ARRAYS

```
=================================================================
PSEUDO CODE VERSION
-------------------------------------------------------------------------------------------------------

'These algorithms assume that an array has been created with the name arr of MAX subscripts.

MAX = 10 'this is a constant defining the limit for the array

'Procedure to initialize an array
PROCEDURE InitializeArray
  FOR x = 1 TO MAX
    arr(x) = 0
  NEXT
END PROCEDURE

'-----------------------------------------------------
'Procedure for inputing values in the array
PROCEDURE ArrayWrite
  FOR x = 1 TO MAX
     INPUT "Enter a value: "; arr(x)
  NEXT
END PROCEDURE

'------------------------------------------------------
'Procedure for outputing values from the array
PROCEDURE ArrayRead
  FOR x = 1 to MAX
      PRINT arr(x)
  NEXT
END PROCEDURE

'------------------------------------------------------
'Procedure to search a value in the array using Serial Search method
'This procedure will search for all occurances of 'v' in the array.

PROCEDURE ArraySearch(v)
  Flag = False
  FOR x = 1 TO MAX
     IF arr(x) = v THEN
         PRINT "Found at position: ", x
         Flag = True
     END IF
  NEXT

  IF Flag = False THEN
     PRINT "Value not found."
  END IF
END PROCEDURE
```

```
'---------------------------------------------------------
'This is the same procedure but it will quit as soon as it find the value in the array.

PROCEDURE ArraySearch(v)
   Flag = False
   FOR x = 1 TO MAX
      IF arr(x) = v THEN
          PRINT "Found at position: ", x
          Flag = True
          EXIT FOR
      END IF
   NEXT

   IF Flag = False THEN
      PRINT "Value not found."
   END IF
END PROCEDURE

'---------------------------------------------------------
'This procedure will copy the contents of one array into another
'This procedure assumes that the second array is equal or greater in size than
'the first array.

PROCEDURE ArrayCopy
   FOR x = 1 TO MAX
      arr2(x) = arr(x)
   NEXT
END PROCEDURE
```

==
STRUCTURED ENGLISH VERSION

```
PROCEDURE InitializeArray
  1. Let x = 1
  2. Assign zero to arr(x)
  3. Increment x by 1
  4. Repeat step 2 and 3 until x is greater than MAX
END PROCEDURE
---------------------------------------------------------------

PROCEDURE ArrayWrite
  1. Let x = 1
  2. Input value from user in arr(x)
  3. Increment x by 1
  4. Repeat steps 2 and 3 until x is greater than MAX
END PROCEDURE
--------------------------------------------------------------
```

```
PROCEDURE ArrayRead
 1. Let x = 1
 2. Print value in arr(x)
 3. Increment x by 1
 4. Repeat steps 2 and 3 until x is greater than MAX
END PROCEDURE
---------------------------------------------------------------

'Search all values
PROCEDURE ArraySearch(v)
 1. Set Flag to False
 2. Let x = 1
 3. If value in arr(x) is equal to v then
   3.1 Display that value was found at position x
   3.2 Set Flag to True
 4. Increment x by 1
 5. Repeat steps 3 to 4 until x is greater than MAX
 6. If Flag is False then
   6.1 Display error message that value was not found
END PROCEDURE
-------------------------------------------------------------------

'Search first value
PROCEDURE ArraySearch(v)
 1. Set Flag to False
 2. Let x = 1
 3. If value in arr(x) is equal to v then
    3.1 Display that value was found at v
    3.2 Set Flag to True
    3.3 Goto step 6
 4. Increment x by 1
 5. Repeat steps 3 to 4 until x is greater than MAX
 6. If Flag = False then
    6.1 Display error message that value was not found
END PROCEDURE
--------------------------------------------------------------------

'Copies contents of one array into another
PROCEDURE ArrayCopy
 1. Let x = 1
 2. Store arr(x) into arr2(x)
 3. Increment x by 1
 4. Repeat steps 2 and 3 until x is greater than MAX
END PROCEDURE
```

==

STACKS

===

```
PSEUDO CODE VERSION
-----------------------------------------------------------------------------------------------------
'These algorithms assume that a stack is being implemented using an array called 'stack'
MAX = 10 'This is the limit of the array
Let x be the counter variable for the stack

'----------------------------------------------------------
'This is the call to the Push procedure to write value in 'v' on the stack.
Call Push(v)

'Procedure to push (write) values onto the stack.
PROCEDURE Push(v)
  IF x > MAX THEN
    PRINT "Stack is full. Can't write value."
  ELSE
    stack(x) = v
    x = x + 1
  END IF
END PROCEDURE

'------------------------------------------------------------
'This procedure pops (reads) a value from the stack.
PROCEDURE Pop
  IF x = 1 THEN
    PRINT "Stack is empty. Nothing to read."
  ELSE
/     x =x - 1
    PRINT stack(x)
  END IF
END PROCEDURE


==============================================================
STRUCTURED ENGLISH VERSION
-----------------------------------------------------------------------------------------------------

'This procedure will push (write) a value onto the stack.
PROCEDURE Push(v)
 1. if counter x is greater than MAX then
   a. output error and stop
 2. otherwise
   a. store value v in subscript stack(x)
   b. increment counter x by 1
END PROCEDURE

---------------------------------------------------------

'This procedure will pop (read) a value from stack
PROCEDURE Pop
 1. if counter is equal to 1 then
```

```
   a. output error and stop
 2. otherwise
   a. decrement counter x by 1
   b. output value from subscript stack(x)
END PROCEDURE

===============================================================

QUEUES

=====================================================
PSEUDO CODE VERSION
-----------------------------------------------------------------------------------------------
'These algorithms assume that a queue is being implemented using an array called 'que'
MAX = 10 'This is the limit of the array
Let x be the counter variable for the queue

'-----------------------------------------------------------------
Call WriteQue(v)

'This procedure will write a value in the queue
PROCEDURE WriteQue(v)
  IF x > MAX THEN
    PRINT "Queue is full. Can't write value."
  ELSE
    que(x) = v
    x = x + 1
  END IF
END PROCEDURE

'-----------------------------------------------------------------
'This procedure will read a value form the queue. Note that it will always be the first value
'as queue is FIFO (First In First Out).

PROCEDURE ReadQue
  IF x = 1 THEN
    PRINT "Queue is empty. Can't read value."
  ELSE
    PRINT que(1)              'read from the front of the queue
    FOR m = 1 TO (x-2)        'now move rest of the values one step forward
       que(m) = que(m+1)
    NEXT
    x = x - 1                 'decrement the counter
  ENDIF
END PROCEDURE

=====================================================
STRUCTURED ENGLISH VERSION
-----------------------------------------------------------------------------------------------
'This procedure will write a value in the queue
```

```
PROCEDURE WriteQue(v)
 1. if counter x is greater than MAX
    a.  report error and stop
 2.otherwise
    a. store value v in subscript que(x)
    b. increment x by 1
END PROCEDURE
-------------------------------------------------------

'This procedure will read a value from the queue
PROCEDURE ReadQue
 1. if counter x is equal to 1 then
    a. report error and stop
 2. otherwise
    a. output value in subscript que(1)
    b. run a loop using counter m from 1 to x-2
    c. store contents of subscript que(m+2) into que(m)
    d. increment counter m by 1
    e. repeat step c and d until counter m is greater than (x-2)
    f.  decrement x by 1
END PROCEDURE

=================================================================

LINKED LISTS

=============================================================
PSEUDO CODE VERSION
---------------------------------------------------------------------------------------------------------------------

'Four pointers are assumed as follows
'HEAD points to the first link in the list
'NEW points to the new node
'CURR points to the current node being examined
'TRAIL point to the node immediately prior to the current node being examined

'These algorithms assume that free nodes are acquired from the OS using a function called
'malloc() that returns the address of the new node.
'These algorithms are for single pointer linked lists.

'Procedure for adding a new node in ascending order of data
PROCEDURE NewNode
   NEW = malloc()                          'ask for a new node
   NEW.NEXT = Null                         'make the NEXT pointer of NEW point to NULL
   IF HEAD = Null THEN                     'in case the list is empty
      HEAD = NEW                           'make the new node the first node in the list
   ELSEIF NEW.Data <= HEAD.Data THEN   'in case the new node belongs at the start of the list
      NEW.NEXT = HEAD                      'make the new node the first node in the list
      HEAD = NEW                           'make the HEAD pointer point to the NEW node
   ELSE                                    'in case the new node belongs in the middle or end
```

```
      CURR = HEAD.NEXT                       'CURR points to the second node
      TRAIL = HEAD                           'TRAIL follows by starting at the first node
'search the list until either CURR.Data is greater than the NEW or end of list is reached
      DO WHILE CURR.Data < NEW.Data AND CURR.NEXT <> NULL
          TRAIL = CURR                       'TRAIL follows CURR on step behind
          CURR = CURR.NEXT                   'CURR moves to the next node
      END WHILE
'examine why the loop was broken.
'last node reached and the NEW.Data is greater than all data
      IF CURR.Data < NEW.Data AND CURR.NEXT = NULL THEN
          CURR.NEXT = NEW                    'join NEW at the end of the list
      ELSEIF CURR.Data > NEW.Data THEN               'NEW belongs somewhere in the middle
           TRAIL.NEXT = NEW                  'make the rear connection
           NEW.NEXT = CURR                   'make the forward connection
      END IF
END PROCEDURE


'Procedure to delete a node from a linked list
PROCEDURE DeleteNode(Item)
   IF HEAD.NEXT = NULL THEN                          'if there are no nodes in the list
      OUTPUT "List is empty. Nothing to delete."     'nothing to delete
   ELSE IF HEAD.Data = Item THEN                     'if its the first node to be deleted
      HEAD = HEAD.NEXT                               'move HEAD to the next node
   ELSE                                              'target node is somewhere in the middle
      CURR = HEAD.NEXT                               'start CURR on the second node
      TRAIL = HEAD                                   'TRAIL follows behind starting at HEAD
'search the list until target node found of end of list reached
      DO WHILE CURR.NEXT <> NULL AND CURR.Data <> Item
            TRAIL = CURR                             'move TRAIL one node forward
            CURR = CURR.NEXT                         'move CURR one node forward
      END WHILE
'examine why the loop was broken.
'if end of list reached and target node not found
      IF CURR.Data <> Item AND CURR.NEXT = NULL THEN
            OUTPUT "Item not found."                 'nothing to delete.
      ELSEIF CURR.Data = Item THEN                   'target node found
'set TRAIL to CURR.NEXT deleting the node in the middle
            TRAIL.NEXT = CURR.NEXT
      END IF
   END IF
END PROCEDURE
```

===

STRUCTURED ENGLISH VERSION

```
'Procedure to add a new node in the linked list
PROCEDURE NewNode
 1. generate a new node and let NEW pointer point to the new node
 2. if HEAD is pointing to NULL then
   a. make HEAD pointer equal to NEW and stop
 3. otherwise if Data in NEW node is less than or equal to Data in HEAD then
   a. make the NEXT pointer of NEW point to HEAD
   b. make HEAD pointer point to NEW
 4. otherwise
   a. point CURR pointer to the NEXT pointer of HEAD
   b. point TRAIL pointer to HEAD
   c. begin loop
     e. point TRAIL pointer to CURR
     f.  point CURR pointer to the NEXT pointer of CURR
   g. repeat steps e and f while Data in node pointed by CURR is less than Data in NEW node
      and NEXT pointer of CURR is not equal to NULL
   h. if Data in CURR is less than Data in NEW and NEXT pointer of CURR is equal to NULL then
     i.  point NEXT pointer of CURR to NEW
   j.  otherwise if Data in CURR is greater than Data in NEW then
    k. point NEXT pointer of TRAIL to NEW
    l.  point NEXT pointer of NEW to CURR
END PROCEDURE

-----------------------------------------------------------------------------------------------

'Procedure to delete a node from the linked list
PROCEDURE DeleteNode(Item)
 1. if the HEAD pointer is pointing to NULL then
   a. output error and stop
 2. otherwise if the Data in the HEAD is equal to the Data in Item then
   a. make HEAD pointer point to the NEXT pointer of HEAD
 3. otherwise
   a. point CURR pointer to the NEXT of HEAD pointer
   b. point TRAIL pointer to the HEAD pointer
   c. begin loop
     d. point TRAIL pointer to the CURR pointer
     e. pointer CURR pointer to the NEXT of CURR pointer
   f.  repeat steps d and e while NEXT of CURR pointer is not NULL
      and Data in CURR is not equal to Data in Item
   g. if Data in CURR is not equal to Data in Item and Next of CURR pointer is NULL then
     h. output error. Item not found. stop.
   i.  otherwise if Data in CURR is equal to Data in Item then
     j. point the NEXT of TRAIL pointer to the NEXT of CURR pointer
END PROCEDURE
=====================================================================
```