H

MINISTRY OF EDUCATION, SINGAPORE
in collaboration with
UNIVERSITY OF CAMBRIDGE LOCAL EXAMINATIONS SYNDICATE
General Certificate of Education Advanced Level
Higher 2

COMPUTING

9597/01

Paper 1

October/November 2014

3 hours 15 minutes

5420091874

Additional Materials: Pre-printed A4 Paper
Removable storage device
Electronic version of `ANIMALS.TXT` data file
Electronic version of `JARGON.TXT` data file
Electronic version of `PSEUDOCODE_TASK_3_4.TXT` file
Electronic version of `EVIDENCE.DOC` file

READ THESE INSTRUCTIONS FIRST

Type in the `EVIDENCE.DOC` document the following:

- Candidate details
- Programming language used

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

All tasks and required evidence are numbered.

The number of marks is given in brackets [ ] at the end of each task.

Copy and paste required evidence of program listing and screenshots into the `EVIDENCE.DOC` document.

At the end of the examination, print out your `EVIDENCE.DOC` document and fasten your printed copy securely together.

This document consists of **14** printed pages and **2** blank pages.

Singapore Examinations and Assessment Board

CAMBRIDGE
International Examinations

[Turn over

1 Many applications require the user to search for a data item from a file or one-dimensional array.

JARGON.TXT is a text file containing computing terms with one term per line. The program will read all the terms from JARGON.TXT into an array.

The user will choose if the type of search is to find:

1. An exact match
2. A match at the beginning of the term text
3. A match anywhere within the term text

Task 1.1
Design and write program code to:

- Read the entire contents of JARGON.TXT to an array
- Allow the user to repeatedly select the type of search, then input a term
- Output the matching term(s) found
- Output a count of the number of matches
- End with the input of term “XXX”

A typical run of the program is shown below:

```
+++++++++++++++++++++++
1. Exact Match
2. Start of term
3. Within term
++++++++++++++++++
Choice ?1
Term?firewall
firewall
There were 1 matching term(s)

+++++++++++++++++++++++
1. Exact Match
2. Start of term
3. Within term
++++++++++++++++++
Choice ?2
Term?data
data flow diagram
database management system
data file
database
There were 4 matching term(s)

+++++++++++++++++++++++
1. Exact Match
2. Start of term
3. Within term
++++++++++++++++++
Choice ?3
Term?box
white box testing
black box testing
There were 2 matching term(s)
```

Evidence 1
The program code. [11]

Task 1.2
Study the contents of JARGON.TXT and then design four test cases to thoroughly test the working of your program code.

Evidence 2
State the test data used in Task 1.2 and show screenshots to confirm the successful testing of each of your four test cases. [4]

2 A binary search (binary chop) is a technique to search for a value in an ordered dataset.

Task 2.1
Study the identifier table and incomplete recursive algorithm.
The missing parts of the algorithm are labelled A, B and C.

Variable	Data Type	Description
`ThisArray`	`ARRAY OF STRING`	Array containing the dataset
`FindValue`	`STRING`	Item to be found
`Low`	`INTEGER`	Lowest index of the considered list
`High`	`INTEGER`	Highest index of the considered list
`Middle`	`INTEGER`	The array index for the middle position of the current list considered

```
FUNCTION BinarySearch(ThisArray, FindValue, Low, High) RETURNS INTEGER
   DECLARE Middle : INTEGER

   IF .................. A .................
      THEN
         RETURN -1 // not found
   ELSE
      // calculate new Middle value
      Middle ← .................... B ..........................
      IF ThisArray[Middle] > FindValue
         THEN
            RETURN BinarySearch(ThisArray, FindValue, Low, Middle - 1)
         ELSE
            IF ThisArray[Middle] < FindValue
               THEN
                     .................... C ....................
               ELSE
                     RETURN Middle // found at position Middle
            ENDIF
      ENDIF
   ENDIF
ENDFUNCTION
```

Evidence 3
What are the three missing lines of this pseudocode? [3]

Task 2.2
Write a program to implement the binary search.
The program will:

- Call procedure `InitialiseAnimals`
- Input an animal name
- Use the function `BinarySearch`
- Report whether or not this animal name was found. If found, also output the index position.

The array in the program has identifier `MyAnimal`.
Use the dataset given in the file `ANIMALS.TXT`. You should paste the contents of this file into your program. The statements will form the basis of the code for the procedure `InitialiseAnimals`.

Evidence 4
Program code for Task 2.2. [7]

Evidence 5
Screenshot to confirm that an animal which is present in the list was found with its index position displayed. [1]

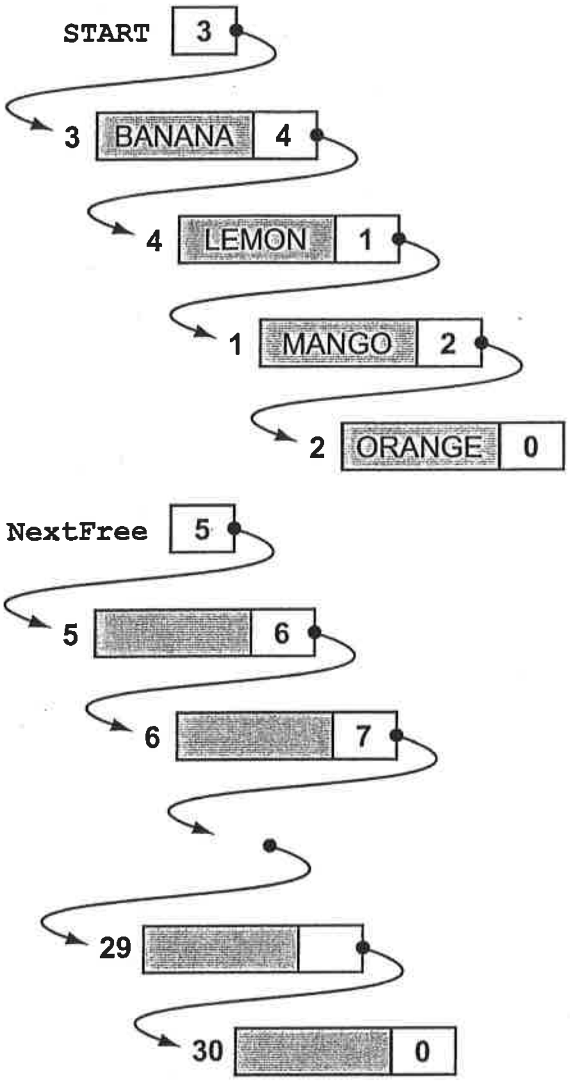
Task 2.3
Amend the program as follows:
The program must also output the number of function calls carried out.

Evidence 6
The amended program code. [4]

Evidence 7
Screenshots showing the amended output for runs of the program where:

- the animal is found
- the animal is not found. [2]

3 A program is to be written to represent and implement a linked list of nodes. Each node contains a string data value and a pointer. The pointers link the data items in alphabetical order.
The unused nodes are linked as shown below. The first unused node is the position where the next new data item is to be stored.



The diagram shows the linked list with:

- the items MANGO, ORANGE, BANANA and LEMON (added in that order).
- the unused nodes linked together.

Each node is implemented as an instance of the class `ListNode`. The class `ListNode` has the following properties:

Class: `ListNode`		
Properties		
Identifier	Data Type	Description
`DataValue`	`STRING`	The node data
`PointerValue`	`INTEGER`	The node pointer

A linked list is implemented as an instance of the class `LinkedList`. The class `LinkedList` has the following properties and methods:

Class: `LinkedList`		
Properties		
Identifier	Data Type	Description
`Node`	`ARRAY[30] OF ListNode`	The linked list data structure – data values and pointers. The array index starts at 1. For testing purposes the dataset has a maximum of 30 items.
`Start`	`INTEGER`	Index position of the node at the start of the linked list
`NextFree`	`INTEGER`	Index position of the next unused node
Methods		
Identifier		Description
`Initialise`	`PROCEDURE`	Sets all the node data values to 'empty string'. Set pointers to indicate all nodes are unused and linked. Initialise values for `Start` and `NextFree`.
`AddNode`	`PROCEDURE`	Add a new data item to the linked list.
`Traversal`	`PROCEDURE`	Display the data items in order.
`ReverseTraversal`	`PROCEDURE`	Display the data items in reverse order.
`DisplayLinkedList`	`PROCEDURE`	Display the current state of pointers and the array contents.
`IsEmpty`	`FUNCTION RETURNS BOOLEAN`	Tests for empty linked list.
`IsFull`	`FUNCTION RETURNS BOOLEAN`	Tests for no unused nodes.

Task 3.1
Write program code that repeatedly:

- displays a menu with the following choices:
  1. Add an item
  2. Traverse the linked list of used nodes and output the data values
  3. Output all pointers and data values
  5. Exit
- calls an appropriate procedure depending on the user's choice.

Evidence 8
Program code for Task 3.1. [5]

Task 3.2
Write program code for the classes `ListNode` and `LinkedList` including the `ISEmpty` method. The code should follow the specification given.
Do not attempt to write the methods `AddNode`, `Traversal`, `ReverseTraversal` or `IsFull` at this stage.

Evidence 9
Program code for the `ListNode` and `LinkedList` classes (Task 3.2). [12]

Task 3.3
Write code to create a `LinkedList` object in the main program.
Run the program and select menu choice 3 to confirm the initial values of the pointers and data values when the linked list is empty.

Evidence 10
Screenshot confirming all values after initialisation of the `LinkedList` object (Task 3.3). [3]

Task 3.4
Consider the `AddNode` method. The following algorithm will add a new data item to the linked list.
The algorithm uses the variables below:

Identifier	Data Type	Description
`NewItem`	`STRING`	New data item input by the user
`Found`	`BOOLEAN`	Flags to TRUE when the position at which to insert the new item has been found
`Current`	`INTEGER`	Current array index position during list traversal
`Previous`	`INTEGER`	Previous array index position during list traversal
`Temp`	`INTEGER`	Temporary storage of pointer value

```
PROCEDURE AddNode
    INPUT NewItem
    Node[NextFree].DataValue ← NewItem

    IF Start = 0
        THEN
            Start ← NextFree
            Temp ← Node[NextFree].PointerValue
            Node[NextFree].PointerValue ← 0
            NextFree ← Temp
        ELSE
            // traverse the list - starting at Start to find
            // the position at which to insert the new item
            Temp ← Node[NextFree].PointerValue

            IF NewItem < Node[Start].DataValue
                THEN
                    // new item will become the start of the list
                    Node[NextFree].PointerValue ← Start
                    Start ← NextFree
                    NextFree ← Temp
                ELSE
                    // the new item is not at the start of the list
                    Previous ← 0
                    Current ← Start
                    Found ← False

                    REPEAT
                        IF NewItem <= Node[Current].DataValue
                            THEN
                                Node[Previous].PointerValue ← NextFree
                                Node[NextFree].PointerValue ← Current
                                NextFree ← Temp
                                Found ← True
                            ELSE
                                // move on to the next node
                                Previous ← Current
                                Current ← Node[Current].PointerValue
                        ENDIF
                    UNTIL Found = True OR Current = 0

                    IF Current = 0
                        THEN
                            Node[Previous].PointerValue ← NextFree
                            Node[NextFree].PointerValue ← 0
                            NextFree ← Temp
                    ENDIF
            ENDIF
    ENDIF
ENDPROCEDURE
```

Note: The above pseudocode is available in the text file PSEUDOCODE_TASK_3_4.TXT but with '=' used in place of '←' shown above.

Write code to implement for the `LinkedList` class:

- the `AddNode` method
- the `IsFull` method.

You may use the text file `PSEUDOCODE_TASK_3_4.TXT` as a basis for the writing of your code.

The main program should check each time that the `LinkedList` object is not full before using the `AddNode` method.

Run the program as follows:

- Menu choice 1 four times, inputting the data values:
  MANGO, ORANGE, BANANA, LEMON in that order.
- Menu choice 3 to display.

Evidence 11
Program code for method `AddNode`. [8]

Evidence 12
Screenshot showing the pointers and the addition of the four nodes to the linked list. [3]

Task 3.5
Write program code to implement the `LinkedList` class method `Traversal` by calling the `TraversalInOrder` procedure given below.

```
PROCEDURE TraversalInOrder(Index)
   IF Index <> 0
      THEN
         OUTPUT Node[Index].DataValue
         // follow the pointer to the next data item in the linked
            list
         TraversalInOrder(Node[Index].PointerValue)
   ENDIF
ENDPROCEDURE
```

Evidence 13
Program code for procedures `Traversal` and `TraversalInOrder`. [2]

Task 3.6
Run the program as follows:

- Menu choice 1 four times, inputting the data values:
  MANGO, ORANGE, BANANA, LEMON in that order.
- Menu choice 2 to display.

Evidence 14
Screenshot showing the program execution to test the `Traversal` method. [2]

Task 3.7
Make a copy of the `TraversalInOrder` and `Traversal` procedures.
Paste to form two new procedures `TraversalInReverseOrder` and `ReverseTraversal`.

Make the necessary changes/additions to these procedures in order that the data items are output in reverse order by calling the new method `ReverseTraversal`.
Run the program code from a new menu choice 4.
Test the method using the four items given in Task 3.6.

Evidence 15
Program code for the new procedures. [2]

Evidence 16
Screenshot showing option 4 selected and the resulting output. [1]

4 Design and code a computer program to simulate the following:

A garden has a rectangular fish pond measuring 15 metres by 8 metres.

The pond is to be represented on the screen by a rectangular grid. Each square metre of the pond is represented by an x-coordinate and a y-coordinate. The top left square metre of the pond display has x = 1 and y = 1.

A boy throws a stone into the pond. The user will input the x-coordinate and y-coordinate of the stone impact position.

A grid representing the pond is then displayed with the stone's impact position:

```
X coordinate <1 to 15>? 9
Y coordinate <1 to 8>? 3

...............
...............
........S......
...............
...............
...............
...............
...............
```

Task 4.1
The following are the suggested characters to use for the visual representation of the pond:

Character	ASCII code (decimal)	Represents
.	46	One square metre of water
S	83	Stone impact position

Decide on the design to be used for:

- The data structure to represent the grid
- The contents of each square metre of the pond
- Procedure(s) and/or function(s)s to be used

Evidence 17
Show your program design (Task 4.1). [6]

Task 4.2
Write program code to display the pond contents after a single stone has been thrown.

Evidence 18
The program code. [7]

Evidence 19
Screenshot for a single run of the program. [1]

Task 4.3
The boy has been told to stop throwing stones into the pond because the pond now has three fish. The fish randomly swim around. Each fish will occupy a unique grid position.

Using a random number generator, simulate the positioning of the three fish.

Use the following character for a fish:

Character	ASCII code (decimal)	Represents
F	70	Fish

Write program code to show the pond containing the three fish at a particular instance of time. The program will now only display the pond and fish.

Evidence 20
The program code for Task 4.3. [6]

Evidence 21
Screenshot for a single run of the program. [1]

Task 4.4
The boy has been asked to feed the fish. He cannot see the fish in the pond. He throws a food pellet into the pond which lands inside one of the square metres. If one of the fish is in this square, it eats the food and becomes a happy fish.
Use character symbols for the pond's grid display as follows:

Character	ASCII code (decimal)	Represents
.	46	One square metre of water
P	80	Pellet (if not eaten by one of the fish)
H	72	Happy (fed) fish
F	70	Fish

Write program code to simulate the boy throwing one food pellet into the pond. The user will input an x-coordinate and y-coordinate for the food pellet position. You should consider the possible reuse of any code from Tasks 4.2 and 4.3.

Evidence 22
The program code. [6]

Evidence 23
Screenshot evidence similar to that shown which shows:

- one throw which did not feed a fish

```
X coordinate <1 to 15> ? 2
Y coordinate <1 to 8>? 5

...............
...............
........F......
...............
.P........F....
...............
....F..........
...............
```

- a second throw where a fish was fed

```
X coordinate <1 to 15> ? 1
Y coordinate <1 to 8>? 5

...............
...............
..............F
...............
H..............
...............
.......F.......
...............
```

[3]

BLANK PAGE