LINKED LISTS

When an array of nodes is first initialised to work as a linked list, the linked list will be empty. So the start pointer will be the null pointer. All nodes need to be linked to form the free list. Figure 23.13 shows an example of an implementation of a linked list before any data is inserted into it.

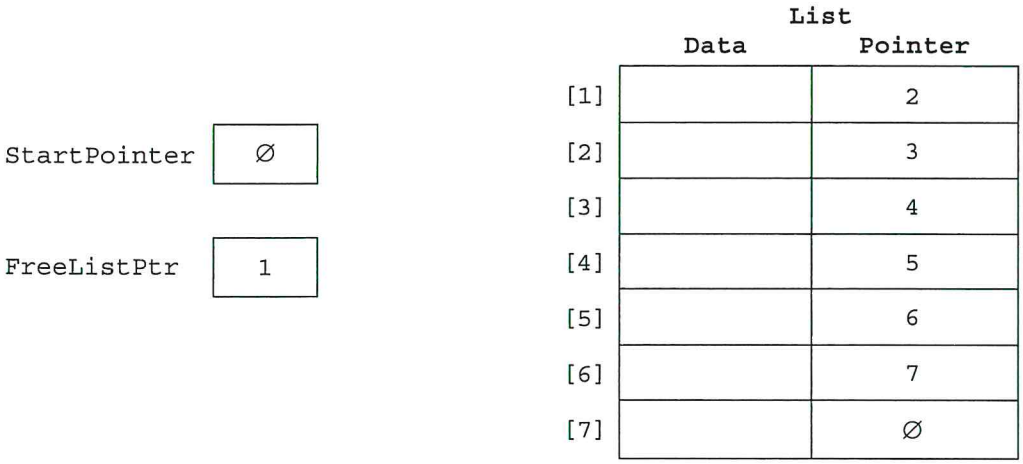

Figure 23.13 A linked list before any nodes are used

We now code the basic operations discussed using the conceptual diagrams in Figures 23.05 to 23.12.

Create a new linked list

initialised

```
// NullPointer should be set to -1 if using array element with index 0
CONSTANT NullPointer = 0
// Declare record type to store data and pointer
TYPE ListNode
   DECLARE Data    : STRING
   DECLARE Pointer : INTEGER
ENDTYPE
DECLARE StartPointer : INTEGER
DECLARE FreeListPtr : INTEGER
DECLARE List[1 : 7] OF ListNode

PROCEDURE InitialiseList
   StartPointer ← NullPointer     // set start pointer
   FreeListPtr ← 1                // set starting position of free list
   FOR Index ← 1 TO 6             // link all nodes to make free list
      List[Index].Pointer ← Index + 1
   ENDFOR
   List[7].Pointer ← NullPointer // last node of free list
ENDPROCEDURE
```

Insert a new node into an ordered linked list

```
PROCEDURE InsertNode(NewItem)
   IF FreeListPtr <> NullPointer
      THEN  // there is space in the array
         // take node from free list and store data item
         NewNodePtr ← FreeListPtr
         List[NewNodePtr].Data ← NewItem
         FreeListPtr ← List[FreeListPtr].Pointer
         // find insertion point
         ThisNodePtr ← StartPointer            // start at beginning of list
         WHILE ThisNodePtr <> NullPointer      // while not end of list
               AND List[ThisNodePtr].Data < NewItem
            PreviousNodePtr ← ThisNodePtr      // remember this node
                                               // follow the pointer to the next node
            ThisNodePtr ← List[ThisNodePtr].Pointer
         ENDWHILE
         IF PreviousNodePtr = StartPointer
            THEN  // insert new node at start of list
               List[NewNodePtr].Pointer ← StartPointer
               StartPointer ← NewNodePtr
            ELSE  // insert new node between previous node and this node
               List[NewNodePtr].Pointer ← List[PreviousNodePtr].Pointer
               List[PreviousNodePtr].Pointer ← NewNodePtr
         ENDIF
   ENDIF
ENDPROCEDURE
```

PreviousNodePtr = NullPointer

IF startPointer = NullPointer OR PreviousNodePtr = NullPointer

After three data items have been added to the linked list, the array contents are as shown in Figure 23.14.

StartPointer 1

FreeListPtr 4

	List Data	List Pointer
[1]	B	2
[2]	D	3
[3]	L	∅
[4]		5
[5]		6
[6]		7
[7]		∅

Figure 23.14 Linked list of three nodes and free list of four nodes

Find an element in an ordered linked list

```
FUNCTION FindNode(DataItem) RETURNS INTEGER  // returns pointer to node
   CurrentNodePtr ← StartPointer  // start at beginning of list
   WHILE CurrentNodePtr <> NullPointer  // not end of list
         AND List[CurrentNodePtr].Data <> DataItem  // item not found
      // follow the pointer to the next node
      CurrentNodePtr ← List[CurrentNodePtr].Pointer
   ENDWHILE
   RETURN CurrentNodePtr  // returns NullPointer if item not found
ENDFUNCTION
```

Delete a node from an ordered linked list

```
PROCEDURE DeleteNode(DataItem)
   ThisNodePtr ← StartPointer                  // start at beginning of list
   WHILE ThisNodePtr <> NullPointer             // while not end of list
         AND List[ThisNodePtr].Data <> DataItem  // and item not found
      PreviousNodePtr ← ThisNodePtr  // remember this node
                                     // follow the pointer to the next node
      ThisNodePtr ← List[ThisNodePtr].Pointer
   ENDWHILE
   IF ThisNodePtr <> NullPointer  // node exists in list
      THEN
         IF ThisNodePtr = StartPointer  // first node to be deleted
            THEN
               StartPointer ← List[StartPointer].Pointer
            ELSE
               List[PreviousNodePtr] ← List[ThisNodePtr].Pointer
         ENDIF
         List[ThisNodePtr].Pointer ← FreeListPtr
         FreeListPtr ← ThisNodePtr
   ENDIF
ENDPROCEDURE
```

Access all nodes stored in the linked list

```
PROCEDURE OutputAllNodes
   CurrentNodePtr ← StartPointer // start at beginning of list
   WHILE CurrentNodePtr <> NullPointer  // while not end of list
      OUTPUT List[CurrentNodePtr].Data
      // follow the pointer to the next node
      CurrentNodePtr ← List[CurrentNodePtr].Pointer
   ENDWHILE
ENDPROCEDURE
```


TASK 23.04

Convert the pseudocode for the linked-list handling subroutines to program code. Incorporate the subroutines into a program and test them.

Note that a stack ADT and a queue ADT can be treated as special cases of linked lists. The linked list stack only needs to add and remove nodes from the front of the linked list. The linked list queue only needs to add nodes to the end of the linked list and remove nodes from the front of the linked list.

TASK 23.05

Write program code to implement a stack as a linked list. Note that the adding and removing of nodes is much simpler than for an ordered linked list.