Higher

Computing Science

Software Design and Development

Computational Constructs

Name:_____

Contents

Computational Constructs

Computational constructs are used in the writing of programs.

At Higher level, the constructs you must make use of are:

- Sequential File Operations
- Parameter Passing
- Subprograms (Procedures)
- Subprograms (Functions)
- File Handling
- Pre-Defined Functions

A table of computational construct examples can be found on the back page of this book.

Pre-defined Functions

There are four new types of pre-defined function that you have to learn.

- Substrings
- Convert Floating Point to Integer
- Convert Characters to ASCII values
- Modulus (find remainder from division calculation)

Predefined functions are commands that can be used in any program to carry out a **calculation** or **format text and numbers** in a particular way.

They are like **shortcuts** as they save you having to write your own lines of code to carry out the function's task.

Format of a function

answerVariable **=** ***functionName*** (parameter1, parameter2, …)

This can be any variable and is used to store the answer returned by the function.

The variable data type must match the type of value returned by the function.

This is the name of the pre-defined to be used.

See the names of functions below.

Parameters are the inputs required by the function.

A function can have none, one or more parameters – it depends on the function.

Function	**Purpose**	**Returned Data Type**	**Parameters**
To convert floating-point numbers to integers			
INT	Converts a floating point number(decimal) to integer	Integer	• a floating point number
To convert from ASCII to Character and vice versa			
ASC	Returns the ASCII value of a character	Integer	• a single character
CHR	Returns the character of an ASCII value	Character	• an integer value
Modulus			
MOD	Returns the remainder of a division operation	Integer	
To create substrings			
LEFT	Returns a substring of characters starting from the left	String	• Main string • Number of characters
RIGHT	Returns a substring of characters starting from the right	String	• Main string • Number of characters
MID	Returns a substring of characters starting from the middle	String	• Main string • Number of characters • Starting point

Int: Use a function to convert a floating point number to an integer

```
IntegerNumber = INT(decimalNumber )
```

Asc: Use a function to return the ASCII code value of a character

```
asciiValue = ASC(myCharacter)
```

Chr: Use a function to return the character of an ASCII value

```
myCharacter = CHR(asciiValue)
```

Mod: Use a function to return the remainder of 7 / 2

```
remainder = 7 MOD 2
```

Left: Use a function to extract the first 4 letters from a string

```
substring = LEFT(myString, 4)
```

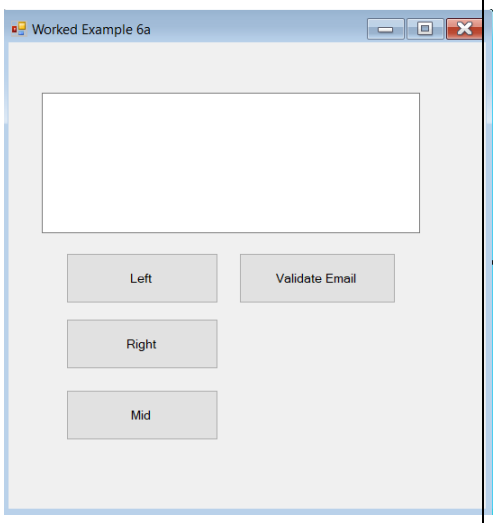
Right: Use a function to extract the last 3 letters from a string

```
substring = RIGHT(myString, 3)
```

Mid: Use a function to extract characters 4 to 6 from a string

```
substring = MID(myString, 4, 3)
```

Worked Example 6a - Substrings



A substring is when part of a larger piece of text is extracted. Substrings can be extracted from the beginning, ending or middle of a larger string.

This example will demonstrate how the substring functions (Left, Right and Mid) can be used to extract substrings.

There is also an example of how the mid function can be used to validate an email address.

Left

```
Private Sub btnLeft_Click(sender As Object, e As EventArgs) Handles btnLeft.Click


  Dim mainString As String
  Dim subString As String
  Dim numChars As Integer


  mainString = InputBox("Enter some text")
  numChars = InputBox("Enter number of characters to extract")


  subString = Microsoft.VisualBasic.Left(mainString, numChars)


  txtOutput.AppendText(numChars & "letters from the left of " & mainString & " is " &
subString)
  txtOutput.AppendText(vbNewLine)


End Sub
```

Get substring from the left

Right

```
Private Sub btnRight_Click(sender As Object, e As EventArgs) Handles btnRight.Click


  Dim mainString As String
  Dim subString As String
  Dim numChars As Integer


  mainString = InputBox("Enter some text")
  numChars = InputBox("Enter number of characters to extract")


  subString = Microsoft.VisualBasic.Right(mainString, numChars)


 txtOutput.AppendText(numChars & "letters from the Right of " & mainString & " is " &
subString)
 txtOutput.AppendText(vbNewLine)
End Sub
```

Get substring from the right

Mid

```
Private Sub btnMid_Click(sender As Object, e As EventArgs) Handles btnMid.Click

Dim mainString As String
Dim subString As String

Dim startChar As Integer
Dim numChars As Integer


mainString = InputBox("Enter some text")
startChar = InputBox("What position do you want to start from?")
numChars = InputBox("Enter number of characters to extract")

subString = Mid(mainString, startChar, numChars)


txtOutput.AppendText(numChars & "letters from the middle of " & mainString & " is " &
subString)
txtOutput.AppendText(vbNewLine)

End Sub
```

Get substring from the middle

The following example shows how the mid function can be used to validate an email address to ensure is contains an @ and dots.

```
Private Sub btnValidate_Click(sender As Object, e As EventArgs) Handles
btnValidate.Click

        Dim email As String
        Dim textLength As Integer
        Dim atCounter As Integer
        Dim dotCounter As Integer

        atCounter = 0
        dotCounter = 0
        email = InputBox("Please enter email address")
        textLength = Len(email)


        For index = 1 To textLength

            If Mid(email, index, 1) = "@" Then
                atCounter = atCounter + 1
            End If

            If Mid(email, index, 1) = "." Then
                dotCounter = dotCounter + 1
            End If

        Next


        If dotCounter >= 1 And atCounter = 1 Then
            MsgBox("email address is valid")
        Else
            MsgBox("No @ or not enough dots in email address")
        End If

End Sub
End Class
```

From the 'email' string
Start at a different char each loop (index)
Extract one char each time to check (1)

Practise Tasks

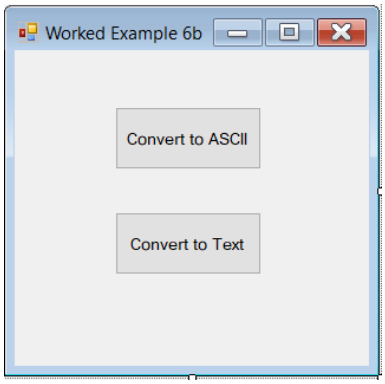
1. Create a program to count the number of users whose first name starts with the letter A. The program should allow 10 users to enter their full first name. At the end a message should indicate how many of the user's names started with the letter A.

2. A program is required to store the names of 5 football teams. The program should also store a yes or a no against each team to indicate whether is contains the word "north" anywhere in the name. Once all names have been entered, the program should display the full list of teams together with a yes or no.

Worked Example 6b - Convert ASCII to Char / Char to ASCII

Characters can be converted into ASCII integer values and vice versa. There are two pre-defined functions which carry out these tasks:

- ASC
- CHR



ASC

```
Private Sub btnASCII_Click(sender As Object, e As EventArgs) Handles btnASCII.Click

        Dim myLetter As Char
        Dim asciiValue As Integer


        myLetter = InputBox("Enter a character")

        asciiValue = Asc(myLetter)

        MsgBox("ASCII value for " & myLetter & " is " & asciiValue)

End Sub
```

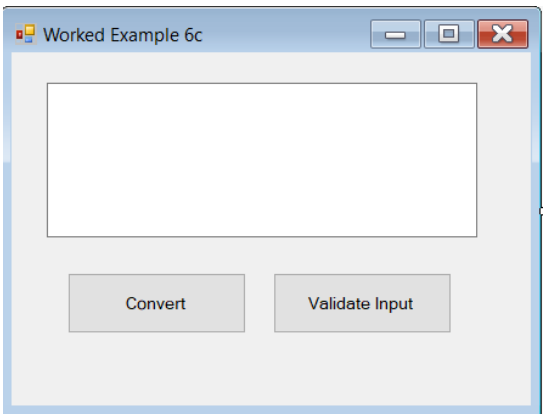
CHR

```
Private Sub btnChar_Click(sender As Object, e As EventArgs) Handles btnChar.Click

        Dim myLetter As Char
        Dim asciiValue As Integer


        asciiValue = InputBox("Enter a number")

        myLetter = Chr(asciiValue)

        MsgBox(asciiValue & " is the ASCII value for " & myLetter)

End Sub
```

Worked Example 6c - Convert Floating Point to Integer

Floating point numbers (Single) values can have the numbers after the decimal point removed by converting them to integers using the INT pre-defined function. INT does **not** round a number – it removes everything after the decimal point.

This example demonstrates a simple conversion and also shows how INT can be used to validate that a whole number has been typed in by the user.



INT

```
Private Sub btnConvert_Click(sender As Object, e As EventArgs) Handles
btnConvert.Click

        Dim myNumber As Single
        Dim outputNumber As Single


        myNumber = InputBox("Please enter a real number")
        outputNumber = Int(myNumber)

        txtOutput.AppendText(myNumber & " has changed to " & outputNumber)


End Sub
```

This code asks the user to enter a whole number. By comparing the input with the INT of what was entered, the program can check that the user entered a whole number.

```
Private Sub btnValidate_Click(sender As Object, e As EventArgs) Handles
btnValidate.Click


        Dim userNumber As Single


        Do
            userNumber = InputBox("Please enter a whole number")
            If userNumber <> Int(userNumber) Then
                MsgBox("Please enter a whole number")
            End If
        Loop Until userNumber = Int(userNumber)

        txtOutput.AppendText("Thank you - whole number entered")


End Sub
End Class
```

Practise Tasks

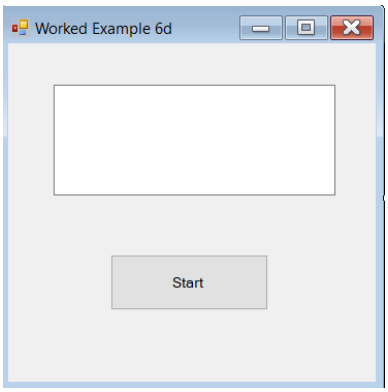
3. A program is required to convert ascii values into words. The user should be prompted to repeatedly enter ascii values, one at a time until they enter the ascii value for a full stop. The program should convert each value into a character and add it to a stored string.

User input should be checked to ensure only printable ascii characters are accepted and also that the number entered is a whole number.

When a full stop is detected, the program should output the final string to display the message.

Worked Example 6d - Modulus

Modulus is used to calculate the remainder when a division calculation is performed.



> ***e.g.***
> 15 *mod* 6 will produce the answer 3 because 15 ÷ 6 is 2 remainder 3.

Modulus is used to calculate the remainder when a division calculation is performed

```
Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles btnStart.Click

  Dim dividend As Integer
  Dim divisor As Integer
  Dim remainder As Integer


  dividend = InputBox("Please enter the dividend")
  divisor = InputBox("Please enter the divisor")

  remainder = dividend Mod divisor

  txtOutput.AppendText("The remainder of " & dividend & " divided by " & divisor & "
  is " &
                         remainder)


End Sub
End Class
```

Practise Questions

Question 1 (SQP Qu 16)

Using a programming language of your choice, state the pre-defined function used to convert (2):

(i) Character to ASCII _____

(ii) ASCII to Character_____

Question 2 (2019 Qu 2)

A string variable called month has been assigned the value ‘April’ and another string variable called year has been assigned the value ‘2019’ as shown below.

Line 1 DECLARE month INITIALLY "April"

Line 2 DECLARE year INITIALLY "2019"

Line 3 _____

The variable shortDate is to be assigned the value ‘Apr19’ using substring operations. Using a programming language of your choice write line 3. (3)

Sub-Programs

Sub-programs are named blocks of code which can be run from within another part of the program.

When a sub-program is used like this we say it is “called”.

Sub-programs can be called from any part of the program and can be used over again.

A sub-program may be called several times during the execution of a single program.

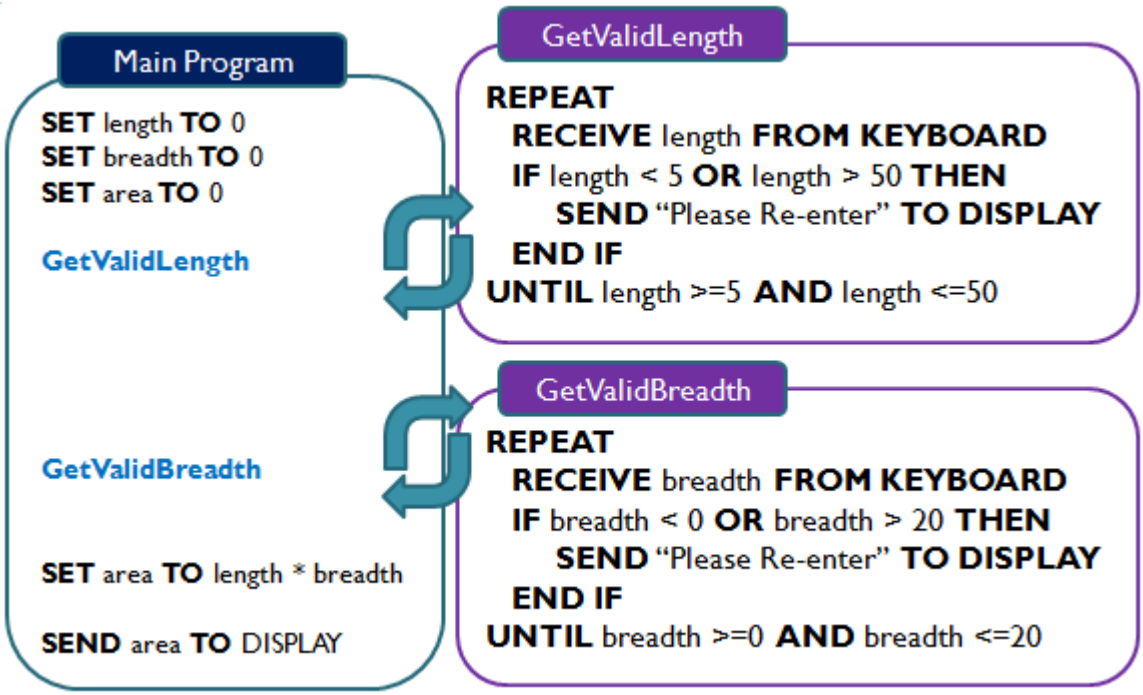
Example

This program works out the area of a room in a building.

Main Program

```
SET length TO 0
SET breadth TO 0
SET area TO 0

REPEAT
  RECEIVE length FROM KEYBOARD
  IF length < 5 OR length > 50 THEN
     SEND “Please Re-enter” TO DISPLAY
  END IF
UNTIL length >=5 AND length <=50

REPEAT
  RECEIVE breadth FROM KEYBOARD
  IF breadth < 0 OR breadth > 20 THEN
     SEND “Please Re-enter” TO DISPLAY
  END IF
UNTIL breadth >=0 AND breadth <=20

SET area TO length * breadth
SEND area TO DISPLAY
```

The Input Validation lines of code can be put into **sub-programs** and **called** by the main program.



Why Create Sub-Programs?

- Creating sub-programs makes the code more **modular** and readable.
- Modular code allows sections of code to be self-contained.
- Different sub-programs can be developed by different programmers without variable name clashes
- Sub-programs can be re-used without any extra coding which saves time.
- Easier to identify errors.

Types of Sub-Program

There are two types of sub-program that can be used in procedural languages.

- **Procedures**
- **Functions**

Procedures and functions are self-contained sections of code that execute a sequence of commands.

They are both given meaningful identifiers (names) which are used to call them.

Procedures

When procedures are called, variables (parameters) to be passed **in or out** of the procedure are **stated in brackets**.

Procedures can pass **any number of parameters** in or out (or sometimes none).

Example

Consider creating the GetValidLength sub-program as a **procedure.**

Main Program

```
SET UserLen TO 0

CALL GetValidLength (UserLen)
```

The ***UserLen*** variable is declared in the main program.

When the procedure is called, ***UserLen*** is

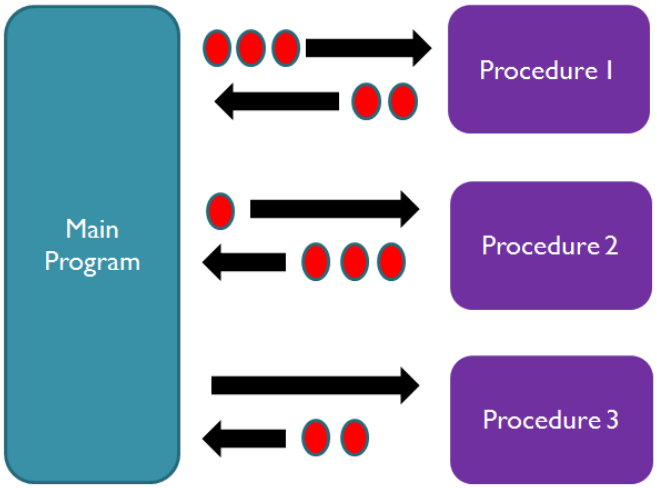
When it is called, the **GetValidLength** procedure executes its lines of code in order.

The ***length*** parameter is changed and **passed back** to the main program.

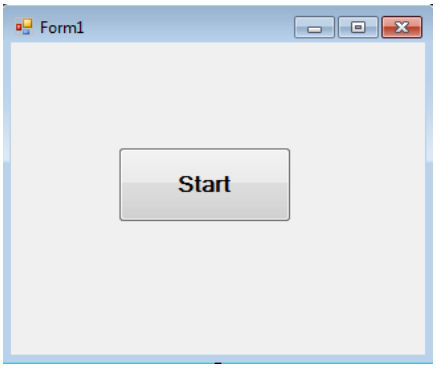
GetValidLength

```
PROCEDURE GetValidLength (length)
REPEAT
  RECEIVE length FROM KEYBOARD
  IF length < 5 OR length > 50 THEN
     SEND "Please Re-enter" TO DISPLAY
  END IF
UNTIL length >=5 AND length <=50
END PROCEDURE
```

Any number of parameters (variables) can be passed **in or out** of procedures.



Worked Example 7 - Procedures



This example demonstrates how to use procedures in a program that calculates the area and perimeter of a rectangle.

It is a very simple example to show where code should be placed, how procedures are called and how parameter passing works.

Program Specification

A program is required to all the user to enter the dimensions of a rectangle (length and breadth). Using these dimensions, the program should calculate the area and perimeter of the rectangle and then display both results on screen.

Design

Algorithm

1. Get Dimensions

2. Calculate Sizes

3. Display Sizes

Step-wise Refinements

1.1 get rectangle length from keyboard

1.2 get rectangle breadth from keyboard

2.1 set area to length * breadth

2.2 set perimeter to (length * 2) + (breadth * 2)

3.1 display “The area is”, area, “ and the perimeter is ”, perimeter

Implementation

Enter the following code. As you do so, consider the main steps of this program, identified at the design stage, in achieving its aims.

```
Public Class Form1

Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles btnStart.Click

    Dim length As Integer
    Dim breadth As Integer
    Dim area As Integer
    Dim perimeter As Integer

    length = 0
    breadth = 0
    area = 0
    perimeter = 0


    length = InputBox("Enter the length")
    breadth = InputBox("Enter the breadth")


    area = length * breadth
    perimeter = (length * 2) + (breadth * 2)



    MsgBox("The area is " & area & " and the perimeter is " & perimeter)

End Sub
```

Get Dimensions

Calculate Sizes

Display Sizes

We will now create a **procedure** for each of the main steps.
Add the following code beneath the code for the button but **before** the *End Class* statement.

```
Private Sub GetDimensions()


End Sub
```

```
Private Sub CalculateSizes()


End Sub
```

```
Private Sub DisplaySizes()



End Sub
```

Now we have to **move** the relevant code to each procedure.

```
Private Sub GetDimensions()

        length = InputBox("Enter the length")
        breadth = InputBox("Enter the breadth")

End Sub
```

```
Private Sub CalculateSizes ()

        area = length * breadth
        perimeter = (length * 2) + (breadth * 2)


End Sub
```

```
Private Sub DisplaySizes()

        MsgBox("The area is " & area & " and the perimeter is " & perimeter)

End Sub
```

Parameters (arguments)

Next, look at the variables required by each procedure. These will now be called **parameters**.
Also, decide whether each procedure will **change** the parameter's value or not.

Procedure	Parameters	Parameter value changed by procedure?	*Explanation*
GetDimensions	length breadth	Yes Yes	Changes from 0 to a value entered by user Changes from 0 to a value entered by user
CalculateArea	length breadth area perimeter	No No Yes Yes	Remains as value entered by user Remains as value entered by user Changes from 0 to result of length * breadth Changes from 0 to total size of sides
DisplayArea	area perimeter	No No	Remains as result of length * breadth Remains as total size of sides

Any parameter **changed** by a procedure will be passed **OUT –** (indicated by ByRef)

Any parameter **not changed** by a procedure will be passed **IN** – (indicated by ByVal)

*Arrays should always be passed ByRef regardless if it will be changed or not by the procedure.

Data Flow

Using the information from the table, data flow can now be added to your algorithm as shown:

1. Get Dimensions	**OUT:** length, breadth
2. Calculate Sizes	**IN:** length, breadth **OUT:** area, perimeter
3. Display Sizes	**IN:** area, perimeter

Using the table, enter the correct parameters for each procedure in brackets next to the procedure name.

Use ***ByRef*** or ***ByVal*** to specify how each parameter will be passed (changed / not changed).

```
Private Sub GetDimensions(ByRef length, ByRef breadth)

       length = InputBox("Enter the length")
       breadth = InputBox("Enter the breadth")

End Sub
```

```
Private Sub CalculateSizes(ByVal length, ByVal breadth, ByRef area, ByRef perimeter)

       area = length * breadth
       perimeter = (length * 2) + (breadth * 2)

End Sub
```

```
Private Sub DisplaySizes(ByVal area)

       MsgBox("The area is " & area & " and the perimeter is " & perimeter)

End Sub
```

Your main **btnStart** sub-program should now only contain the following:

```
Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles btnStart.Click

        Dim length As Integer
        Dim breadth As Integer
        Dim area As Integer
        Dim perimeter As Integer

        length = 0
        breadth = 0
        area = 0
        perimeter = 0

End Sub
```

Try running your program – you should notice that nothing actually happens.

Now add the code to call the procedures you have created.

It is very important here that parameters (in brackets) are listed in the **same order** as in the procedure declaration.

```
Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles btnStart.Click

        Dim length As Integer
        Dim breadth As Integer
        Dim area As Integer
        Dim perimeter As Integer

        length = 0
        breadth = 0
        area = 0
        perimeter = 0

        Call GetDimensions(length, breadth)
        Call CalculateSizes(length, breadth, area, perimeter)
        Call DisplaySizes(area, perimeter)


End Sub
```

Add these Call statements

You can run your program now and it should work as expected.

Try:

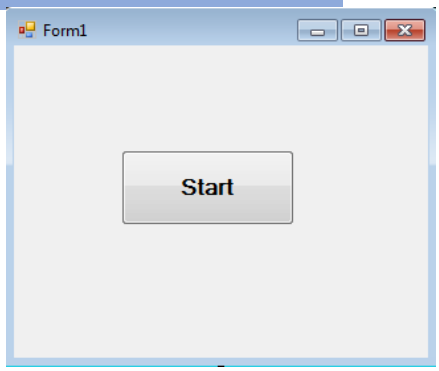
Changing the names of length, breadth and area in the **main program section only**.

Does the program still work correctly? Can you explain this?

```
Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles btnStart.Click

        Dim myLen As Integer
        Dim myBre As Integer
        Dim myArea As Integer
        Dim myPerim As Integer

        myLen = 0
        myBre = 0
        myArea = 0
        myPerim = 0

        Call GetDimensions(myLen, myBre)
        Call CalculateSizes(myLen, myBre, myArea, myPerim)
        Call DisplaySizes (myArea, myPerim)

    End Sub
```

Add these Call statements

Worked Example 8



This example demonstrates how procedures can be re-used preventing code having to be written repeatedly.

Rather than writing input validation code for each range on numbers, this example re-uses a procedure which can validate different ranges each time.

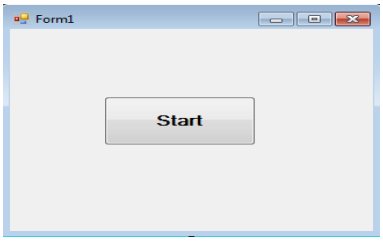
```
Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles btnStart.Click


        Dim lower As Integer
        Dim upper As Integer
        Dim userValue As Integer


        lower = InputBox("Please enter lower limit")
        upper = InputBox("Please enter upper limit")

        Call GetValidValue(lower, upper, userValue)



        lower = InputBox("Please enter a new lower limit")
        upper = InputBox("Please enter a new upper limit")

        Call GetValidValue(lower, upper, userValue)



        Call GetValidValue(20, 50, userValue)


End Sub
```

```
Private Sub GetValidValue(ByVal low, ByVal high, ByRef userValue)

        Do
            userValue = InputBox("Please enter a number between " & low & " and " &
high)
            If userValue < low Or userValue > high Then
                MsgBox("Invalid value, please re-enter")
            End If

        Loop Until userValue >= low And userValue <= high


    End Sub
```

Worked Example 9

This example demonstrates how procedures can declare local variables whose scope is limited to that procedure only.

Also, notice that the array is always passed ByRef, even when its values are not changed.




```
Public Class Form1

Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles btnStart.Click

        Dim scores(5) As Integer
        Dim average As Single

        average = 0

        Call GetValidScores(scores)
        Call CalcAverage(scores, average)
        Call DisplayAverage(average)

End Sub
```

```
Private Sub GetValidScores(ByRef scores)

        For index = 1 To 5
            Do
                scores(index) = InputBox("Please enter a score (0-50)")
                If scores(index) < 0 Or scores(index) > 50 Then
                    MsgBox("Invalid score entered")
                End If
            Loop Until scores(index) >= 0 And scores(index) <= 50
        Next

End Sub
```

scores array is passed ByRef here, even though its values are changing

```
Private Sub CalcAverage(ByRef scores, ByRef average)

        Dim total As Integer
        total = 0

        For index = 1 To 5
            total = total + scores(index)
        Next

        average = total / 5
End Sub
```

total is a local variable – it can only be used or updated within this procedure

```
Private Sub DisplayAverage(ByVal average)

        MsgBox("The average score is " & average)

End Sub
End Class
```

Functions

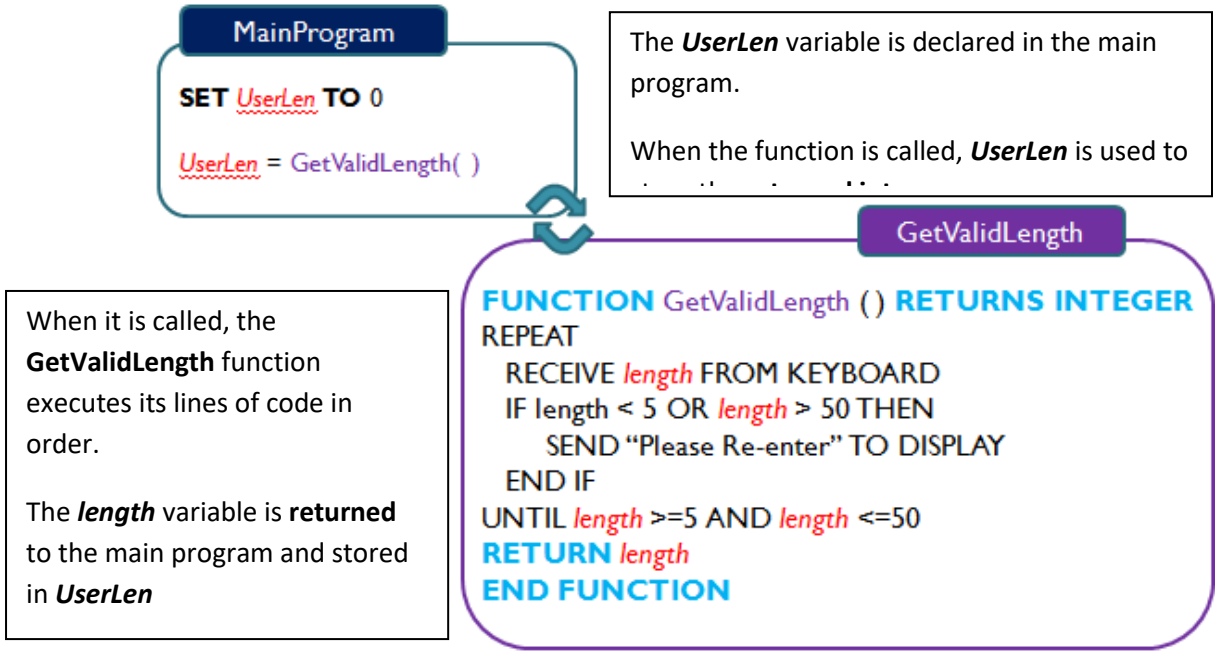
When functions are called, variables (parameters) to be passed **in only** are **stated in brackets**.

Functions can **return** only a **single value.**

The returned value from a function is **assigned to a variable** to be used in subsequent operations in the program.

Example

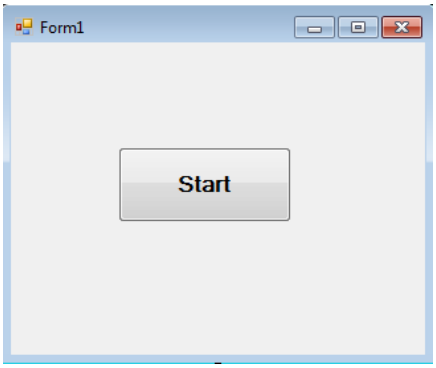
Consider creating the GetValidLength sub-program as a **function**



Worked Example 10 - Functions

This example demonstrates how to use functions in a program that calculate the area and perimeter of a rectangle.

We have to use two functions to calculate area and perimeter. Compare this to example 5 where we used one procedure to calculate the area and perimeter




```
Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles btnStart.Click

        Dim length As Integer
        Dim breadth As Integer
        Dim area As Integer
        Dim perimeter As Integer

        length = 0
        breadth = 0
        area = 0
        perimeter = 0


        length = InputBox("Enter the length")
        breadth = InputBox("Enter the breadth")


        area = CalculateArea(length, breadth)
        perimeter = CalculatePerimeter(length, breadth)


        MsgBox("The area is " & area & " and the perimeter is " & perimeter)

End Sub
```

```
Function CalculateArea(ByVal length, ByVal breadth)

        Dim recArea As Integer

        recArea = length * breadth

        Return recArea

End Function
```

```
Function CalculatePerimeter(ByVal length, ByVal breadth)

        Dim recPerimeter As Integer

        recPerimeter = (2 * length) + (2 * breadth)

        Return recPerimeter

End Function
End Class
```

Re-Using Sub-Programs

The most efficient use of sub-programs is when they can be re-used .

When coding procedures and functions, consideration should be given to making them able to solve **any related problem** rather than **one specific problem**.

e.g. a calculator that can only solve the calculation 2+2 would be very limited.

Example

The procedures below are almost identical except for the range of values they validate.

GetValidLength

```
PROCEDURE GetValidLength (length)
REPEAT
   RECEIVE length FROM KEYBOARD
   IF length < 5 OR length > 50 THEN
      SEND "Please Re-enter" TO DISPLAY
   END IF
UNTIL length >=5 AND length <=50
END PROCEDURE
```

GetValidBreadth

```
PROCEDURE GetValidBreadth (breadth)
REPEAT
   RECEIVE breadth FROM KEYBOARD
   IF breadth < 0 OR breadth > 20 THEN
      SEND "Please Re-enter" TO DISPLAY
   END IF
UNTIL breadth >=0 AND breadth <=20
END PROCEDURE
```

They could instead be made more generic by allowing the range of values to be changed each time it is called.

GetValidValue

```
PROCEDURE GetValidValue (low, high, userVal)
REPEAT
   RECEIVE userVal FROM KEYBOARD
   IF userVal < low OR userVal > high THEN
      SEND "Please Re-enter" TO DISPLAY
   END IF
UNTIL userVal >=low AND userVal <=high
END PROCEDURE
```

The GetValidValue procedure can now be called to obtain a value within **any range** specified.

Main Program

```
SET UserLen TO 0
SET UserBre TO 0

CALL GetValidValue (5, 50, UserLen)

CALL GetValidValue (0, 20, UserBre)
```

GetValidValue

```
PROCEDURE GetValidValue (low, high, userVal)
REPEAT
   RECEIVE userVal FROM KEYBOARD
   IF userVal < low OR userVal > high THEN
      SEND "Please Re-enter" TO DISPLAY
   END IF
UNTIL userVal >=low AND userVal <=high
END PROCEDURE
```

The implementation of a reusable **function** would look like this.

Main Program

```
SET UserLen TO 0
SET UserBre TO 0

UserLen = GetValidValue (5, 50)

UserBre = GetValidValue (0, 20)
```

GetValidValue

```
FUNCTION GetValidLength (low, high) RETURNS INTEGER
REPEAT
   RECEIVE userVal FROM KEYBOARD
   IF userVal < low OR userVal > high THEN
      SEND "Please Re-enter" TO DISPLAY
   END IF
UNTIL userVal >= low AND userVal <= high
RETURN userVal
END FUNCTION
```

Functions v Procedures

Functions	Procedures
Parameters only used for input values	Parameters used for input and output values
Returned value is stored in a variable	Formal parameters update the actual parameters to return a value.
Only one output allowed	Multiple outputs allowed
Multiple input parameters allowed	Multiple input parameters allowed

Practise Task

4. A modular program is required to input the names of 10 cities together with their average summer temperature and their average winter temperature. All details should be stored in a record structure.

All average temperatures should be whole numbers between -20 and 50 degrees Celsius.

The program should find the details of the cities with the highest summer temperature and those with the lowest winter temperature. Full details for these cities should be written to a csv file.

Read the information below before starting this task

Parameter passing with arrays of records

It is important to note that when arrays of records are passed as parameters, the subprogram must include

- brackets to indicate it is an array
- AS to indicate the data type (all other arguments for this subprogram must have data types explicitly stated as well)

```
Private Sub btnStart_Click(sender As Object, e As EventArgs) Handles
btnStart.Click

   Dim users(10) As recordDetails
   Dim counter As Integer
   counter = 0

   Call getDetails(users, counter)

End Sub
```

Actual Parameters listed here as normal

```
Private Sub getDetails(ByRef users() As recordDetails, ByRef counter As
Integer)

   users(counter).firstname = InputBox("Enter first name")
   users(counter).surname = InputBox("Enter surname")
   users(counter).age = InputBox("Enter age")

End Sub
```

Formal Parameters must have empty brackets for the array and all data types declared

Parameter Passing

What is parameter passing?

Parameters are

- the **variables** or **values** that are passed in or out of procedures.
- the **variables** or **values** that are passed into functions

Parameter passing allows variables to be **used and updated** by sub-programs.

The program below uses three procedures.

Main Program

```
SET UserLen TO 0
SET UserBre TO 0
SET TotalArea TO 0

CALL GetDimensions (UserLen, UserBre)
CALL CalculateArea (UserLen, UserBre, Area)
CALL DisplayArea (TotalArea)
```

Procedures must be **defined** before they are **called**.

Main Program

```
...
CALL GetDimensions (UserLen, UserBre)
...
```

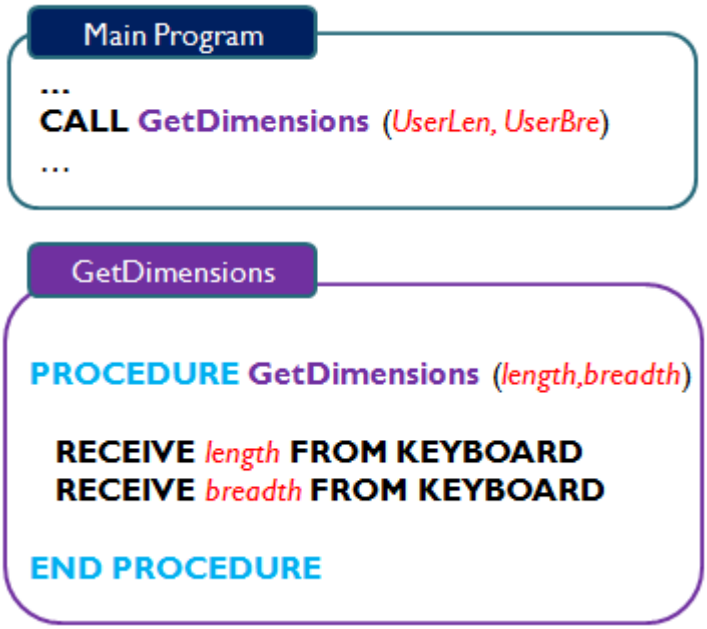
This line **CALLS** the procedure

GetDimensions

```
PROCEDURE GetDimensions (length,breadth)

  RECEIVE length FROM KEYBOARD
  RECEIVE breadth FROM KEYBOARD

END PROCEDURE
```

This section **DEFINES** the procedure

The **order** in which parameters are listed is important.

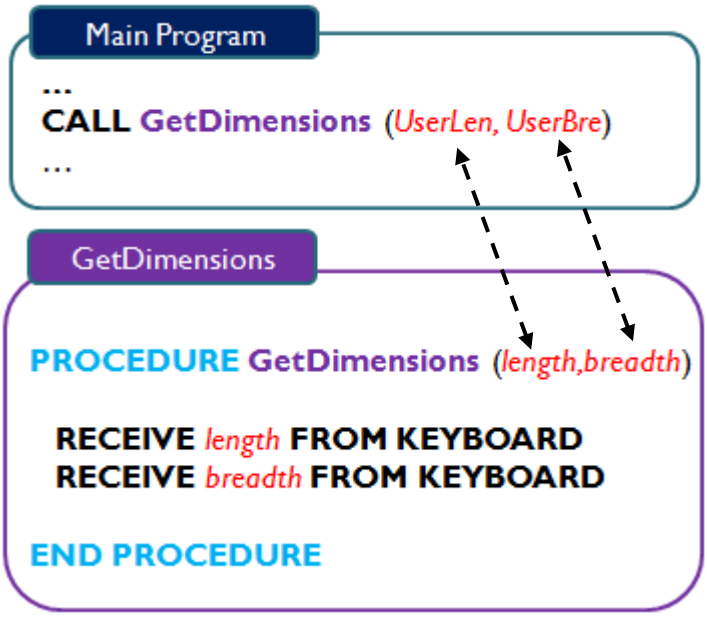


Notice the **order** of the parameters here…

…must be the same as the order here – but the **names can be different**

Actual and Formal Parameters

Parameters can be **actual** or **formal**.



These are known as **actual parameters**

These are known as **formal parameters**

Actual parameters contain the value which is to be passed to the sub-program's formal parameter.

Formal parameters are used by the sub-program and contain a copy of or link to the values passed from the actual parameters.

Main Program

```
CALL GetDimensions (UserLen, UserBre)
CALL CalculateArea (UserLen, UserBre, Area)
CALL DisplayArea (TotalArea)
```

GetDimensions

```
PROCEDURE GetDimensions (length,breadth)

  RECEIVE length FROM KEYBOARD
  RECEIVE breadth FROM KEYBOARD

END PROCEDURE
```

CalculateArea

```
PROCEDURE CalculateArea (length, breadth,Area)

  SET Area TO length * breadth

END PROCEDURE
```

DisplayArea

```
PROCEDURE DisplayArea (Area)

  SEND Area TO DISPLAY

END PROCEDURE
```

Scope of Variables

The scope of a variable is the area of code in which the variable is usable

i.e. how much of the program has access to it.

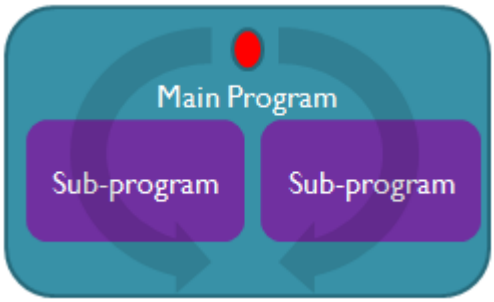
The scope of a variables can be either:

- **Global**
- **Local**

Global Variables

A global variable exists and can be accessed and changed from **any part of the program.**

Global variables do not have to be passed into procedures as parameters because the procedure can access it without doing so.



Global variables **reduce modularity** of a program and should be avoided wherever possible.

The use of global variables reduces modularity because:

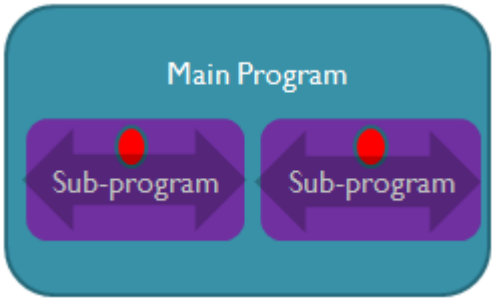
- Different programmers could use **conflicting variable names** which would cause errors.
- Any procedure could **accidentally alter** a global variable as it doesn't have to be passed in to be used.

Local Variables

Local variables exist only **within a procedure** or **function**. They are **declared within** a sub-program

They are **not passed in or out** and can only be used within the sub-program they were declared in.

Local variables **cannot be accessed** from out with their own sub-program which limits their scope.



It is always preferable to limit the scope of a variable to an individual sub-program wherever possible.

Limiting the scope of a variable is done by:

- Using **local variables** which can only be accessed with their own sub-program.
- Using **parameter passing** to only pass to a sub-program the variables it requires.

Practice Questions

Question 1 (SQP Qu 5)

Describe one problem that can occur when using global variables in a program. (1)

Question 2 (2019 Qu 18)

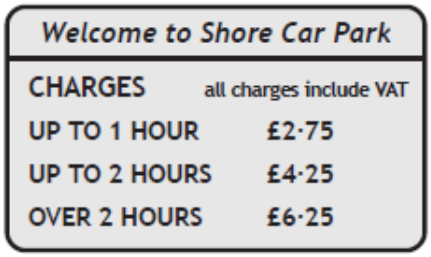
Part of the program code is shown below.

```
Line 1    DECLARE emails AS ARRAY OF STRING INITIALLY []
Line 2    DECLARE uniques AS ARRAY OF STRING INITIALLY []
...
Line 11   importEmails(emails)
Line 12   sortEmails(emails)
Line 13   removeDuplicates(emails, uniques)
Line 14   chooseWinner(uniques)
...
Line 70   PROCEDURE removeDuplicates(ARRAY OF STRING list,
          ARRAY of STRING newList)
Line 71      DECLARE position INITIALLY 0
Line 72      SET newList[position] TO list[0]
Line 73      FOR index FROM 1 to length(list)-1 DO
Line 74         IF list[index] ≠ newList[position] THEN
Line 75            SET position TO position + 1
Line 76            SET newList[position] TO list[index]
Line 77         END IF
Line 78      END FOR
Line 79   END PROCEDURE
```

a) Identify the formal parameter and identify the actual parameter. (2)

Question 3 (2017)

A program is used to calculate parking charges for a public car park.
The arrival and departure times are converted to and stored as real numbers, for example: 06:30 hours will be converted to and stored as 6.5.



The function below is used to calculate the cost of parking for each car.

```
Line 1 FUNCTION calcCost(REAL departure, REAL arrival) RETURNS REAL
Line 1     DECLARE hours_parked INITIALLY 0
Line 3     DECLARE parking_charge INITIALLY 0
Line 4     SET hours_parked TO departure – arrival
Line 5         IF hours_parked <= 1 THEN
Line 6             SET parking_charge TO 2.75
Line 7        ELSE
Line 8        IF hours_parked <=2 THEN
Line 9            SET parking_charge TO 4.25
Line 10     ELSE
Line 11           SET parking_charge TO 6.25
Line 12   END IF
Line 13   END IF
Line 14   RETURN parking_charge
Line 15 END FUNCTION
```

This function is called using the line below:
SET cost TO calcCost (arrived, left)

Identify a formal parameter used in the code above and explain what is meant by a formal parameter. (2)