Chapter 3:
Procedures

introduction

Procedures are a fundamental concept in programming that allow you to organize and structure your code into reusable blocks of instructions. They provide a way to encapsulate a sequence of operations or calculations that can be called and executed multiple times from different parts of a program. Procedures help improve code readability, promote code reuse, and make programs more modular and maintainable

In programming, the program structure consists of the following components:

1. Main Program:

   The main program is the entry point of the program. It is where the execution begins and controls the overall flow of the program. The main program typically contains variable declarations, function/subroutine calls, and the main logic of the program.

2. Functions and Subroutines:

   Functions and subroutines are blocks of code that perform specific tasks and can be called from within the main program or other functions/subroutines. Functions are procedures that return a value, while subroutines are procedures that do not return a value. They encapsulate a set of instructions and can be reused at different parts of the program. Functions and subroutines help modularize the code and make it more organized and maintainable.

3. Data Blocks: Data blocks, also known as data sections or data areas, are used to declare and define variables that are used in the program. These variables can have global scope, meaning they can be accessed by multiple functions/subroutines within the program. Data blocks provide a centralized location to declare and manage data, making it easier to share information between different parts of the program.

4. Modules:

   Modules are programming units that encapsulate related functions, subroutines, and data blocks. They provide a way to organize and structure the code by grouping related components together. Modules promote code reusability, modularity, and encapsulation. They can be used to create libraries or reusable components that can be easily included in different programs

Functions:

Functions are standalone code blocks that perform a specific calculation and return a value as output. They are commonly used for computations and operations on data. Functions can take input arguments, perform operations on those arguments, and return a value as output. Generally, functions are used to perform calculations and return the result to where they are called in the program.

Here's the general syntax of a function:

```
FUNCTION function_name(arg1, arg2, ...)
        ! Local declarations
        ! Instructions
  function_name = return_value
END FUNCTION
```

- **Subroutines:**

Subroutines are standalone code blocks that perform a sequence of specific instructions. They are commonly used for repetitive tasks or data manipulation operations. Subroutines can take input arguments, perform operations on those arguments, and do not return a value as output. They are often used to perform actions without returning a specific result to where they are called in the program.

Here's the general syntax of a subroutine:

```
SUBROUTINE subroutine_name(arg1, arg2, ...)
  ! Local declarations
  ! Instructions
END SUBROUTINE
```

- **Data Blocks:**

Data blocks, also known as COMMON blocks, are used to group variables with global scope in a program. They allow sharing variables among different parts of the source code. Data blocks are typically placed at the beginning of the program, before variable declarations.

Here's an example syntax for defining a data block:

```
DATA [type] /block_name/ variable1, variable2, ..
```

- **Modules:**

Modules are program units that group related functions, subroutines, and data blocks. They help organize and modularize the source code by grouping similar functionalities. Modules can be used to create reusable function libraries or to organize a program into multiple distinct parts.

Here's an example syntax for defining a module:

```
MODULE module_name
! Variable declarations
! Definition of functions and subroutines
CONTAINS
! Definition of additional functions and subroutines
END MODULE
```

- **Advantages of Procedures:**
  Functions and subroutines offer several advantages in programming:

- Reusability: Functions and subroutines can be called at multiple places in a program, allowing code reuse and avoiding duplication.
- Modularity: Modules help organize source code into separate units, facilitating program maintenance and understanding.
- Readability: Using functions and subroutines helps divide a complex program into smaller, more manageable parts, making the code more readable and easier to understand.
- Encapsulation: Functions and subroutines allow encapsulating specific functionalities, making code management and modification easier.

`These basic concepts of functions, subroutines, data blocks, and modules are commonly used in programming. They allow you to organize and structure your code effectively, promoting reusability, modularity, and readability.

- **Program 1: Factorial Calculation Using a Function**

```
PROGRAM Factorial
IMPLICIT NONE
INTEGER :: num, result
INTEGER FUNCTION factorial(n)
INTEGER :: n
IF (n == 0 .OR. n == 1) THEN
factorial = 1
ELSE
factorial = n * factorial(n - 1)
END IF
END FUNCTION factorial
WRITE(*,*) "Enter a number: "
READ(*,*) num
result = factorial(num)
WRITE(*,*) "Factorial of ", num, " is ", result
END PROGRAM Factorial
```

In this Fortran program, we define a function called factorial that takes an integer n as an argument and calculates the factorial of n. The function uses recursion to calculate the factorial by multiplying n with the factorial of n-1 until n reaches 0 or 1. The main program prompts the user to enter a number, calls the factorial function with the input number, and prints the factorial result.