



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 2013

3 hours 15 minutes

Additional Materials:

- Pre-printed A4 Paper
- Removable storage device
- Electronic version of EMPLOYEEDATA.TXT data file
- Electronic version of WORDS1.TXT data file
- Electronic version of WORDS2.TXT data file
- Electronic version of PSEUDOCODE_TASK_3_2.TXT data file
- Electronic version of EVIDENCE.DOC data 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.

Evidence 5:

Your program code.

Task 2.2

Devise a set of four test cases with the data to be used.

Evidence 6:

This document consists of 9 printed pages and 3 blank pages.



Singapore Examinations and Assessment Board



UNIVERSITY of CAMBRIDGE
International Examinations

- 1 The files WORDS1.TXT and WORDS2.TXT store a list of single word computing terms used in a textbook.

Each entry has the following format:

<computing term>
<number>

One of the file entries (in both files) is:

program
52

This means that after a complete scan of the textbook the word 'program' was found 52 times.

Task 1.1

Write program code to find and output the term with the highest number of occurrences.
Use the file WORDS1.TXT to test your program.

Evidence 1:

The program code.

[8]

Evidence 2:

Screenshot of output.

[1]

Task 1.2

Amend your program code so that if more than one term exists with the highest number of occurrences, all terms are reported. Use the file WORDS2.TXT to test your program.

Evidence 3:

The program code.

[5]

Evidence 4:

Screenshot of output.

[1]

- 2 A company keeps data about its employees. The employee surname and employee ID are recorded.
All employee IDs are unique and have the format C999 where C is any uppercase letter and 9 is a digit.

A program is to be produced to search by either:

- The surname, which then reports the matching employee ID
- The employee ID, which then reports the matching surname

The programmer stores the data as two 1-dimensional arrays and produces the following search algorithm to search a string array and output the matching value from the second array.

```
INPUT SearchItem
FOR Index ← 1 TO UpperBound
    IF SearchItem = Array1[Index]
        THEN OUTPUT Array2[Index]
    ENDIF
ENDFOR
```

This search algorithm is inefficient.

The programmer uses the following design to produce the program code:

code to search by surname

The search algorithm has Surname as Array1 and EmployeeID as Array2
followed by code to search by employee ID

The search algorithm has EmployeeID as Array1 and Surname as Array2

This design would produce repetition of code.

Task 2.1

Write program code which performs each of the searches:

- Search by surname
- Search by employee ID

Your code should address the issues of inefficiency and repetition of code described in the scenario above.

Use the sample array data available from the text file EMPLOYEE DATA .TXT and paste this into your program code.

Evidence 5:

Your program code.

[11]

Task 2.2

Devise a set of four test cases with the data to be used.

Evidence 6:

A screenshot for each test case you considered. Annotate the screenshot explaining the purpose of each test.

[4]



- 3 The task is to store a dataset (maximum size 20 items) as a binary tree structure. You should assume that the data items are unique.

The program will use a user-defined type `Node` for each node defined as follows:

| Identifier | Data Type | Description |
|------------|-----------|--------------------------------|
| LeftP | INTEGER | The left pointer for the node |
| Data | STRING | The node's data value |
| RightP | INTEGER | The right pointer for the node |

A linked list is maintained of all the unused nodes which do not form part of the tree. The first available node which is used for a new item is indicated by `NextFreePosition`. Items in the unused list are linked using their left pointers.

The binary tree and linked list are implemented using variables as follows:

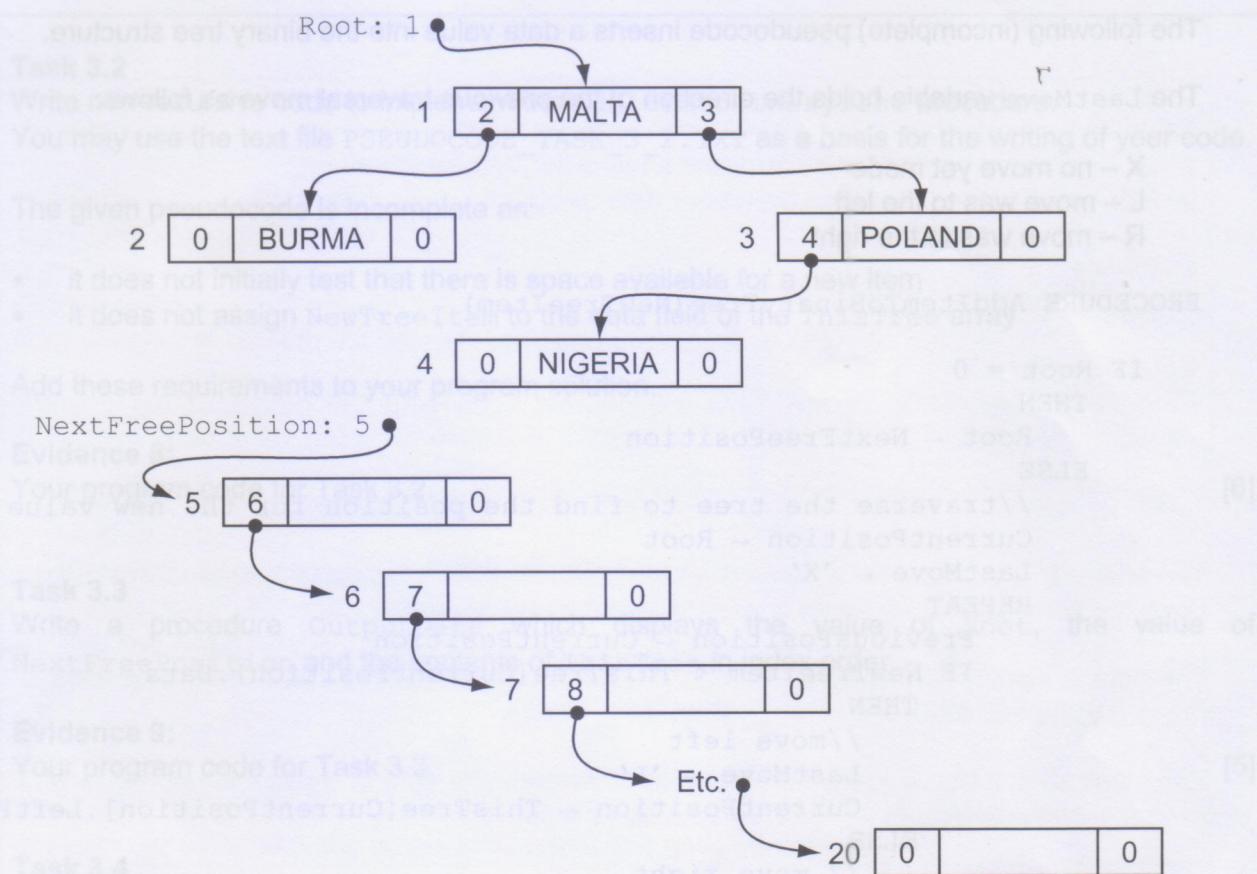
| Identifier | Data Type | Description |
|------------------|------------------|--|
| ThisTree | ARRAY[20] : Node | The tree data |
| Root | INTEGER | Index for the root position of the <code>ThisTree</code> array |
| NextFreePosition | INTEGER | Index for the next unused node |

Evidence 3:

The program code.

Evidence 4:

Screenshot of output.



The diagram shows the binary tree and linked list after four values have been added.

Task 3.1

Write the program code to declare all the required variables and create the initial linked list which contains all 20 nodes.

Add statement(s) to initialise the empty tree.

Evidence 7:

Your program code for Task 3.1.

[11]

Task 3.5

Test your program using the following data items input in the order shown:
 INDIA, NEPAL, MALAYSIA, SINGAPORE, BURMA, CANADA, SAUDI ARABIA, TURKEY, GERMANY, SWITZERLAND, JAPAN, CHINA, AUSTRALIA, VIETNAM, MONGOLIA, HUNGARY, ROMANIA, POLAND, MEXICO, TURKEY, INDIA, NEPAL, MALAYSIA, SINGAPORE, BURMA, CANADA, SAUDI ARABIA, TURKEY, GERMANY, SWITZERLAND, JAPAN, CHINA, AUSTRALIA, VIETNAM, MONGOLIA, HUNGARY, ROMANIA, POLAND, MEXICO

Evidence 11:

Provide screenshot/test evidence for Task 3.5.



- The following (incomplete) pseudocode inserts a data value into the binary tree structure.

The `LastMove` variable holds the direction of the previous traversal move as follows:

X – no move yet made

L - move was to the left

R – move was to the right

```

PROCEDURE AddItemToBinaryTree(NewTreeItem)
    IF Root = 0
        THEN
            Root ← NextFreePosition
    ELSE
        //traverse the tree to find the position for the new value
        CurrentPosition ← Root
        LastMove ← 'X'
        REPEAT
            PreviousPosition ← CurrentPosition
            IF NewTreeItem < ThisTree[CurrentPosition].Data
                THEN
                    //move left
                    LastMove ← 'L'
                    CurrentPosition ← ThisTree[CurrentPosition].LeftP
                ELSE
                    // move right
                    LastMove ← 'R'
                    CurrentPosition ← ThisTree[CurrentPosition].RightP
                ENDIF
            UNTIL CurrentPosition = 0
        ENDIF
        IF LastMove = 'R'
            THEN
                ThisTree[PreviousPosition].RightP ← NextFreePosition
        ELSE
            ThisTree[PreviousPosition].LeftP ← NextFreePosition
        ENDIF
        NextFreePosition ← ThisTree[NextFreePosition].LeftP
    ENDPROCEDURE

```

Note: The above text is available in the text file PSEUDOCODE_TASK_3_2.TXT

Task 3.2

Write non-recursive code to implement the AddItemToBinaryTree procedure.

You may use the text file PSEUDOCODE_TASK_3_2.TXT as a basis for the writing of your code.

The given pseudocode is incomplete as:

- it does not initially test that there is space available for a new item
- it does not assign NewTreeItem to the data field of the ThisTree array

Add these requirements to your program solution.

Evidence 8:

Your program code for Task 3.2.

[6]

Task 3.3

Write a procedure OutputData which displays the value of Root, the value of NextFreePosition and the contents of ThisTree in index order.

Evidence 9:

Your program code for Task 3.3.

[5]

Task 3.4

Write a main program to:

- Input new data items and add them to the binary tree by calling procedure AddItemToBinaryTree. The input is terminated with value “XXX”.
- Do not attempt to validate the input of the country names.
- Your program will then call procedure OutputData.

Run the program with the input of the single value “XXX”.

Evidence 10:

Screenshot showing the output from running the program in Task 3.4.

[3]

Task 3.5

Test your program using the following data items input in the order shown:

INDIA, NEPAL, MALAYSIA, SINGAPORE, BURMA, CANADA, LATVIA, XXX

Evidence 11:

Provide screenshot test evidence for Task 3.5.

[5]

- Dataset 1 as used in Task 3.1 which needs no scaling
- Dataset 2 of your choice to demonstrate frequencies which must be scaled
- Dataset 3 of your choice to demonstrate frequencies which must be scaled differently to Dataset 2



Further program code is required to carry out an **in-order traversal**.

Task 3.6

Write a recursive procedure to carry out an in-order tree traversal.

Include a call to the procedure from your main program.

Evidence 12:

Exercise 12: Your program code.

[8]

Evidence 13:

Evidence 15: Produce a screenshot for the Task 3.5 dataset confirming the output of the countries in alphabetical order [2]

4 The task is to input data for a frequency distribution and then output to the screen a horizontal bar chart.

The data is input as an X value followed by its frequency. Assume the frequency is always in the range 0 to 60 and there are no more than six X values.

The input shown below shows the number of sweatshirts sold in a retail shop over a one week period; for example there were 39 size XL items sold.

```
Next X value ... <ZZZ to END> XS
Frequency ... 12
Next X value ... <ZZZ to END> S
Frequency ... 22
Next X value ... <ZZZ to END> M
Frequency ... 45
Next X value ... <ZZZ to END> L
Frequency ... 56
Next X value ... <ZZZ to END> XL
Frequency ... 39
Next X value ... <ZZZ to END> XXL
Frequency ... 11
Next X value ... <ZZZ to END> ZZZ
```

Frequency distribution

Task 4.1

Write a program which inputs a set of X values and frequencies and produces output in the format shown.

Evidence 14:

Your program code for Task 4.1.

[8]

Evidence 15:

A screenshot to confirm the dataset used and the output produced.

[2]

The appearance of the bar chart display is to be improved as follows:

- Each bar is to be represented by more than one line of the same character so that its bar width is increased.
- Each bar will be shown with the same number of lines.
- The complete bar chart, including the heading, is to take up no more than 40 lines.
- The line width for the output is exactly 80 characters.
- Its appearance could be improved by changing the @ character.

Task 4.2

Write code to produce a new chart for the data used in Task 4.1 showing the maximum possible bar width and any other refinements you have introduced.

Evidence 16:

Your program code for Task 4.2.

[4]

Evidence 17:

A screenshot showing the data entry followed by the bar chart.

[2]

Some datasets will have a frequency which is greater than 60 and so the frequencies of the dataset can no longer be shown with a corresponding number of characters in the line. The frequencies will need to be scaled before the output is attempted.

The bar chart would benefit by the inclusion of a horizontal axis labelled with a scale showing the frequency values.

Task 4.3

Revise your program code to meet these new requirements.

Evidence 18:

Your program code for Task 4.3.

[8]

Evidence 19:

Screenshots demonstrating:

- Dataset 1 as used in Task 4.1 which needs no scaling
- Dataset 2 of your choice to demonstrate frequencies which must be scaled
- Dataset 3 of your choice to demonstrate frequencies which must be scaled differently to Dataset 2.

[6]

