Juan

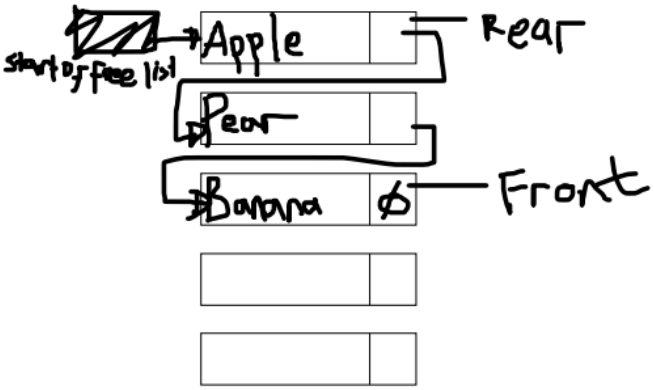
1.a.
INDEX = -1
Names[Middle] > s
First <- Middle + 1

b. Not sorted
c. i. Index
ii. -1

2.a.i

2 A queue Abstract Data Type (ADT) is to be implemented as a linked list of nodes. Each node is a record, consisting of a data field and a pointer field. The queue ADT also has a `FrontOfQueue` pointer and an `EndOfQueue` pointer associated with it. The possible queue operations are: `JoinQueue` and `LeaveQueue`.

a i Add labels to the diagram to show the state of the queue after three data items have been added to the queue in the given order: Apple, Pear, Banana.



[5]

ii Add labels to the diagram to show how the unused nodes are linked to form a list of free

B.i.
TYPE Node
    DECLARE Pointe r: INTEGER
    DECLARE Data : STRING
ENDTYPE

b.ii
DECLARE Queue : ARRAY[1:50] OF Node
DECLARE i : INTEGER

FOR i <- 1 to 49
    Queue[i].Pointer <- i + 1
NEXT i

Queue[50].Pointer <- 0
StartOfFreeList <- 1
FrontOfQueue <- 0
EndOf Queue <- 0

c The pseudocode algorithm for the queue operation `JoinQueue` is written as a procedure with the header:

- `PROCEDURE JoinQueue(NewItem)`

where `NewItem` is the new value to be added to the queue. The procedure uses the variables shown in the following identifier table:

Identifier	Data type	Description
`NullPointer`	`INTEGER`	Constant set to –1
Queue	Array OF Node	Array to store queue data
NewItem	`STRING`	Value to be added
StartOfFreeList	INTEGER	Pointer to next free node in array
FrontOfQueue	INTEGER	Pointer to first node in queue
EndOfQueue	INTEGER	Pointer to last node in queue
NextFreeNode	INTEGER	Pointer to node to be added

C.ii

NullPointer
FrontOfQueue <- NextFreeNode
EndOfQueue <- NextFreeNode
Queue[EndOfQueue] <- NextFreeNode
EndOfQueue <- NextFreeNode

3.a

```
DECLARE WordList : ARRAY [1:500] OF STRING

FOR i<- 1 TO 500
        WordList[i] <- ""
NEXT i
```

B.

```
PROCEDURE OutputList()
        FOR i <- 1 TO 500
                OUTPUT WordList[i]
        NEXT i
ENDPROCEDURE
```

C.

```
PROCEDURE LoadWords()
        DECLARE File : STRING
        DECLARE Search : INTEGER
        DECLARE Word : STRING

        OPENFILE FILENAME FOR READ
        FOR ….
                IF EOF(FileName) AND NOT FileExists(FileName) THEN
                        OUTPUT “Error”
                ELSE
                        Search <- 0
                        IF WordList[i] = “” AND Search =0 THEN
                                Search <- i
                        ENDIF
                NEXT i

        IF Index = 0 THEN
                OUTPUT “Full”
        ELSE
                WHILE NOT EOF (FILENAME)
                        READFILE FILENAME, Word
                        IF Index > 500 THEN
                                OUTPUT “Full”
                        ELSE
                                WordList[Index] <- Word
                                Index <- Index + 1
                        ENDIF
                ENDWHILE
        ENDIF

        CLOSEFILE FILENAME
ENDPROCEDURE
```

d.

```
PROCEDURE SortWord()
        DECLARE Temp : STRING
        DECLARE I : INTEGER
        DECLARE Pass: INTEGER

        FOR Pass <- 1 TO 499
                FOR I <- 1 TO 500-i
                        IF WordList[I]> WordList[I + 1] THEN
                                Temp <- WordList[I]
```

```
                        WordList[I] <- WordList {i + 1]
                        WordList[I+1] <- Temp
                    ENDIF
                NEXT I
        NEXT Pass
ENDPROCEDURE
```

Q1.

```
PROCEDURE FindStudent()
        DECLARE Index : INTEGER
        DECLARE Found : BOOLEAN

Found <- FALSE
Index <- 0              // Index <- 1

WHILE Index <= 10 AND Found = FALSE DO (10 = not less then ten)
        IF ….
        Index <- Index + 1 (9+1)
        IF SearchID= StudentID[Index] THEN (10)
                Found <- TRUE (10)
        ELSE
                Index <- Index + 1
        ENDIF
ENDWHILE

IF Found = TRUE THEN
        OUTPUT StudentName[Index]
ELSE
        OUTPUT “Student not found”
ENDIF
ENDPROCEDURE
```

Q2.
A.

```
PROCEDURE FindProduct()
DECLARE Index : INTEGER
DECLARE Found : BOOLEAN

INPUT SearchCode
Found <- FALSE
Index <- 0

        WHILE Index < 15 AND Found = FALSE DO
        Index <- Index + 1
                IF SearchCode = ProductCode[Index] THEN
                        Found <- TRUE
                ENDIF
        ENDWHILE

        IF Found = TRUE THEN
                OUTPUT “Found”
        ELSE
                OUTPUT “Not found”
        ENDIF

ENDPROCEDURE
```

B. (index can already count it, so output index)

C. 1, 8, 15, 15

```
125, 317, 204, 156, 421,109, 350, 278, 190, 463,312, 145, 399,
231, 176
```

125 278 176 500

Q3.

```
PROCEDURE BubbleSort()
        DECLARE Pass : INTEGER
        DECLARE Index : INTEGER
        DECLARE Temp : INTEGER
        DECLARE Max : INTEGER


        Pass <- MaxIndex - 1

        FOR Pass <- 1 TO MaxIndex - 1
                FOR Index <- 1 TO Max-1 - Pass
                        IF Score[Index] > Score[Index + 1] THEN
                                Temp <- Score[Index]
                                Score[Index] <- Score[Index+1]
                                Score[Index +1] <- Temp
                        ENDIF
                NEXT Index
                Pass <- Pass - 1
        NEXT Pass
ENDPROCEDURE
```

Q7.

Q7.

```
PROCEDURE BinarySearch()
        DECLARE Index : INTEGER
        DECLARE Middle : INTEGER
        DECLARE Low : INTEGER
        DECLARE High: INTEGER
        DECLARE Found : BOOLEAN

INPUT SearchID
High <- 16
Low <- 0
Found <- FALSE

WHILE High > Low AND Found = FALSE DO
        Middle <- (High + Low) DIV 2
        IF StudentID[Middle] = SearchID THEN
                Found <- TRUE
        ELSE
                IF StudentID[Middle] > SearchID THEN
                        High <- Middle - 1
                ELSE
                        Low <- Middle + 1
                ENDIF
        ENDIF
ENDWHILE

IF Found = TRUE THEN
        OUTPUT “Student found”
ELSE
        OUTPUT “Student not found”
ENDIF
```

Q8.

```
PROCEDURE BinarySearch()
        DECLARE Middle : INTEGER
        DECLARE Low : INTEGER
        DECLARE High: INTEGER
        DECLARE Found : BOOLEAN
        DECLARE Comparisons : INTEGER

INPUT SearchCode
High <- 20
Low <- 0
Found <- FALSE
Comparisons <- 0

WHILE High > Low AND Found = FALSE DO
        Comparisons <- Comparisons + 1
        Middle <- (High + Low) DIV 2
        IF ProductCode[Middle] = SearchCode THEN
                Found <- TRUE
        ELSE
                IF ProductCode[Middle] > SearchCode THEN
                        High <- Middle - 1
                ELSE
                        Low <- Middle + 1
                ENDIF
        ENDIF
ENDWHILE

OUTPUT “Number of comparisons: “,Comparisons

IF Found = TRUE THEN
        OUTPUT “Student found”
ELSE
        OUTPUT “Student not found”
ENDIF
```

Bryan

```
3. A. DECLARE WordList : ARRAY [1:500] OF STRING
FOR Index <- 1 TO 500

        WordList[Index] <- " "
NEXT Index

NumberOfWords <- 0

B. PROCEDURE OutputList
        FOR Index <- 1 TO NumberOfWords
                OUTPUT WordList[Index]
        Next Index
ENDPROCEDURE

C. PROCEDURE LoadWords
        DECLARE FileName : STRING

        OUTPUT "Enter filename: "
        INPUT FileName

        IF NOT FILEEXIST(FileName)
                THEN
                        OUTPUT "Error: File doesn't exist"
                ELSE
                        OPENFILE FileName FOR READ

                        NumberOfWords <- 0

                        WHILE Not EOF(FileName) AND NumberOfWords < 500
                                NumberOfWords <- NumberOfWords + 1
                                READFILE FileName, WordList[NumberOfWords]
                        ENDWHILE

                        CLOSEFILE FileName

                        IF NumberOfWords = 500 AND NOT EOF(FileName)
                                THEN
                                        OUTPUT "Error: No more words"
                ENDIF
        ENDIF
ENDPROCEDURE

D. PROCEDURE SortWords
   DECLARE Temp : STRING
   DECLARE Swapped : BOOLEAN

   REPEAT
      Swapped <- FALSE
```

```
        FOR Index <- 1 TO NumberOfWords - 1
            IF WordList[Index] > WordList[Index + 1]
                THEN
                    Temp <- WordList[Index]
                    WordList[Index] <- WordList[Index + 1]
                    WordList[Index + 1] <- Temp

                    Swapped <- TRUE
            ENDIF
        NEXT Index

    UNTIL Swapped = FALSE
ENDPROCEDURE
```

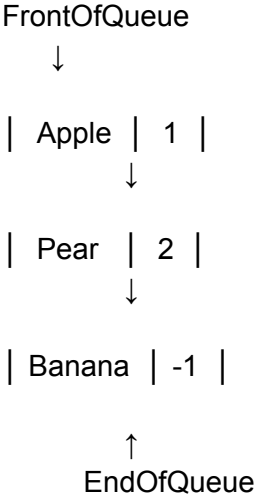
E. LoadWords
OutputList
SortWords
OutputList

Darren

```
FUNCTION FindName(s : STRING) RETURNS INTEGER
   Index ← -1
   First ← 0
   Last ← 50

   WHILE (Last >= First) AND (Index = -1) DO
      Middle ← (First + Last) DIV 2

      IF Names[Middle] = s
      THEN
         Index ← Middle
      ELSE
         IF Names[Middle] < s
         THEN
            First ← Middle + 1
         ELSE
            Last ← Middle - 1
         ENDIF
      ENDIF
   ENDWHILE

   RETURN Index
ENDFUNCTION
```

b)
The data must be in alphabetical/ascending order (sorted order).

c)
i. If the name exists, the function returns the index of the name.
ii. If the name does not exist, the function returns -1.



2b)

```
TYPE Node
   DECLARE Data : STRING
   DECLARE Pointer : INTEGER
ENDTYPE

DECLARE Queue : ARRAY[0 : 49] OF Node

FrontOfQueue ← -1
EndOfQueue ← -1
StartOfFreeList ← 0

FOR Index ← 0 TO 48
   Queue[Index].Pointer ← Index + 1
NEXT Index

Queue[49].Pointer ← -1
```

2c)

Identifier	Data type	Description
NullPointer	INTEGER	Constant set to -1
Queue	ARRAY OF Node	Array to store queue data
NewItem	STRING	Value to be added
StartOfFreeList	INTEGER	Pointer to next free node in array
FrontOfQueue	INTEGER	Pointer to first node in queue
EndOfQueue	INTEGER	Pointer to last node in queue
NewNodePointer	INTEGER	Pointer to node to be added

2cii)

```
PROCEDURE JoinQueue(NewItem : STRING)

   IF StartOfFreeList = NullPointer
   THEN
      Report Error
   ELSE
      NewNodePointer ← StartOfFreeList
      Queue[NewNodePointer].Data ← NewItem

      StartOfFreeList ← Queue[NewNodePointer].Pointer
      Queue[NewNodePointer].Pointer ← NullPointer

      IF FrontOfQueue = NullPointer
      THEN
         FrontOfQueue ← NewNodePointer
         EndOfQueue ← NewNodePointer
      ELSE
```

```
            Queue[EndOfQueue].Pointer ← NewNodePointer
            EndOfQueue ← NewNodePointer
        ENDIF
    ENDIF

ENDPROCEDURE

3a)
DECLARE WordList : ARRAY[1 : 500] OF STRING

FOR Index ← 1 TO 500
    WordList[Index] ← ""
NEXT Index

3b)
PROCEDURE OutputList()

    FOR Index ← 1 TO 500
        IF WordList[Index] <> ""
        THEN
            OUTPUT WordList[Index]
        ENDIF
    NEXT Index

ENDPROCEDURE

3c)
PROCEDURE LoadWords()

    DECLARE FileName : STRING
    DECLARE Index : INTEGER
    DECLARE Word : STRING

    OUTPUT "Enter filename"
    INPUT FileName

    OPENFILE FileName FOR READ

    IF EOF(FileName)
    THEN
        Report Error
    ELSE
        Index ← 1

        WHILE NOT EOF(FileName)
            INPUTFILE FileName, Word

            IF Index > 500
```

```
            THEN
               Report Error
            ELSE
               WordList[Index] ← Word
               Index ← Index + 1
            ENDIF
         ENDWHILE

         CLOSEFILE FileName
      ENDIF

ENDPROCEDURE
```

Q1. on paper
Q2.

```
DECLARE ProductCode : ARRAY[1:15] OF INTEGER
DECLARE Target, Index, Count : INTEGER
DECLARE Found : BOOLEAN

OUTPUT "Enter product code to search for:"
INPUT Target

Found ← FALSE
Index ← 1
Count ← 0

WHILE Index <= 15 AND Found = FALSE
   Count ← Count + 1
   IF ProductCode[Index] = Target THEN
      Found ← TRUE
   ELSE
      Index ← Index + 1
   ENDIF
ENDWHILE

IF Found = TRUE THEN
   OUTPUT "Found"
ELSE
   OUTPUT "Not found"
ENDIF

OUTPUT "Number of comparisons: ", Count
```

Q4.

```
DECLARE Temperature : ARRAY [1:8] OF INTEGER
```

```
PROCEDURE BubbleSort(BYREF Temperature : ARRAY[1:8] OF INTEGER)
   DECLARE i, j, Temp : INTEGER
   DECLARE Swapped : BOOLEAN

   I <- 1
   Swapped <- TRUE

   WHILE i <= 7 AND Swapped = TRUE
      Swapped <- FALSE
      FOR j <- 1 TO 8 - i
         IF Temperature[j] > Temperature[j + 1] THEN
            Temp <- Temperature[j]
            Temperature[j] <- Temperature[j + 1]
            Temperature[j+1] <- Temp
            Swapped <- TRUE
         ENDIF
     NEXT j
     I <- i + 1
   ENDWHILE
ENDPROCEDURE
```

Costa

Q1.

```
PROCEDURE FindStudent(TargetID : INTEGER)
   DECLARE Index : INTEGER
   DECLARE Found : BOOLEAN

   Index <- 1
   Found <- FALSE

   WHILE (Index <= 10) AND (Found = FALSE) DO
      IF StudentID[Index] = TargetID THEN
         Found <- TRUE
         OUTPUT StudentName[Index]
      ELSE
         Index <- Index + 1
      ENDIF
   ENDWHILE

   IF Found = FALSE THEN
      OUTPUT "Student not found"
   ENDIF
ENDPROCEDURE
```

Q2a.

```
DECLARE SearchCode : INTEGER
DECLARE Index : INTEGER
DECLARE Found : BOOLEAN

OUTPUT "Enter product code to search:"
INPUT SearchCode

Index <- 1
Found <- FALSE

WHILE (Index <= 15) AND (Found = FALSE) DO
   IF ProductCode[Index] = SearchCode THEN
      Found <- TRUE
   ELSE
      Index <- Index + 1
   ENDIF
ENDWHILE

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

Q2b.

```
DECLARE SearchCode : INTEGER
DECLARE Index : INTEGER
DECLARE Found : BOOLEAN
DECLARE Comparisons : INTEGER

OUTPUT "Enter product code to search:"
INPUT SearchCode

Index <- 1
Found <- FALSE
Comparisons <- 0

WHILE (Index <= 15) AND (Found = FALSE) DO
   Comparisons <- Comparisons + 1
   IF ProductCode[Index] = SearchCode THEN
      Found <- TRUE
   ELSE
      Index <- Index + 1
   ENDIF
ENDWHILE

IF Found = TRUE THEN
   OUTPUT "Found"
ELSE
   OUTPUT "Not found"
ENDIF

OUTPUT "Number of comparisons: ", Comparisons
```

Q2c.

125: 1 comparison
278: 8 comparisons
176: 15 comparisons
500: 15 comparisons

Q7.

```
PROCEDURE BinarySearch(StudentID : ARRAY, SearchValue : INTEGER)
   DECLARE Low : INTEGER
   DECLARE High : INTEGER
   DECLARE Middle : INTEGER
   DECLARE Found : BOOLEAN

   Low ← 1
```

```
    High ← 16
    Found ← FALSE

    WHILE (Low <= High) AND (Found = FALSE) DO
        Middle ← (Low + High) DIV 2

        IF StudentID[Middle] = SearchValue THEN
            Found ← TRUE
        ELSE
            IF StudentID[Middle] > SearchValue THEN
                High ← Middle - 1
            ELSE
                Low ← Middle + 1
            ENDIF
        ENDIF
    ENDWHILE

    IF Found = TRUE THEN
        OUTPUT "Student found"
    ELSE
        OUTPUT "Student not found"
    ENDIF
ENDPROCEDURE
```

Q8.

```
PROCEDURE SearchProduct(ProductCode : ARRAY, SearchCode : INTEGER)
    DECLARE Low : INTEGER
    DECLARE High : INTEGER
    DECLARE Middle : INTEGER
    DECLARE Found : BOOLEAN
    DECLARE Comparisons : INTEGER

    Low ← 1
    High ← 20
    Found ← FALSE
    Comparisons ← 0

    WHILE (Low <= High) AND (Found = FALSE) DO
        Middle ← (Low + High) DIV 2
        Comparisons ← Comparisons + 1

        IF ProductCode[Middle] = SearchCode THEN
            Found ← TRUE
        ELSE
```

```
            IF ProductCode[Middle] > SearchCode THEN
                High ← Middle - 1
            ELSE
                Low ← Middle + 1
            ENDIF
        ENDIF
    ENDWHILE

    IF Found = TRUE THEN
        OUTPUT "Product available"
    ELSE
        OUTPUT "Product unavailable"
    ENDIF

    OUTPUT "Number of comparisons made: ", Comparisons
ENDPROCEDURE
```