19.2.2 How a compiler implements recursion

Recursive code needs to make use of the stack; therefore, in order to implement recursive procedures and functions in a high-level programming language, a compiler must produce object code that pushes return addresses and values of local variables onto the stack with each recursive call, winding. The object code then pops the return addresses and values of local variables off the stack, unwinding.

ACTIVITY 19T

1 Explain what is meant by *recursion* and give the benefits of using recursion in programming.

2 Explain why a compiler needs to produce object code that uses the stack for a recursive procedure.

End of chapter questions

1 Data is stored in the array NameList[1:10]. This data is to be sorted.

a) i) Copy and complete this pseudocode algorithm for an insertion sort.

[7]

```
FOR ThisPointer ← 2 TO .........................................
    // use a temporary variable to store item which is to
    // be inserted into its correct location
    Temp ← NameList[ThisPointer]
    Pointer ← ThisPointer – 1
    WHILE (NameList[Pointer] > Temp) AND ....................
        // move list item to next location
        NameList[....................] ← NameList[....................]
        Pointer ← .............................................
    ENDWHILE
    // insert value of Temp in correct location
    NameList[.................................] ← .........................
ENDFOR
```

ii) A special case is when NameList is already in order. The algorithm in part a) i) is applied to this special case.
Explain how many iterations are carried out for each of the loops.

[3]

b) An alternative sort algorithm is a bubble sort:

```
FOR ThisPointer ← 1 TO 9
    FOR Pointer ← 1 TO 9
        IF NameList[Pointer] > NameList[Pointer + 1]
          THEN
            Temp ← NameList[Pointer]
            NameList[Pointer] ← NameList[Pointer + 1]
            NameList[Pointer + 1] ← Temp
        ENDIF
    ENDFOR
ENDFOR
```

i) As in part a) ii), a special case is when NameList is already in order. The algorithm in part b) is applied to this special case.
Explain how many iterations are carried out for each of the loops.

[2]

ii) Rewrite the algorithm in part b), using **pseudocode**, to reduce the number of unnecessary comparisons.
Use the same variable names where appropriate.

[5]

Cambridge International AS & A Level Computer Science 9608
Paper 41 Q5 June 2015

2 A Queue Abstract Data type (ADT) has these associated operations:

– create queue
– add item to queue
– remove item from queue

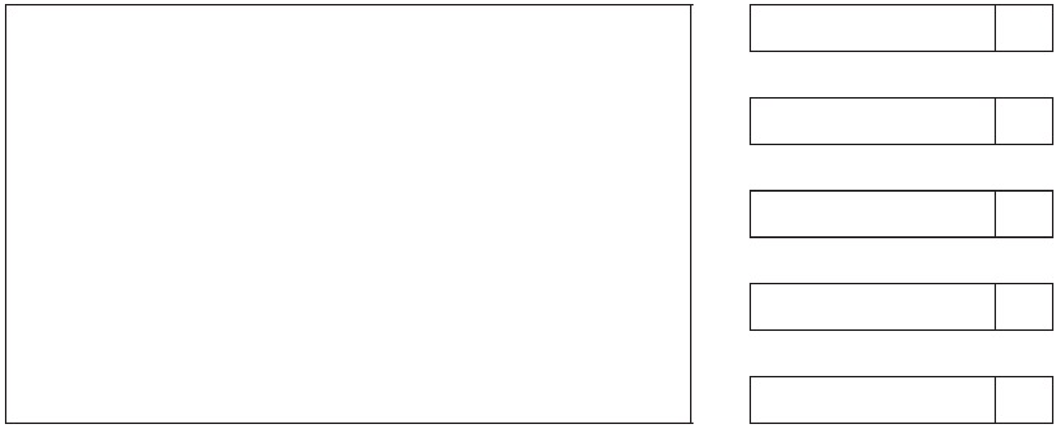
The queue ADT is to be implemented as a linked list of nodes.
Each node consists of data and a pointer to the next node.

a) The following operations are carried out:

```
CreateQueue
AddName("Ali")
AddName("Jack")
AddName("Ben")
AddName("Ahmed")
RemoveName
AddName("Jatinder")
RemoveName
```

Copy the diagram and add appropriate labels to show the final state of the queue. Use the space on the left as a workspace.
Show your final answer in the node shapes on the right.

[3]



b) Using pseudocode, a record type, Node, is declared as follows:

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

The statement

```
DECLARE Queue : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array Queue.

i) The CreateQueue operation links all nodes and initialises the three pointers that need to be used: HeadPointer, TailPointer and FreePointer.
Copy and complete the diagram to show the value of all pointers after CreateQueue has been executed.

[4]

HeadPointer

TailPointer

FreePointer

Queue

	Name	Pointer
[1]		
[2]		
[3]		
[4]		
[5]		
[6]		
[7]		
[8]		
[9]		
[10]		

ii) The algorithm for adding a name to the queue is written, using pseudocode, as a procedure with the header:

```
PROCEDURE AddName(NewName)
```

where NewName is the new name to be added to the queue.
The procedure uses the variables as shown in the identifier table.

Identifier	Data type	Description
Queue	Array[1:10] OF Node	Array to store node data
NewName	STRING	Name to be added
FreePointer	INTEGER	Pointer to next free node in array
HeadPointer	INTEGER	Pointer to first node in queue
TailPointer	INTEGER	Pointer to last node in queue
CurrentPointer	INTEGER	Pointer to current node

```
PROCEDURE AddName(BYVALUE NewName : STRING)
   // Report error if no free nodes remaining
   IF FreePointer = 0
     THEN
       Report Error
     ELSE
       // new name placed in node at head of
       free list
        CurrentPointer ← FreePointer
        Queue[CurrentPointer].Name ← NewName
       // adjust free pointer
       FreePointer ← Queue[CurrentPointer].
       Pointer
        // if first name in queue then adjust
        head pointer
        IF HeadPointer = 0
         THEN
           HeadPointer ← CurrentPointer
      ENDIF
     // current node is new end of queue
     Queue[CurrentPointer].Pointer ← 0
     TailPointer ← CurrentPointer
   ENDIF
ENDPROCEDURE
```

Copy and complete the **pseudocode** for the procedure RemoveName. Use the variables listed in the identifier table.

[6]

```
PROCEDURE RemoveName()
   // Report error if Queue is empty
.....................................................................
.....................................................................
.....................................................................
.....................................................................
   OUTPUT Queue[..............................].Name
   // current node  is head of queue
.....................................................................
   // update head pointer
.....................................................................
   // if only one element in queue then update tail
      pointer
.....................................................................
.....................................................................
.....................................................................
.....................................................................
   // link released node to free list
.....................................................................
.....................................................................
.....................................................................
ENDPROCEDURE
```

Cambridge International AS & A Level Computer Science 9608
Paper 41 Q6 June 2015