# Computer Science 9608 (Notes)
# Chapter: 2.3 Programming

## Topic: 2.3.6 Structured programming

Procedural programs are ones in which instructions are executed in the order defined by the programmer.

Procedural languages are often referred to as third generation languages and include FORTRAN, ALGOL, COBOL, BASIC, and PASCAL.

**Statement**

A statement is a single instruction in a program, which can be converted into machine code and executed.

In most languages a statement is written on a single line, but some languages allow multiple lines for single statements.

Examples of statements are:

- `DIM name As String`
- `A=X*X`
- `While x < 10`

**Subroutine**

A subroutine is a self-contained section of program code that performs a specific task, as part of the main program.

**Procedure**

A procedure is a subroutine that performs a specific task without returning a value to the part of the program from which it was called.

**Function**

A function is a subroutine that performs a specific task and returns a value to the part of the program from which it was called.

Note that a function is 'called' by writing it on the right hand side of an assignment statement.

## Topic: 2.3.6 Structured programming

**Parameter**

A parameter is a value that is 'received' in a subroutine (procedure or function).

The subroutine uses the value of the parameter within its execution. The action of the subroutine will be different depending upon the parameters that it is passed.

Parameters are placed in parenthesis after the subroutine name. For example:

`Square(5)` 'passes the parameter 5 – returns 25

`Square(8)` 'passes the parameter 8 – returns 64

`Square(x)` 'passes the value of the variable x

**Use subroutines to modularise the solution to a problem**

**Subroutine/sub-program**

A subroutine is a self-contained section of program code which performs a specific task and is referenced by a name.

A subroutine resembles a standard program in that it will contain its own local variables, data types, labels and constant declarations.

There are two types of subroutine. These are procedures and functions.

- Procedures are subroutines that input, output or manipulate data in some way.
- Functions are subroutines that return a value to the main program.

A subroutine is executed whenever its name is encountered in the executable part of the main program. The execution of a subroutine by referencing its name in the main program is termed 'calling' the subroutine.

## Topic: 2.3.6 Structured programming

The benefits of using procedures and functions are that:

- The same lines of code are re-used whenever they are needed – they do not have to be repeated in different sections of the program.
- A procedure or function can be tested/improved/rewritten independently of other procedures or functions.
- It is easy to share procedures and functions with other programs – they can be incorporated into library files which are then 'linked' to the main program.
- A programmer can create their own routines that can be called in the same way as any built-in command.

**Sub Procedures (Visual Basic)**

A Sub procedure is a series of Visual Basic statements enclosed by the Sub and End Sub statements. The Sub procedure performs a task and then returns control to the calling code, but it does not return a value to the calling code.

Each time the procedure is called, its statements are executed, starting with the first executable statement after the Sub statement and ending with the first End Sub, Exit Sub, or Return **statement** encountered.

You can define a Sub procedure in modules, classes, and structures. By default, it is Public, which means you can call it from anywhere in your application that has access to the module, class, or structure in which you defined it. The term, method, describes a Sub or Function procedure that is accessed from outside its defining module, class, or structure. For more information, see Procedures in Visual Basic.

A Sub procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

## Topic: 2.3.6 Structured programming

**Declaration Syntax**

The syntax for declaring a Sub procedure is as follows:

```
[ modifiers ] Sub subname [( parameterlist )]
' Statements of the Sub procedure.
End Sub
```

The modifiers can specify access level and information about overloading, overriding, sharing, and shadowing. For more information, see Sub Statement (Visual Basic).

**Parameter Declaration**

You declare each procedure parameter similarly to how you declare a variable, specifying the parameter name and data type. You can also specify the passing mechanism, and whether the parameter is optional or a parameter array.

The syntax for each parameter in the parameter list is as follows:

```
[Optional] [ByVal | ByRef] [ParamArray] parametername As datatype
```

If the parameter is optional, you must also supply a default value as part of its declaration. The syntax for specifying a default value is as follows:

```
Optional [ByVal | ByRef] parametername As datatype = defaultvalue
```

Parameters as Local Variables

When control passes to the procedure, each parameter is treated as a local variable. This means that its lifetime is the same as that of the procedure, and its scope is the whole procedure.

## Topic: 2.3.6 Structured programming

**Calling Syntax**

You invoke a Sub procedure explicitly with a stand-alone calling statement. You cannot call it by using its name in an expression. You must provide values for all arguments that are not optional, and you must enclose the argument list in parentheses. If noarguments are supplied, you can optionally omit the parentheses. The use of
the Call keyword is optional but not recommended.

The syntax for a call to a Sub procedure is as follows:

```
[Call] subname [( argumentlist )]
```

**Illustration of Declaration and Call**

The following Sub procedure tells the computer operator which task the application is about to perform, and also displays a time stamp. Instead of duplicating this code at the start of every task, the application just calls tellOperator from various locations. Each call passes a string in the task argument that identifies the task being started.

VB

```
Sub tellOperator(ByVal task As String)
 Dim stamp As Date
 stamp = TimeOfDay()
 MsgBox("Starting "& task &" at "&CStr(stamp))
End Sub
```

```
The following example shows a typical call to tellOperator.
```

VB

```
tellOperator("file update")
```

**Function Procedures (Visual Basic)**

A Function procedure is a series of Visual Basic statements enclosed by the Function and End Function statements. The Function procedure performs a task and then returns control to the calling code. When it returns control, it also returns a value to the calling code.

Each time the procedure is called, its statements run, starting with the first executable statement after the Function statement and ending with the first End Function, Exit Function, or Return statement encountered.

## Topic: 2.3.6 Structured programming

You can define a Function procedure in a module, class, or structure. It is Public by default, which means you can call it from anywhere in your application that has access to the module, class, or structure in which you defined it.

A Function procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

**Declaration Syntax**

The syntax for declaring a Function procedure is as follows:

VB

```
[Modifiers] FunctionFunctionName [(ParameterList)] AsReturnType
    [Statements]
EndFunction
```

**Data Type**

Every Function procedure has a data type, just as every variable does. This data type is specified by the As clause in the Function statement, and it determines the data type of the value the function returns to the calling code. The following sample declarations illustrate this.

VB

```
Function yesterday() AsDate
EndFunction

FunctionfindSqrt(ByVal radicand AsSingle) AsSingle
EndFunction
```

## Topic: 2.3.6 Structured programming

Returning Values

The value a Function procedure sends back to the calling code is called its return value. The procedure returns this value in one of two ways:

- It uses the Return statement to specify the return value, and returns control immediately to the calling program. The following example illustrates this.

  VB

```
Function FunctionName [(ParameterList)] As ReturnType
' The following statement immediately transfers control back
' to the calling code and returns the value of Expression.
Return Expression
EndFunction
```

- It assigns a value to its own function name in one or more statements of the procedure. Control does not return to the calling program until an Exit Function or End Functionstatement is executed. The following example illustrates this.

  VB

```
Function FunctionName [(ParameterList)] As ReturnType
‘ The following statement does not transfer control back to the calling code.
FunctionName = Expression
' When control returns to the calling code, Expression is the return value.
End Function
```

The advantage of assigning the return value to the function name is that control does not return from the procedure until it encounters an Exit Function or End Function statement. This allows you to assign a preliminary value and adjust it later if necessary.

## Topic: 2.3.6 Structured programming

**Calling Syntax**

You invoke a Function procedure by including its name and arguments either on the right side of an assignment statement or in an expression. You must provide values for all arguments that are not optional, and you must enclose the argument list in parentheses. If no arguments are supplied, you can optionally omit the parentheses.

The syntax for a call to a Function procedure is as follows:

```
lvalue = functionname [( argumentlist )]
If (( functionname [( argumentlist )] / 3) <= expression ) Then
```

When you call a Function procedure, you do not have to use its return value. If you do not, all the actions of the function are performed, but the return value is ignored." MsgBox" is often called in this manner.

**Illustration of Declaration and Call**

The following Function procedure calculates the longest side, or hypotenuse, of a right triangle, given the values for the other two sides.

VB

```
Functionhypotenuse(ByVal side1 As Single, ByVal side2 As Single) As Single
ReturnMath.Sqrt((side1 ^ 2) + (side2 ^ 2))
End Function
```

The following example shows a typical call to hypotenuse.

VB

```
DimtestLength, testHypotenuseAs Single
testHypotenuse = hypotenuse(testLength, 10.7)
```

Zak ZAFAR ALI KHAN

## Topic: 2.3.6 Structured programming

**Passing Arguments by Value and by Reference (Visual Basic)**

In Visual Basic, you can pass an argument to a procedure by value or by reference. This is known as the passing mechanism, and it determines whether the procedure can modify the programming element underlying the argument in the calling code. The procedure declaration determines the passing mechanism for each parameter by specifying the ByVal (Visual Basic) or ByRef (Visual Basic) keyword.

**Distinctions**

When passing an argument to a procedure, be aware of several different distinctions that interact with each other:

- Whether the underlying programming element is modifiable or not
- Whether the argument itself is modifiable or not
- Whether the argument is being passed by value or by reference
- Whether the argument data type is a value type or a reference type

For more information, see Differences Between Modifiable and Nonmodifiable Arguments (Visual Basic) and Differences Between Passing an Argument By Value and By Reference (Visual Basic).

**Choice of Passing Mechanism**

You should choose the passing mechanism carefully for each argument.

- Protection.
  In choosing between the two passing mechanisms, the most important criterion is the exposure of calling variables to change. The advantage of passing an argument ByRef is that the procedure can return a value to the calling code through that argument. The advantage of passing an argument ByVal is that it protects a variable from being changed by the procedure.

- Performance.
  Although the passing mechanism can affect the performance of your code, the difference is usually minor. One exception to this is a value type passed ByVal. In this case, Visual Basic copies the entire data contents of the argument. Therefore, for a large value type such as a structure, it can be more efficient to pass it ByRef.

## Topic: 2.3.6 Structured programming

### Determination of the Passing Mechanism

The procedure declaration specifies the passing mechanism for each parameter. The calling code can't override a ByVal mechanism.
If a parameter is declared with ByRef, the calling code can force the mechanism to ByVal by enclosing the argument name in parentheses in the call. For more information, see How to: Force an Argument to Be Passed by Value (Visual Basic).
The default in Visual Basic is to pass arguments by value.

### When to Pass an Argument by Value

- If the calling code element underlying the argument is a nonmodifiable element, declare the corresponding parameter ByVal (Visual Basic). No code can change the value of a non-modifiable element.

- If the underlying element is modifiable, but you do not want the procedure to be able to change its value, declare the parameter ByVal. Only the calling code can change the value of a modifiable element passed by value.

### When to Pass an Argument by Reference

- If the procedure has a genuine need to change the underlying element in the calling code, declare the corresponding parameter ByRef (Visual Basic).

- If the correct execution of the code depends on the procedure changing the underlying element in the calling code, declare the parameter ByRef. If you pass it by value, or if the calling code overrides the ByRef passing mechanism by enclosing the argument in parentheses, the procedure call might produce unexpected results.

**Example:**

Description:The following example illustrates when to pass arguments by value and when to pass them by reference. Procedure Calculate has both a ByVal and a ByRef parameter. Given an interest rate, rate, and a sum of money, debt, the task of the procedure is to calculate a new value for debt that is the result of applying the interest rate to the original value of debt. Because debt is a ByRef parameter, the new total is reflected in the value of the argument in the calling code that corresponds to debt. Parameter rate is a ByVal parameter because Calculate should not change its value.

## Topic: 2.3.6 Structured programming

**Code**

```
Module Module1

Sub Main()
' Two interest rates are declared, one a constant and one a
' variable.
Const highRate As Double = 12.5
Dim lowRate = highRate * 0.6

Dim initialDebt = 4999.99
' Make a copy of the original value of the debt.
Dim debtWithInterest = initialDebt

' Calculate the total debt with the high interest rate applied.
' ArgumenthighRate is a constant, which is appropriate for a
' ByVal parameter. Argument debtWithInterest must be a variable
' because the procedure will change its value to the calculated
' total with interest applied.
Calculate(highRate, debtWithInterest)

' Format the result to represent currency, and display it.
Dim debtString = Format(debtWithInterest, "C")
Console.WriteLine("What I owe with high interest: "&debtString)

' Repeat the process with lowRate. Argument lowRate is not a
' constant, but the ByVal parameter protects it from accidental
' or intentional change by the procedure.

' SetdebtWithInterest back to the original value.
debtWithInterest = initialDebt
Calculate(lowRate, debtWithInterest)
debtString = Format(debtWithInterest, "C")
Console.WriteLine("What I owe with low interest:  "&debtString)
End Sub

' Parameter rate is a ByVal parameter because the procedure should
' not change the value of the corresponding argument in the
' calling code.

' The calculated value of the debt parameter, however, should be
' reflected in the value of the corresponding argument in the
' calling code. Therefore, it must be declared ByRef.
Sub Calculate(ByVal rate As Double, ByRef debt As Double)
debt = debt + (debt * rate / 100)
End Sub
End Module
```