ALevel CS C13 Data types and structures

Primitive data types(atomic data types): defined simply by commands built into the programming language.
Integer - whole number
Real - a number with a decimal point
TRUE or FALSE - Conditions
BOOLEAN - logical values
Char - single character

Futher data types:
String - store serveral characters
Data - A data consisting of day, month and year, sometimes including a time in hours, minutes and seconds

Recode type: know as a user-define type.

```
// declare a record type
TYPE <TypeIdentifier>
   DECLARE <field identifier> : <data type>
   .
   .
ENDTYPE
// declare a variable of this recode type
DECLARE <variable identifier> : <record type>
// access an individual filed using the dot notation
<variable identifier>.<field identifier>
```

Examples:

```
// declare a Persion record type
TYPE PersonType
   Name : STRING
   DateOfBirth : DATE
   Height : REAL
   NumberOfSiblings : INTEGER
   IsFullTimeStudent : BOOLEAN
ENDTYPE
// declare a variable of this type
DECLARE Person : PersonType
// assign a value to a file of this Person record
Person.Name ← "Fred"
Person.NumberOfSiblings ← 3
Person.IsFullTimeStudent ← TRUE
// output a field
OUTPUT Person.Name
```

Two-dimensional array:

```
// declaration
DECLARE <identifier> : ARRAY[<lBound1>:<uBound1>, <lBound2>:
<uBound2>] OF <dataType>
# example
Board : ARRAY[1:6,1:7] OF INTEGER

// Accessing 2D arrays
<arrayIdentifier>[x,y]
# example
Board[3,4] ← 0 // sets the element in row 3 and column 4 to zero

// set each element of array to zero
FOR Row ← 0 TO MaxRowIndex
   FOR Column ← 0 TO MaxColumnIndex
      ThisTable[Row, Column] ← 0
   NEXT Column
NEXT Row

// output

FOR Row ← 0 TO MaxRowIndex
   FOR Column ← 0 TO MaxColumnIndex
      OUTPUT ThisTable[Row, Column] // stay on same line
   NEXT Column
   OUTPUT Newline // move to next line for next row
NEXT Row
```

Text file

```
// writing a text file
OPENFILE <filename> FOR WRITE   //open the file for writing
WRITEFILE <filename>, <stringValue>    //write a line of text to the file
CLOSEFILE <filename> //close file

// reading from a text file
OPENFILE <filename> FOR READ // open file for reading
READFILE <filename>, <stringVariable> // read a line of text from the
file
CLOSEFILE <filename>  // close file
```

One-dimensional array:

```
// 1D array declaration
DECLARE <arrayIdentifier> : ARRAY[<lowerBound>:<upperBound>]
OF <dataType>
// pseudocode example
DECLARE List1 : ARRAY[1:3] OF STRING // 3 elements in this list
DECLARE List2 : ARRAY[0:5] OF INTEGER // 6 elements in this list
DECLARE List3 : ARRAY[1:100] OF INTEGER // 100 elements in this
list
DECLARE List4 : ARRAY[0:25] OF CHAR // 26 elements in this list
// accessing 1D array
<arrayIdentifier>[x]
// pseudocode example
NList[25] ← 0
AList[3] ← 'D'
```

iterator for loop:

```
FOR Index ← 0 TO 6
   INPUT MyList[Index]
NEXT Index
```

Linear search

```
MaxIndex ← 6
INPUT SearchValue
Found ← FALSE
Index ← –1
REPEAT
   Index ← Index + 1
   IF MyList[Index] = SearchValue
      THEN
         Found ← TRUE
   ENDIF
UNTIL FOUND = TRUE OR Index >= MaxIndex
IF Found = TRUE
   THEN
      OUTPUT "Value found at location: " Index
   ELSE
      OUTPUT "Value not found"
ENDIF
```

Bubble sort

```
n ← MaxIndex – 1
FOR i ← 0 TO MaxIndex – 1
   FOR j ← 0 TO n
      IF MyList[j] > MyList[j + 1]
         THEN
            Temp ← MyList[j]
            MyList[j] ← MyList[j + 1]
            MyList[j + 1] ← Temp
      ENDIF
   NEXT j
   n ← n – 1 // this means the next time round the inner loop, we don't
         // look at the values already in the correct positions.
NEXT i
```

improved bubble sort algorithm

```
n ← MaxIndex – 1
REPEAT
   NoMoreSwaps ← TRUE
   FOR j ← 0 TO n
      IF MyList[j] > MyList[j + 1]
      THEN
         Temp ← MyList[j]
         MyList[j] ← MyList[j + 1]
         MyList[j + 1] ← Temp
         NoMoreSwaps ← FALSE
      ENDIF
   NEXT j
   n ← n – 1
UNTIL NoMoreSwaps = TRUE
```

```
// Appending to a text file
OPENFILE <filename> FOR APPEND // open file for append
WRITEFILE <filename>, <stringValue> // write a line of text to the file
CLOSEFILE <filename> // close file

// The end-of-file(EOF) marker
OPENFILE "Test.txt" FOR READ
WHILE NOT EOF("Test.txt") DO
   READFILE "Test.txt", TextString
   OUTPUT TextString
ENDWHILE
CLOSEFILE "Test.txt"
```