Paper 2

Papers Dock

COMPUTER SCIENCE 9618 PAPER 2

String Concatenation

String concatenation is the process of combining two or more strings into a single string. In pseudocode we use & symbol to combine two strings

```
FirstName <-- "Taha"
LastName <-- "Ali"
FullName <-- FirstName & LastName

TahaAli
```

If you need space between them then you would have to concatenate space character aswell

```
FirstName <-- "Taha"
LastName <-- "Ali"
FullName <-- FirstName &  " " & LastName

Taha Ali
```

Selection Statement

Selection statements are programming constructs that allow the execution of certain blocks of code based on conditions.

IF .....THEN.....ELSE.....ENDIF
CASE.....OF.....OTHERWISE.....ENDCASE

Syntax Of IF Statement

IF statements may or may not have an ELSE clause.
IF statements without an else clause are written as follows:

```
IF <condition> THEN
    <statement(s)>
ENDIF
```

IF statements with an else clause are written as follows:

```
IF <condition> THEN
    <statement(s)>
ELSE
    <statement(s)>
ENDIF
```

```
IF Num > 2 THEN
    OUTPUT "Greater Than 2"
ELSE
    IF Num < 2 THEN
        OUTPUT "Less Than 2"
    ELSE
        OUTPUT "Equal To 2"
    ENDIF
ENDIF
```

Logical Operators

AND	OR
Both Condition Should Be True	Any one Condition Should Be True

Relational Operators

- `>` Greater than
- `<` Less than
- `>=` Greater than or equal to
- `<=` Less than or equal to
- `=` Equal to
- `<>` Not equal to

The result of these operations is always of data type BOOLEAN. In complex expressions it is advisable to use parentheses

Practice Question

Write a pseudocode of a program in which Input three numbers and print the Largest Number

```
OUTPUT "Enter the Number"
INPUT Num1
INPUT Num2
INPUT Num3

IF Num1 > Num2 AND Num1 > Num2 THEN
    OUTPUT Num1, " is largest"
ELSE
    IF Num2 > Num1 AND Num2 > Num3 THEN
        OUTPUT Num2, " is largest"
    ELSE
        OUTPUT Num3, " is largest"
    ENDIF
ENDIF
```

Syntax Of CASE Statement

CASE statements allow one out of several branches of code to be executed, depending on the value of a variable.

CASE statements are written as follows:

```
CASE OF <identifier>
    <value 1> : <statement1>
                <statement2>

    <value 2> : <statement1>
                <statement2>
ENDCASE
```

An OTHERWISE clause can be the last case:

```
CASE OF <identifier>
    <value 1> : <statement1>
                <statement2>

    <value 2> : <statement1>
                <statement2>

    OTHERWISE : <statement1>
                <statement2>
ENDCASE
```

Each value may be represented by a range, for example:

```
<value1> TO <value2> : <statement1>
                       <statement2>
```

```
Position <-- 0
INPUT Move
CASE OF Move
 'W' : Position ← Position - 10
 'S' : Position ← Position + 10
 'A' : Position ← Position - 1
 'D' : Position ← Position + 1
 OTHERWISE : OUTPUT "Error"
ENDCASE
```

Practice Question

Write a program that Outputs grades based on the following criteria:

- Score >= 90: "A"
- Score >= 80: "B"
- Score >= 70: "C"
- Else: "F"

Practice Question

Create a calculator that takes two numbers and an operator (+, -, *, /) and performs the operation using a CASE statement.

Built in Function

An error will be generated if a function call is not properly formed or if the parameters are of an incorrect type or an incorrect value.

String and Character Functions

- A string of length 1 may be either of type `CHAR` or `STRING`
- A `CHAR` may be assigned to, or concatenated with, a `STRING`
- A `STRING` of length greater than 1 cannot be assigned to a `CHAR`

`LEFT(ThisString : STRING, x : INTEGER) RETURNS STRING`

returns leftmost `x` characters from `ThisString`
Example: `LEFT("ABCDEFGH", 3)` returns `"ABC"`

`RIGHT(ThisString : STRING, x : INTEGER) RETURNS STRING`

returns rightmost `x` characters from `ThisString`
Example: `RIGHT("ABCDEFGH", 3)` returns `"FGH"`

`MID(ThisString : STRING, x : INTEGER, y : INTEGER) RETURNS STRING`

returns a string of length `y` starting at position `x` from `ThisString`
Example: `MID("ABCDEFGH", 2, 3)` returns string `"BCD"`

`LENGTH(ThisString : STRING) RETURNS INTEGER`

returns the integer value representing the length of `ThisString`
Example: `LENGTH("Happy Days")` returns `10`

`TO_UPPER(x : <datatype>) RETURNS <datatype>`
`<datatype>` may be `CHAR` or `STRING`

returns an object of type `<datatype>` formed by converting all characters of `x` to upper case.

Examples:
- `TO_UPPER("Error 803")` returns `"ERROR 803"`
- `TO_UPPER('a')` returns `'A'`

`TO_LOWER(x : <datatype>) RETURNS <datatype>`
`<datatype>` may be `CHAR` or `STRING`

returns an object of type `<datatype>` formed by converting all characters of `x` to lower case.

Examples:
- `TO_LOWER("JIM 803")` returns `"jim 803"`
- `TO_LOWER('W')` returns `'w'`

`NUM_TO_STR(x : <datatype1>) RETURNS <datatype2>`

returns a string representation of a numeric value.
`<datatype1>` may be `REAL` or `INTEGER`, `<datatype2>` may be `CHAR` or `STRING`
Example: `NUM_TO_STR(87.5)` returns `"87.5"`

`STR_TO_NUM(x : <datatype1>) RETURNS <datatype2>`

returns a numeric representation of a string.
`<datatype1>` may be `CHAR` or `STRING`, `<datatype2>` may be `REAL` or `INTEGER`
Example: `STR_TO_NUM("23.45")` returns `23.45`

`IS_NUM(ThisString : <datatype>) RETURNS BOOLEAN`

returns `TRUE` if `ThisString` represents a valid numeric value.
`<datatype>` may be `CHAR` or `STRING`
Example: `IS_NUM("-12.36")` returns `TRUE`

```
ASC(ThisChar : CHAR) RETURNS INTEGER
```

returns an integer value (the ASCII value) of character `ThisChar`
Example: `ASC('A')` returns `65`, `ASC('B')` returns `66`, etc.

```
CHR(x : INTEGER) RETURNS CHAR
```

returns the character whose integer value (the ASCII value) is `x`
Example: `CHR(65)` returns `'A'`, `CHR(66)` returns `'B'`, etc.

Numeric Functions

```
INT(x : REAL) RETURNS INTEGER
```

returns the integer part of `x`
Example: `INT(27.5415)` returns `27`

```
RAND(x : INTEGER) RETURNS REAL
```

returns a real number in the range `0` to `x` (**not** inclusive of `x`).
Example: `RAND(87)` may return `35.43`

(b) Program variables have values as follows:

Variable	Value
Title	"101 tricks with spaghetti"
Version	'C'
Author	"Eric Peapod"
PackSize	4
WeightEach	6.2
Paperback	TRUE

(i) Evaluate each expression in the following table.
If an expression is invalid, write ERROR.

For the built-in functions list, refer to the **Appendix** on page 16.

Expression	Evaluates to
MID(Title, 5, 3) & RIGHT(Author, 3)	
INT(WeightEach * PackSize)	
PackSize >= 4 AND WeightEach < 6.2	
LEFT(Author, ASC(Version) - 65)	
RIGHT(Title, (LENGTH(Author) - 6))	

ISNUM() And String Comparison

IS_NUM(ThisString : <datatype>) RETURNS BOOLEAN

returns TRUE if ThisString represents a valid numeric value.
<datatype> may be CHAR or STRING
Example: IS_NUM("-12.36") returns TRUE

ISNUM() is basically a built in function that takes string value as a parameter and return true if that string value are Numbers

In pseudocode you can compare the strings for eg

“Z” > “A”

Character >= “A” and Character <= “Z”

Practice Question

There are Certain Conditions for a password to be correct

1) First Digit Should Be Capital
2) 2nd, 3rd, 4th Digit Should be Numbers
3)Length Of the Password Should Be 9

Write a pseudocode of a program in which take input the password and print "Approve” If these Conditions are met and print "Re write Password" If not

Note : First think how you are going to solve this Problem. Imagine and write Correct Value and Expected output this will help you in thinking about the solution

Solution

```
DECLARE Pass, First, Digit : STRING
DECLARE LenPass : INTEGER
DECLARE DigitFlag : BOOLEAN

OUTPUT "Enter the password"
INPUT Pass

First <-- LEFT(Pass, 1)
Digit <-- MID(Pass, 2, 3)
LenPass <-- LENGTH(Pass)
DigitFlag <-- IS_NUM(Digit)

IF First >= "A" AND First <= "Z" AND DigitFlag = TRUE AND
LenPass = 9 THEN
    OUTPUT "Approve"
ELSE
    OUTPUT "Re Write Password"
ENDIF
```

Loops

Count-controlled (FOR) loops	Post-condition (REPEAT) loops	Pre-condition (WHILE) loops

FOR LOOP

```
FOR <identifier> ← <value1> TO <value2>
    <statement(s)>
NEXT <identifier>
```

```
DECLARE Count : INTEGER

FOR Count <-- 1 TO 10
    OUTPUT "Papersdock"
NEXT Count
```

Practice Question

Input One Name and then Output “Good Morning” 23 Times concatenated with the Name

```
DECLARE x : INTEGER
DECLARE Name : STRING

OUTPUT "Enter the Name"
INPUT Name
FOR x <-- 1 TO 23
    OUTPUT Name  & " " & "Good Morning"
NEXT x
```

- Repetition Known
- The identifier must be an INTEGER variable.
- No condition
- Fixed Loops and will always execute
- Values must evaluate to integers.
- The loop runs from value1 to value2 inclusive.
- Statements inside the loop execute for each assigned value.
- If value1 = value2, the loop runs once.
- If value1 > value2, the loop does not execute.

```
FOR <identifier> ← <value1> TO <value2> STEP <increment>
    <statement(s)>
NEXT <identifier>
```

```
DECLARE X : INTEGER

FOR x <-- 20 TO 1 STEP -1
    OUTPUT x
NEXT x
```

- The increment must evaluate to an integer.
- The identifier starts at value1 and increases by the increment.
- The loop runs until the identifier reaches or exceeds value2.
- If the identifier goes past value2, the loop ends.
- The increment can be negative.

WHILE LOOP Pre Condition

```
WHILE <condition>
        <statement(s)>
ENDWHILE
```

- The condition must evaluate to a Boolean.
- The condition is tested before executing the statements.
- Statements run only if the condition is TRUE.
- After each execution, the condition is tested again.
- The loop stops when the condition is FALSE.
- If the condition is FALSE initially, the statements are not executed

```
DECLARE Count : INTEGER

Count <-- 0
WHILE Count < 10
    OUTPUT "Papersdock"
    Count <-- Count + 1
ENDWHILE
```

REPEAT LOOP Post Condition

```
REPEAT
        <statement(s)>
UNTIL <condition>
```

- The condition must evaluate to a Boolean.
- The statements are executed at least once.
- The condition is tested after the statements run.
- If the condition is TRUE, the loop terminates.
- If the condition is FALSE, the statements run again.

```
DECLARE Count : INTEGER

Count <-- 0
REPEAT
    OUTPUT "Papersdock"
    Count <-- Count + 1
UNTIL Count = 10
```

Practice Question

Input 500 Names and Print in the format "Hello Name" and Prompt the user aswell

PROGRAMMING TECHNIQUES

1) SUM TECHNIQUE

SUM <-- 0 (Outside the loop)

SUM <-- Sum + Number (Inside the loop)

2) Counting TECHNIQUE

Count <-- 0 (Outside the loop)

Count <-- Count + 1 (Inside the loop)

QUESTION

Input 10 Numbers and Print the sum of those 10 Numbers

```
DECLARE Count, Sum, Num : INTEGER

Sum <-- 0
FOR Count <-- 1 TO 10
    OUTPUT "Enter the number"
    INPUT Num
    Sum <-- Sum + Num
NEXT Count

OUTPUT Sum
```

QUESTION

Input 500 Numbers and print how many numbers are Positive

```
DECLARE Count, PosCount, Num : INTEGER

PosCount <-- 0
FOR Count <-- 1 TO 500
    OUTPUT "Enter the number"
    INPUT Num
    IF Num > 0 THEN
        PosCount <-- PosCount + 1
    ENDIF
NEXT Count
OUTPUT PosCount
```

PRACTICE QUESTION

Input 499 and print how many number are positive , negative and zero with a suitable message

```
DECLARE Count, PosCount, NegCount, ZeroCount Num : INTEGER

PosCount <-- 0
NegCount <-- 0
ZeroCount <-- 0
FOR Count <-- 1 TO 499
    OUTPUT "Enter the number"
    INPUT Num
    IF Num > 0 THEN
        PosCount <-- PosCount + 1
    ELSE
        IF Num < 0 THEN
            NegCount <-- NegCount + 1
        ELSE
            ZeroCount <-- ZeroCount + 1
        ENDIF
    ENDIF
NEXT Count
OUTPUT NegCount, "Negative"
OUTPUT PosCount, "Positive"
```

Flag Looping

Whenever we need to stop the loop or to make the code efficient we use a flag and conditional loops

```
Flag <-- TRUE
WHILE Flag = True
        INPUT Response
        IF Response = "No" THEN
                Flag <-- FALSE
        ENDIF
ENDWHILE
```

QUESTION

Condition for a password to be correct is that first digit should be capital

Write Pseudocode in which take input again and again until the Password is correct and print approve if it is correct and if it is wrong print re write password and take input again

```
DECLARE CorrectFlag : BOOLEAN
DECLARE Pass : STRING
DECLARE FirstChar : CHAR

CorrectFlag <-- FALSE
WHILE CorrectFlag = FALSE
    OUTPUT "Enter your password"
    INPUT Pass

    FirstChar <-- LEFT(Pass, 1)
    IF FirstChar >= 'A' AND FirstChar <= 'Z' THEN
        OUTPUT "Approve"
        CorrectFlag <-- TRUE
    ELSE
        OUTPUT "Re-Write Your Password"
    ENDIF
ENDWHILE
```

PRACTICE QUESTION

Take Numbers as input again and again until the user types in 0 and find the average of those Numbers

```
DECLARE Flag : BOOLEAN
DECLARE Num, Sum, Count : INTEGER
DECLARE Avg : REAL
Flag <-- TRUE
Sum <-- 0
Count <-- 0

WHILE Flag = TRUE
    OUTPUT "Enter the number"
    INPUT Num

    IF  Num = 0 THEN
        Flag <-- FALSE
    ELSE
        Count <-- Count + 1
        Sum <-- Sum + Num
    ENDIF
ENDWHILE

Avg <-- Sum/Count
OUTPUT Avg
```

DIV AND MOD

19 DIV 3
19 MOD 3

27 DIV 4
27 MOD 4

`MOD`	finds the remainder when one number is divided by another. Example: `10 MOD 3` evaluates to `1`
`DIV`	finds the quotient when one number is divided by another. Example `10 DIV 3` evaluates to `3`

PRACTICE QUESTION

Take numbers as input repeatedly until the user types in -1. Count how many of those numbers are even and display the count when the loop ends.

Hint : Use Built-in Function

Solution For Even Numbers

```
DECLARE Flag : BOOLEAN
DECLARE Number, temp, EvenNumber : INTEGER
Flag <-- TRUE

EvenNumber <-- 0
WHILE Flag = TRUE
    OUTPUT "Enter the number"
    INPUT Number

    IF Number = -1 THEN
        Flag <-- FALSE
    ELSE
        temp <-- Number MOD 2
        IF temp = 0 THEN
            EvenNumber <-- EvenNumber + 1
        ENDIF
    ENDIF
ENDWHILE
OUTPUT "Even Numbers Count: ", EvenNumber
```

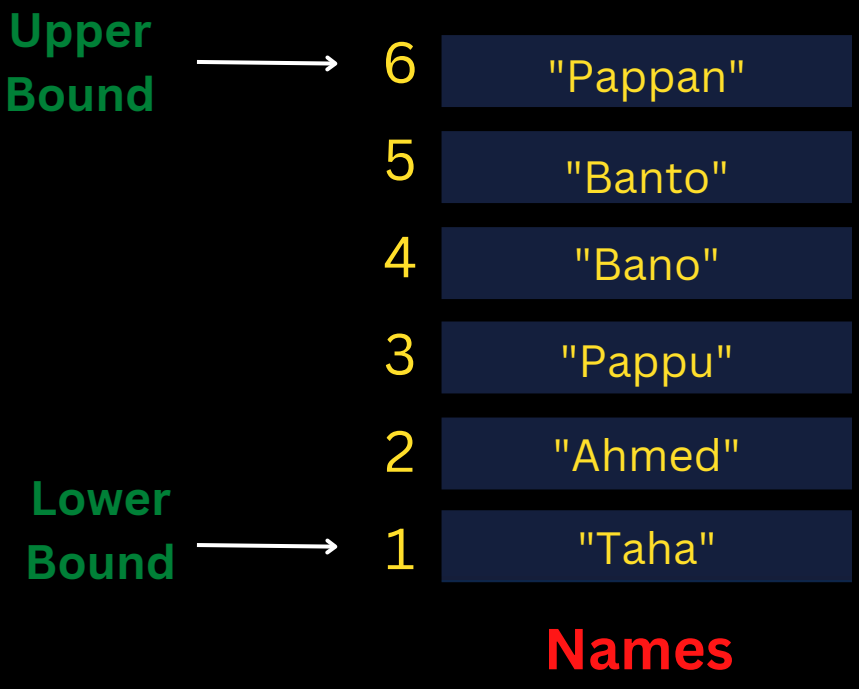
Solution For Odd Numbers

```
DECLARE Flag : BOOLEAN
DECLARE Number, temp, OddNumber : INTEGER
Flag <-- TRUE

OddNumber <-- 0
WHILE Flag = TRUE
    OUTPUT "Enter the number"
    INPUT Number

    IF Number = -1 THEN
        Flag <-- FALSE
    ELSE
        temp <-- Number MOD 2
        IF temp <> 0 THEN
            OddNumber <-- OddNumber + 1
        ENDIF
    ENDIF
ENDWHILE
OUTPUT "Odd Numbers Count: ", OddNumber
```

Array



What Is Array ?

Arrays stores multiple values of same datatype and the data is stored on position known as index

How to access Individual Element in an array

Names[3] "Bano"
Names[0] "Taha"
Names[4]
Names[5]

How to Declare the Array

```
DECLARE  NameOfArray : ARRAY [ LowerBound  :  UpperBound ] OF DataType
```

Practice Question

Create an array with the name "FoodItems" with the Lower Bound (1) and Upper Bound (6) Of DataType String

```
DECLARE  FoodItem : ARRAY [ 1  :  6 ] OF STRING
```

Question

Construct an array of 10 Elements and store Names in it which User Will Input.

```
DECLARE Names : ARRAY [ 1 : 10 ] OF STRING
DECLARE Temp : STRING

FOR Index <-- 1 TO 10
    INPUT Temp
    Names[Index] <-- Temp
NEXT Index
```

```
DECLARE Names : ARRAY [ 1 : 10 ] OF STRING
DECLARE Temp : STRING

FOR Index <-- 1 TO 10
    INPUT Names[Index]
NEXT Index
```

Practice Question

construct an array of 100 Elements and store "No Data" in all the elements and the name of Array is " ResultArray"

```
DECLARE ResultArray : ARRAY [ 1 : 100 ] OF STRING


FOR Index <-- 1 TO 100
    ResultArray[Index] <-- "No Data"
NEXT Index
```

Linear Searching

In this each element of an array is compared with the value to be found in order from lower bound to upper bound until the item is found or upper bound is reached

Note: For Searching purpose you need to use Selection Statement

Question

There is an Array which contains names of 500 students. Search at which index position "Bano" is stored

Name Of Array : STName

```
DECLARE Index : INTEGER

FOR Index <-- 1 TO 500
    IF STName[Index] = "Bano" THEN
        OUTPUT Index
    ENDIF
NEXT Index
```

Practice Question

Already there is an Array with the name "SearchBox" with 500 Elements of DataType String.

Search through the array and find how many times "Empty" was Repeated

```
DECLARE Index, EmptyCount : INTEGER

EmptyCount <-- 0
FOR Index <-- 1 TO 500
    IF STName[Index] = "Empty" THEN
        EmptyCount <-- EmptyCount + 1
    ENDIF
NEXT Index

OUTPUT "Empty: ", EmptyCount
```

Practice Question

There is an Array which contains names of 500 students. Output found if “Bano” is in the array and Not Found if not in the array

Name Of Array : STName

```
DECLARE Index, EmptyCount : INTEGER
DECLARE Flag : BOOLEAN

Flag <-- FALSE
FOR Index <-- 1 TO 500
    IF STName[Index] = "Bano" THEN
        Flag <-- TRUE
    ENDIF
NEXT Index

IF Flag = FALSE THEN
    OUTPUT "Not Found"
ELSE
    OUTPUT "Found"
```

Practice Question

There is an Array which contains 10 numbers. Write an algorithm which will find the values which are less than 3 and will make them 5

Name Of Array : NumArray

Practice Question

There is an Array which contains 6 numbers. Reverse the array so for example the number at position 1 should be at position 6

Name Of Array : NumArray

```
DECLARE Temparray : ARRAY [ 1 : 6 ] OF INTEGER
DECLARE Index : INTEGER

FOR Index <-- 1 TO 6
    Temparray[7 - Index] <-- NumArray[Index]
NEXT Index
```

2 Dimensional Array

A 2D array is a data structure that stores elements in a grid or matrix of rows and columns. It consists of multiple rows and columns, where each element is uniquely identified by its row and column values. 2D arrays are commonly used in programming to represent images, screens and game boards

Create an Array with 5 rows and 4 Columns

Declaration Of 2D Array

ROWS COLUMN

DECLARE NameOfArray : ARRAY [L.B : U.B, L.B : U.B] OF DataType

DECLARE Numbers : ARRAY [1 : 5 , 1 : 4] OF INTEGERS

Practice Question

Declare an array of datatype String (Names) of 545 rows and 250 columns

```
DECLARE  Names : ARRAY [ 1 : 545 ,  1 : 250 ] OF INTEGERS
```

Concept Of Nested Loops

```
DECLARE Count, Index : INTEGER

FOR Count <-- 1 TO 10
    FOR Index <-- 1 TO 5
            OUTPUT "PAPERSDOCK"
            OUTPUT "Count: ",Count, " Index: ", Index
    NEXT Index
NEXT Count
```

Accessing The Individual Element

Number[1,1] 21
Number[3,3] 5
Number[5,2] 78

	1	2	3	4
1	21			
2				
3			5	
4				
5		78		

Question

Construct an Array Of DataType String and 500 Rows and 45 columns and initialize all the Elements

(Name Of The Array = EmptyBox)

```
DECLARE EmptyBox : ARRAY [ 1 : 500, 1 : 45 ] OF STRING
DECLARE Row, Col : INTEGER

FOR Row <-- 1 TO 500
    FOR Col <-- 1 TO 45
        EmptyBox[Row, Col] <-- ""
    NEXT Col
NEXT Row
```

Question

There is already an array (SearchBox) with 500 rows and 30 Columns. Search for the word "Empty" and Print How many times it was repeated

```
DECLARE Row, Col, Count : INTEGER

Count <-- 0
FOR Row <-- 1 TO 500
    FOR Col <-- 1 TO 30
        IF SearchBox[Row, Col] = "Empty" THEN
            Count <-- Count + 1
        ENDIF
    NEXT Col
NEXT Row
```

Question

There is already an array SearchBox with 500 rows and 356 columns. Search for the word "Empty" and Replace it with "Bano"

```
DECLARE Row, Col: INTEGER


FOR Row <-- 1 TO 500
    FOR Col <-- 1 TO 30
        IF SearchBox[Row, Col] = "Empty" THEN
            SearchBox[Row, Col] <-- "Bano"
        ENDIF
    NEXT Col
NEXT Row
```

Files

The purpose of file is to store data permenantly

Three Modes In File

READ MODE

Read mode is used when you only want to read data from existing file

WRITE MODE

Write mode is used when you want to write data to a new file

APPEND MODE

Append mode is used when you want to write data to an existing file

Write Mode

Writing to a text file means Creating a text file. All the previous data would be deleted and a file will be written from scratch

Pseudocode For Writing

```
OPENFILE  "FileName.txt" FOR WRITE
WRITEFILE "FileName.txt" , "String"/Variable
CLOSEFILE "FileName.txt"
```

Question

Input 5 names and store them in a new file “Names.txt”

```
DECLARE Index : INTEGER
DECLARE Name : STRING

OPENFILE "Names.txt" FOR WRITE

FOR Index <-- 1 TO 5
    OUTPUT "Enter Names"
    INPUT Name
    WRITEFILE "Names.txt", Name
NEXT Index

CLOSEFILE "Names.txt"
```

Practice Question

Store all the names which are present in an Array with the name "School" and contains 700 elements into a new file "SchoolNames.txt'

```
DECLARE Index : INTEGER
DECLARE Name : STRING

OPENFILE "SchoolNames.txt" FOR WRITE

FOR Index <-- 1 TO 700
    Name <-- School[Index]
    WRITEFILE "SchoolNames.txt", Name
NEXT Index

CLOSEFILE "SchoolNames.txt"
```

Practice Question

There are 54 employee and they will input their ID numbers with the names. ID will be the first 4 characters. Store only the ID number in the text file "New.txt"

E.g : "2675Bano"

```
DECLARE Value, ID : STRING

OPENFILE "New.txt" FOR WRITE

FOR Index <-- 1 TO 54
    OUTPUT "Enter the ID with Name"
    INPUT Value
        ID <-- LEFT(Value, 4)
        WRITEFILE  "New.txt", ID
NEXT Index

CLOSEFILE "New.txt"
```

Append Mode

Sometimes we want to add data to an existing file rather than creating a new file.
Append Mode adds the new data to an existing file

Pseudocode For Append

```
OPENFILE  "FileName.txt" FOR APPEND
WRITEFILE "FileName.txt" , "String"/Variable
CLOSEFILE "FileName.txt"
```

Question

There is already a File with the name "New.txt"
add one more ID into the file.
The ID "2356"

```
OPENFILE  "New.txt" FOR APPEND
WRITEFILE "New.txt" , "2356"
CLOSEFILE "New.txt"
```

Read Mode

Whenever you want to search something from an existing file we use Read Mode

Pseudocode For Read

OPENFILE "FileName.txt" FOR READ
READFILE "FileName.txt" , Variable
CLOSEFILE "FileName.txt"

READS A LINE AND STORED ALL THE DATA IN VARIABLE

Question

There is an existing file known as sample.txt, Print all of its content and the file contains 500 line

```
DECLARE Index : INTEGER
DECLARE FileData : STRING

OPENFILE "Sample.txt"  FOR READ
FOR Index <-- 1 TO 500
    READFILE " Sample.txt", Filedata
    OUTPUT Filedata
NEXT Index
```

Practice Question

There is a file “Student.txt” and contains 300 lines. Search the whole file and output found if “Bano” is in the file and Not Found if not in the file

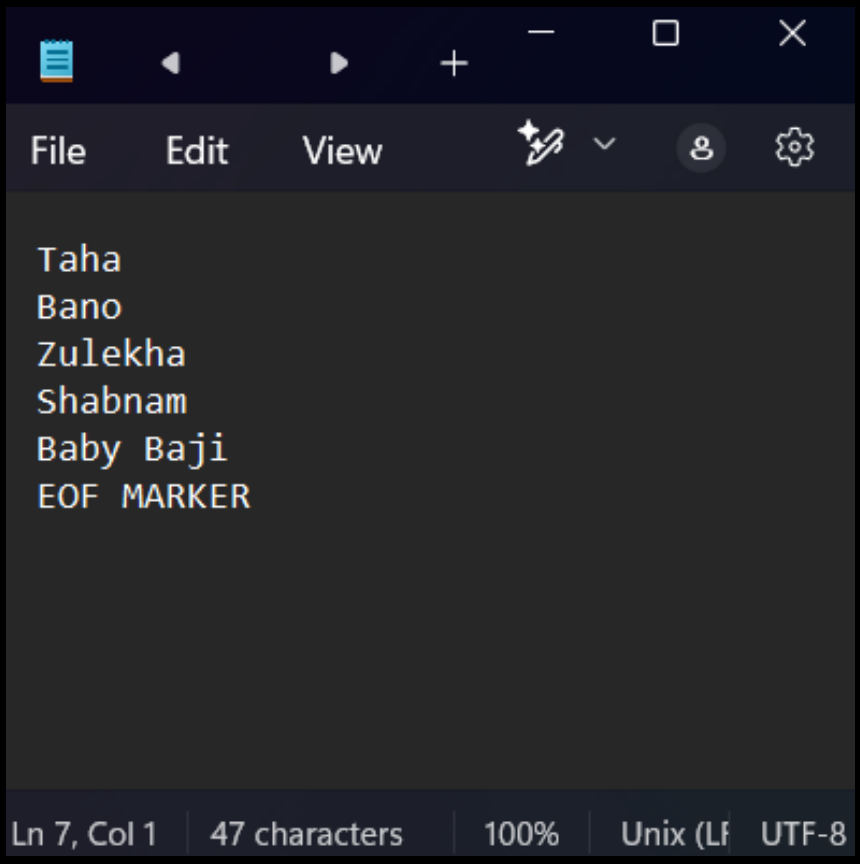
```
DECLARE line : INTEGER
DECLARE FileData : STRING
DECLARE Flag : BOOLEAN

Flag <-- TRUE
OPENFILE "Student.txt" FOR READ
FOR line <-- 1 TO 300
    READFILE "Student.txt", FileData
    IF FileData = "Bano" THEN
        Flag <-- FALSE
    ENDIF
NEXT line

IF Flag = TRUE THEN
    OUTPUT "Not Found"
ELSE
    OUTPUT "Found"
ENDIF
```

Concept Of EOF (End Of File)

If you want to read a file from begining to end and you don't know how much lines a file contains, we can use a conditional loop in this scenario



Text files contains a special market at the end of the file that we can use known as End Of file Marker

EOF (“text file”) is a built-in function that returns TRUE if the End Of File Marker Is Reached

Question

Read a file "Names.txt" and print all of its Content

```
DECLARE FileData : STRING

OPENFILE "Names.txt" FOR READ
WHILE NOT EOF("Names.txt")
    READFILE "Names.txt", FileData
    OUTPUT FileData
ENDWHILE
CLOSEFILE "Names.txt"
```

Practice Question

There is a music file and it contains data about CD on each line

FORMAT : Title ArtistName Location

40 Char 40 Char 8 Char

User will input location and you have to search that location and then print the title and artist name and how many CDs are related to that location

```
Count ← 0
OPENFILE "Music.txt" FOR READ
OUTPUT "Enter The Location"
INPUT CDLocation

WHILE NOT EOF ("Music.txt")
    READFILE "Music.txt", Filedata
    FileLocation ← RIGHT (Filedata, 8)
    IF CDLocation = FileLocation THEN
        CDTitle ← LEFT (Filedata, 40)
        CDartist ← MID (Filedata, 41, 40)
        Count ← count + 1
        OUTPUT CDTitle & " " & CDartist
    ENDIF
ENDWHILE
OUTPUT Count
CLOSEFILE "Music.txt"
```

Functions And Procedures

Functions and procedures are essential concepts in programming. They are both subroutines. Blocks of reusable code that perform specific tasks

What Are Functions ?

Function Returns A Value
Procedure Does Not Return A Value

Difference Between Output And Return

Imagine you are making Lemo Pani . If you pour it directly into glasses and serve it to everyone, that’s like Output, it is displayed immediately, but once it's gone, you can't reuse it.

However, if you store the Lemo Pani in a jug, that’s like Return, you’re keeping it for later use. It’s not immediately visible, but you can pour it into different glasses whenever needed.

In programming, Output simply shows the result, while Return saves it so it can be used again later.

Header Of Function

FUNCTION Name (Parameter : Datatype, Parameter : Datatype) RETURNS Datatype

ENDFUNCTION

The keyword RETURN is used as one of the statements within the body of the function to specify the value to be returned. Normally, this will be the last statement in the function definition, however, if the RETURN statement is in the body of the function its execution is immediate and any subsequent lines of code are omitted.

Header Of Procedure

PROCEDURE Name (Parameter : Datatype, Parameter : Datatype)

ENDPROCEDURE

CALL is used to run a procedure

Question

Write a program code for module named square that takes an integer value as a parameter and returns the square value of the number

```
FUNCTION Square (X : INTEGER) RETURNS INTEGER
    DECLARE temp : INTEGER

    temp <-- X * X
    RETURN temp
ENDFUNCTION
```

Practice Question

Write a program code for module named Greeting that takes a name as a parameter and outputs Good Morning concatenated with the name 10 times.

6 A company hires out rowing boats on a lake. The company has ten boats, numbered from 1 to 10.

The company is developing a program to help manage and record the hiring out process.

Hire information is stored in three global 1D arrays when a boat is hired out. Each array contains 10 elements representing each of the ten boats.

The three 1D arrays are summarised as follows:

Array	Data type	Description	Example data value
`HireTime`	`STRING`	The time the boat was hired out	`"10:15"`
`Duration`	`INTEGER`	The number of minutes of the hire	`30`
`Cost`	`REAL`	The cost of the hire	`5.75`

If an individual boat is not currently on hire, the corresponding element of the `HireTime` array will be set to `"Available"`.

The programmer has started to define program modules as follows:

Module	Description

`AddTime()`	• Called with two parameters:
	  ○ a `STRING` value representing a time
	  ○ an `INTEGER` value representing a duration in minutes
	• Adds the duration to the time to give a new time
	• Returns the new time as a `STRING`

`ListAvailable()`	• Called with a `STRING` value representing the time the hire will start
	• Outputs the boat numbers that will be available for hire at the given start time. A boat will be available for hire if it is either:
	  ○ currently not on hire, or
	  ○ due back before the given hire start time
	• Outputs the number of each boat available
	• Outputs the total number of boats available or a suitable message if there are none

`RecordHire()`	• Called with four parameters:
	  ○ an `INTEGER` value representing the boat number
	  ○ a `STRING` value representing the hire start time
	  ○ an `INTEGER` value representing the hire duration in minutes
	  ○ a `REAL` value representing the cost of hire
	• Updates the appropriate element in each array
	• Adds the cost of hire to the global variable `DailyTakings`
	• Converts the four input parameters to strings, concatenated using commas as separators, and writes the resulting string to the end of the existing text file `HireLog.txt`

(a) Write **pseudocode** for the module `ListAvailable()`.

```
PROCEDURE ListAvailable ( StartTime : STRING)
    DECLARE BoatCount , Index , DurationTimeNew : INTEGER
    DECLARE HireTimeNew , DueBackTime : STRING
    DECLARE Available : BOOLEAN.
    BoatCount ← 0
    FOR Index ← 1 TO 10
        Available ← FALSE
        IF HireTime [Index] = "Available" THEN
            Available ← TRUE
        ELSE
            HireTimeNew ← HireTime [Index]
            DurationTimeNew ← Duration [Index]
            DueBackTime ← AddTime (HireTimeNew , DurationTimeNew)
            IF StartTime > DueBackTime THEN
                Available ← TRUE
            ENDIF
        ENDIF
        IF Available = TRUE THEN
            BoatCount ← BoatCount + 1
            OUTPUT "Boat" , Index , "Is Available"
        ENDIF
    NEXT Index
    IF BoatCount = 0 THEN
        OUTPUT "No boats Available"
    ELSE
        OUTPUT "There are" , BoatCount , "Boats Available"
    ENDIF
ENDPROCEDURE.
```