
Defining and Calling Functions.....	3
Parameters.....	4
Passing mechanisms	4
Returning values from functions.....	5
Structured Diagram.....	6
Global and local variables	7
Global Variables:	7
Local Variables:	7
Challenge yourself questions	8
Open-ended Questions	8
Close-ended Questions:	8
Fill-in-the-blanks Questions:	8
Scenario-based Questions	8
Answers.....	9
Open-ended questions:.....	9
Close-ended questions:.....	9
Fill-in-the-blanks questions:.....	9
Scenario-based questions:	9

Procedures and functions

- Both are used to organize code into reusable pieces.
- Procedures perform a specific action and do not return a value.
- Functions perform a specific action and return a value.

Defining and calling procedures

- Syntax for defining a procedure without parameters:

```
PROCEDURE <identifier()>  
    <statements>  
ENDPROCEDURE
```

- Syntax for defining a procedure with parameters:

```
PROCEDURE <identifier()>(<param1>:<datatype>, <param2>:<datatype>...)  
    <statements>  
ENDPROCEDURE
```

- To call a procedure:

```
CALL <identifier()>  
CALL <identifier()>(Value1, Value2...)
```

- Note that when using parameters, the values must match the data types specified in the procedure definition.

Example:

```
//Main Program  
IF MySize = Default  
    THEN  
        CALL DefaultLine()  
    ELSE  
        CALL Line(MySize)  
    ENDIF  
  
PROCEDURE DefaultLine()  
    CALL LINE(60)  
ENDPROCEDURE  
  
PROCEDURE Line(Size : INTEGER)  
    DECLARE Length : INTEGER  
    FOR Length ← 1 TO Size  
        OUTPUT '-'  
    NEXT Length  
ENDPROCEDURE
```


Defining and Calling Functions

- Syntax for defining a function without parameters:

```
FUNCTION <identifier()> RETURNS <data type>
    <statements>
    RETURN <identifier>
ENDFUNCTION
```

Syntax for defining a function with parameters:

```
FUNCTION <identifier()>(<param1>:<datatype>, <param2>:<datatype>...) RETURNS <data type>
    <statements>
    RETURN <identifier>
ENDFUNCTION
```

 The keyword RETURN is used to specify the value to be returned. Typically, this is the last statement in the function's body.

 Functions are called as part of an expression and do not use the keyword CALL.

Example:

```
OUTPUT "Sum of squares = ", SumSquare(10, 20)
X ← SumSquare(10, 20)
IF SumSquare(x, y) > 450 THEN DO_SOMETHING

FUNCTION SumSquare(Number1 : INTEGER, Number2 : INTEGER) RETURNS INTEGER
    RETURN Number1 * Number1 + Number2 * Number2
ENDFUNCTION
```

To summarize, procedures and functions help organize code into reusable pieces. Procedures perform actions without returning values, while functions return values. The syntax for defining and calling procedures and functions depends on whether parameters are used. Remember that functions should be called as part of an expression, assignment or condition without using the CALL keyword.

Parameters

- Parameters are variables passed to procedures and functions to customize their behavior.
- They are specified in the definition of the procedure or function, and their data type must be explicitly declared.
- When calling a procedure or function, values are provided for each parameter. These values are called arguments.

Passing mechanisms

- In programming, there are two main ways to pass parameters: pass by value (`ByVal`) and pass by reference (`ByRef`).
- Pass by value: This is the default mechanism in most programming languages. When passing by value, a copy of the value is passed to the procedure or function. Any changes made to the parameter within the procedure or function will not affect the original value outside of it.
- Pass by reference: In this mechanism, a reference (memory address) to the original value is passed to the procedure or function. Any changes made to the parameter within the procedure or function will directly affect the original value outside of it. Some programming languages support this feature explicitly with specific syntax, while others do not.

```
X ← 9
Y ← 15
CALL aPro(X, Y)
```

```
PROCEDURE aPro(ByVal A : INTEGER, ByRef B : INTEGER)
    A = 50
    B = 150
ENDPROCEDURE
```

Returning values from functions

- ❏ Functions are designed to return a single value, which can be of any specified data type.
- ❏ The RETURN keyword is used within the function to specify the value to be returned. Typically, this is the last statement in the function definition.
- ❏ When a function is called as part of an expression, the returned value replaces the function call in the expression, and the expression is then evaluated.

Here's an example demonstrating parameters, passing mechanism (pass by value), and returning values from functions:

```
//Global Variables
DECLARE num1: INTEGER
DECLARE num2: INTEGER
DECLARE num3: INTEGER

num1 ← 10
num2 ← 20
num3 ← 0

OUTPUT "Original values: num1 = ", num1, ", num2 = ", num2
num3 ← AddNumbers(num1, num2)
OUTPUT "Addition result: ", num3

CALL SwapValues(num1, num2)
OUTPUT "Swapped values: num1 = ", num1, ", num2 = ", num2

FUNCTION AddNumbers(a: INTEGER, b: INTEGER) RETURNS INTEGER
    DECLARE result: INTEGER //Local Variable
    result ← a + b
    RETURN result
ENDFUNCTION

PROCEDURE SwapValues(ByRef x: INTEGER, ByRef y: INTEGER)
    DECLARE temp: INTEGER //Local Variable
    temp ← x
    x ← y
    y ← temp
ENDPROCEDURE
```

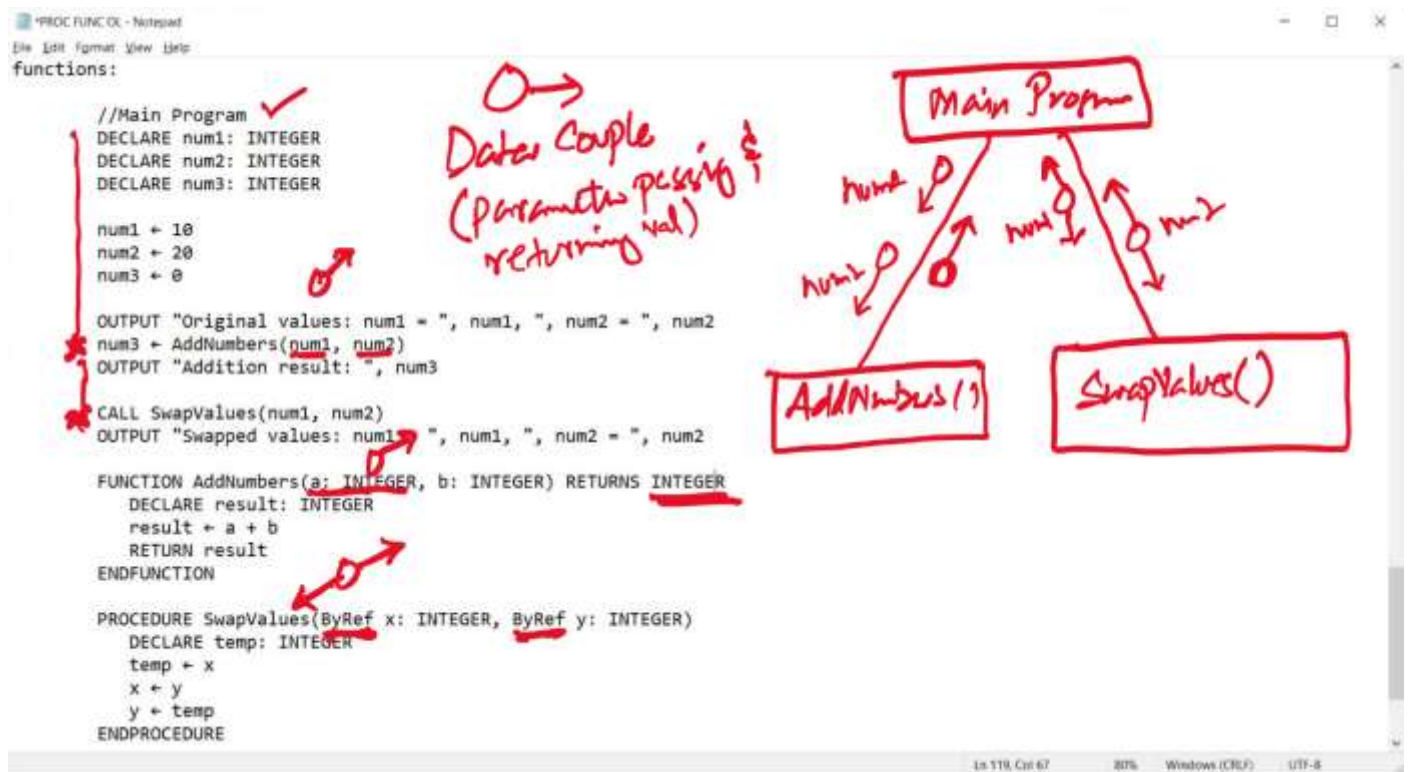
In this example, AddNumbers() is a function that takes two INTEGER parameters (a and b) and returns their sum as an INTEGER. SwapValues() is a procedure that takes two INTEGER parameters (x and y) and swapped values. Since values were received ByRef the swap of x and y in procedure SwapValues() made both num1 and num2 changed as well, So, after calling SwapValue() procedure both num1 and num2 swap their values.

The example demonstrates how to pass arguments to functions/procedures and use their returned values in expressions. Note that the passing mechanism used in this example is pass by value as well as by reference.

Structured Diagram

A structure diagram (or structure chart) is a graphical representation of the organization and hierarchy of procedures and functions in a program. It visually displays the relationships between different components of the program, making it easier to understand the program's overall structure.

Here's a structure diagram for the examples provided above:



In this diagram, the main program is the top-level component, which calls the `AddNumbers()` and `SwapValues()` functions. The diagram shows that both functions are separate components and are directly called from the main program. There are no additional layers or nested calls in this example.

In the previous code of `DefaultLine()` and `Line()`, the main program is the top-level component, which conditionally calls either the `DefaultLine()` procedure or the `Line` procedure. The `DefaultLine()` procedure, in turn, calls the `Line` procedure. This structure chart shows the hierarchy and relationships between the procedures in the program.

Structure diagrams can be more complex for larger programs with multiple layers of nested procedure or function calls. They can be a useful tool for visualizing and understanding the organization and flow of a program.

Global and local variables

Global and local variables are two types of variables in programming languages that differ in their scope and lifetime. I will explain both types and their characteristics below:

Global Variables:

- Global variables are declared outside any function or procedure, typically at the beginning of the program or in a separate module.
- They have a global scope, meaning they can be accessed and modified from any part of the program, including within functions and procedures.
- Global variables have a longer lifetime, as they exist for the entire duration of the program's execution. This means their values are retained even after a function or procedure has finished executing.
- While global variables can be convenient for sharing data across different parts of a program, their unrestricted access can lead to unintended side effects and make the program harder to maintain and debug.

Local Variables:

- Local variables are declared within a function or procedure.
- They have a local scope, meaning they can only be accessed and modified within the function or procedure they are declared in.
- Local variables have a shorter lifetime, as they are created when the function or procedure is called and destroyed when it finishes executing. This means their values are not retained between calls to the function or procedure.
- Using local variables can make the program more modular and easier to maintain, as they limit the impact of changes and prevent unintended side effects.

Here's an example demonstrating the use of global and local variables:

```
DECLARE globalVar: INTEGER
globalVar ← 5

FUNCTION MultiplyByGlobalVar(localVar: INTEGER) RETURNS INTEGER
    RETURN localVar * globalVar
ENDFUNCTION

PROCEDURE DisplayGlobalAndLocal()
    DECLARE localVar: INTEGER ← 10
    OUTPUT "Global variable: ", globalVar, ", Local variable: ", localVar
ENDPROCEDURE

CALL DisplayGlobalAndLocal()

OUTPUT "Multiplication result: ", MultiplyByGlobalVar(3)
```

In this example, `globalVar` is a global variable that can be accessed and used by both the `MultiplyByGlobalVar()` function and the `DisplayGlobalAndLocal()` procedure. In contrast, `localVar` in the `DisplayGlobalAndLocal()` procedure is a local variable and can only be accessed within that procedure.

Challenge yourself questions

Open-ended Questions

1. What are the main differences between procedures and functions, and when should you use one over the other?
2. How do parameters help make procedures and functions more adaptable and reusable? Provide an example.
3. What are the advantages and disadvantages of using global variables compared to local variables in a program?

Close-ended Questions:

1. Does a procedure return a value? (Yes/No)
2. Is it necessary to use the keyword CALL when calling a function? (Yes/No)
3. Do local variables keep their values between different calls to the function or procedure they are declared in? (Yes/No)

Fill-in-the-blanks Questions:

1. A _____ performs a specific task but does not return any value.
2. A _____ performs a specific task and returns a value.
3. When using a procedure with parameters, the provided values are called _____.
4. If a variable is declared within a function or procedure, it is called a _____ variable.
5. If a variable is declared outside any function or procedure, it is called a _____ variable.
6. The _____ keyword is used within a function to specify the value to be returned.
7. A function should only be called as part of a(n) _____.
8. The keyword _____ should not be used when calling a function.
9. The method of passing a copy of the value to the procedure or function is called pass by _____.
10. The method of passing a reference (memory address) to the original value to the procedure or function is called pass by _____.

Scenario-based Questions

1. Imagine you are creating a program that calculates the area and perimeter of different shapes like rectangles, triangles, and circles. How would you organize your code using procedures and functions to make it easy to understand and maintain?
2. Consider a program where users can input two numbers and choose a mathematical operation (addition, subtraction, multiplication, or division) to perform on them. How would you design the program using functions, and what parameters would be needed for each function?
3. You are developing a game with multiple levels, and each level has different rules and ways to earn points. How would you use procedures, functions, and parameters to create a well-organized program that can handle the different levels and their requirements?

Answers

Open-ended questions:

1. Procedures perform specific tasks without returning a value, while functions perform specific tasks and return a value. You should use a procedure when the task doesn't require a value to be returned, and a function when you need to return a value based on the task's results.
2. Parameters make procedures and functions more adaptable and reusable because they allow the same code to work with different inputs. For example, a function to calculate the area of a rectangle can take two parameters (length and width) and be used for different rectangle sizes.
3. Global variables can be accessed and modified from any part of the program, making it easy to share data across different components. However, their unrestricted access can lead to unintended side effects and make the program harder to maintain and debug. Local variables are limited in scope and can only be accessed within the function or procedure they are declared in, which makes the program more modular and easier to maintain.

Close-ended questions:

1. No
2. No
3. No

Fill-in-the-blanks questions:

1. procedure
2. function
3. arguments
4. local
5. global
6. RETURN
7. expression
8. CALL
9. value
10. reference

Scenario-based questions:

1. You can create separate functions for calculating the area and perimeter of each shape. For example, create functions like `AreaRectangle`, `PerimeterRectangle`, `AreaTriangle`, `PerimeterTriangle`, etc. Each function would take the appropriate parameters (e.g., length and width for rectangles, base and height for triangles) and return the calculated value.
2. Design the program with separate functions for each mathematical operation, such as `Add`, `Subtract`, `Multiply`, and `Divide`. Each function would take two parameters (the two numbers to be operated on) and return the result of the chosen operation. The main program would read the user's input, call the appropriate function based on the chosen operation, and display the result.
3. Use procedures and functions to modularize the game's code. Create separate procedures or functions for each level's rules and scoring systems, such as `Level1Rules`, `Level2Rules`, and so on. Use parameters to pass in any data needed for each level, like the player's current score, remaining lives, or other game-specific data. The main program would call the appropriate procedure or function based on the current level, keeping the code organized and easier to maintain.