



## COMPUTING

9597/01

Paper 1

October/November 2017

3 hours 15 minutes

Additional Materials:

- Pre-printed A4 paper
- Removable storage device
- Electronic version of INVENTORY.TXT data file
- Electronic version of ISBNPRE.TXT data file
- Electronic version of PSEUDOCODE\_TASK\_3\_2.TXT file
- Electronic version of SEARCHTREE.TXT file
- Electronic version of EVIDENCE.DOCX document

### READ THESE INSTRUCTIONS FIRST

Type in the EVIDENCE.DOCX 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.DOCX document.

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

This document consists of 12 printed pages.



Singapore Examinations and Assessment Board



CAMBRIDGE  
International Examinations

- 1 A role-playing computer game includes a list of items called the inventory. This inventory can be represented using a one-dimensional (1-D) array or a list structure.

INVENTORY.TXT is a text file containing the items from the computer game inventory. Each item type can have many occurrences. For example:

#### Inventory

Iron Ore  
Stone  
Sticky Piston  
Glass  
Stone  
Stone  
Sand  
Sticky Piston  
Iron Ore

#### ItemType

Iron Ore  
Stone  
Sticky Piston  
Glass  
Sand

### Task 1.1

Design and write program code to:

- read the entire contents of INVENTORY.TXT to an appropriate data structure called `Inventory`
- find each item type in this inventory and write these into a second similar data structure called `ItemTypes`
- count the number of each item type in the inventory and store this in a third similar data structure called `ItemCounts`
- display the contents of the `ItemTypes` and `ItemCounts` data structures using the format given below.

Example run of the program:

#### Input file:

Iron Ore  
Stone  
Sticky Piston  
Glass  
Stone  
Stone  
Sand  
Sticky Piston  
Iron Ore

The output generated from this input file would be:

ItemType	Count
Iron Ore	2
Stone	3
Sticky Piston	2
Glass	1
Sand	1

[14]

[1]



**Question 2 begins on the next page.**



- 2 Every published book has an International Standard Book Number (ISBN). This ISBN is a 9-digit number plus a check digit which is calculated and added to the end of the number. A weighted-modulus method is used to calculate the check digit.

This method uses a weighted modulus 11. If the check digit is calculated as 10, it is replaced with the character 'X'. Where the check digit is calculated as 11, it will be replaced with 0.

184146208 will be calculated as:

$$\begin{aligned} 1 \times 10 &= 10 \\ 8 \times 9 &= 72 \\ 4 \times 8 &= 32 \\ 1 \times 7 &= 7 \\ 4 \times 6 &= 24 \\ 6 \times 5 &= 30 \\ 2 \times 4 &= 8 \\ 0 \times 3 &= 0 \\ 8 \times 2 &= 16 \end{aligned}$$

$$\begin{aligned} \text{Total} &= 199 \\ 199 / 11 &= 18 \text{ remainder } 1 \\ 11 - 1 &= 10 \end{aligned}$$

Therefore, 10 is replaced with X:

ISBN is 184146208X

034085045 will be calculated as:

$$\begin{aligned} 0 \times 10 &= 0 \\ 3 \times 9 &= 27 \\ 4 \times 8 &= 32 \\ 0 \times 7 &= 0 \\ 8 \times 6 &= 48 \\ 5 \times 5 &= 25 \\ 0 \times 4 &= 0 \\ 4 \times 3 &= 12 \\ 5 \times 2 &= 10 \end{aligned}$$

$$\begin{aligned} \text{Total} &= 154 \\ 154 / 11 &= 14 \text{ remainder } 0 \\ 11 - 0 &= 11 \end{aligned}$$

Therefore, 11 is replaced with 0:

ISBN is 0340850450

075154926 will be calculated as:

$$\begin{aligned} 0 \times 10 &= 0 \\ 7 \times 9 &= 63 \\ 5 \times 8 &= 40 \\ 1 \times 7 &= 7 \\ 5 \times 6 &= 30 \\ 4 \times 5 &= 20 \\ 9 \times 4 &= 36 \\ 2 \times 3 &= 6 \\ 6 \times 2 &= 12 \end{aligned}$$

$$\begin{aligned} \text{Total} &= 214 \\ 214 / 11 &= 19 \text{ remainder } 5 \\ 11 - 5 &= 6 \end{aligned}$$

Therefore, 6 is added to the end of the ISBN:

0751549266

### Task 2.1

Study the identifier table and the incomplete recursive algorithm on the opposite page.

The missing lines in the algorithm are labelled **A**, **B** and **C**.

Write the **three** missing lines of code. Label each as **A**, **B** or **C**.

[3]

Identifier	Data type	Description
Number	STRING	The ISBN to be processed
Digit	INTEGER	A digit from the ISBN to be processed
Total	INTEGER	Running total for modulus calculation
NewNumber	STRING	A version of the list string shortened by removing the first character
CheckDigit	STRING	The calculated check digit value
CalcModulus	INTEGER	Used to store the result of (Total MOD 11)
CheckValue	INTEGER	Used to store the result of (11 - CalcModulus)



FUNCTION CalCheckDigit(Number AS STRING, Total AS INTEGER) RETURNS STRING

```

    IF LENGTH(Number) > 1 THEN
        Digit ← INTEGER(LEFT(Number,1))
        Total ← Total + (Digit * (LENGTH(Number)+1))
        NewNumber ← RIGHT(Number, LENGTH(Number)-1)
        CheckDigit ← ..... A .....
    ELSE
        Digit ← INTEGER(LEFT(Number,1))
        Total ← Total + (Digit * (LENGTH(Number)+1))
        CalcModulus ← Total MOD 11
        CheckValue ← 11 - CalcModulus
        IF CheckValue = 11 THEN
            RETURN STRING(0)
        ELSE
            IF CheckValue = 10 THEN
                ..... B .....
            ELSE
                RETURN STRING(CheckValue)
            ENDIF
        ENDIF
    ENDIF
    IF LENGTH(Number) = 9 THEN
        RETURN ..... C .....
    ELSE
        RETURN CheckDigit;
    ENDIF
END FUNCTION

```

```

// Calculate ISBN, an example of how the function is called.
// Second parameter is always 0.
ISBN = CalCheckDigit("184146208",0)

```

## Task 2.2

Write a program to implement the ISBN program using the CalCheckDigit function.

The program will:

- read the entire contents of the file ISBNPRE.TXT (seven 9-digit ISBNs without check digits) into an appropriate data structure
- use the function CalCheckDigit to calculate the result (ISBN with check digit) for each entry in the file
- write each result (ISBN with check digit) to the screen.

[11]

[1]



- 3 A data structure is required to store 25 nodes. A linked list is maintained of all the nodes. A node contains a data value and two pointers: a left pointer and a right pointer. Items in the list are initially linked using their `LeftChild` pointers.

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

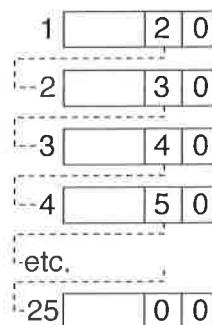
Class: <code>ConnectionNode</code>		
Attributes		
Identifier	Data Type	Description
<code>DataValue</code>	STRING	The node data
<code>LeftChild</code>	INTEGER	The left node pointer
<code>RightChild</code>	INTEGER	The right node pointer

The structure for the linked list is implemented as follows:

Identifier	Data Type	Description
<code>RobotData</code>	ARRAY[1 : 25] OF <code>ConnectionNode</code>	An array used to store the 25 nodes.
<code>Root</code>	INTEGER	Index for the root position of the <code>RobotData</code> array.
<code>NextFreeChild</code>	INTEGER	Index for the next available empty node.

The first available node is indicated by `NextFreeChild`. The initial value of `Root` is 1 and the initial value of `NextFreeChild` is 1.

The diagram shows the empty data structure with the linked list to record the unused nodes.



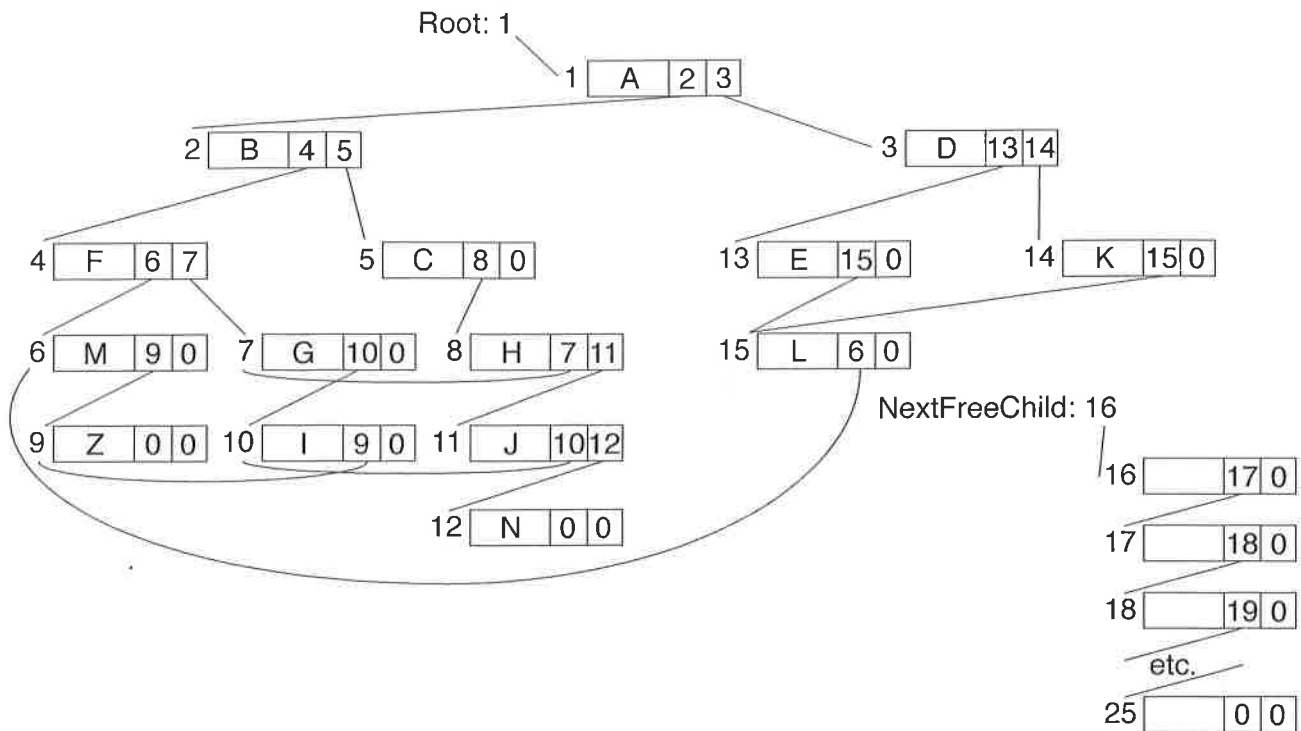
### Task 3.1

Write the program code to declare the **empty** data structure and linked list of 25 unused nodes. Add statement(s) to initialise the empty data structure.

[12]



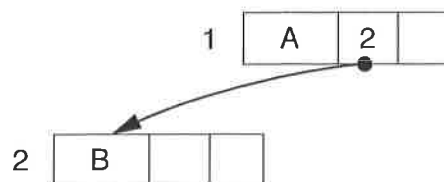
This data structure is used to record the possible routes for a robot to travel from a node A to a node Z. The following data structure illustrates many possible routes, for example,  $A \rightarrow D \rightarrow K \rightarrow L \rightarrow M \rightarrow Z$ . It is only possible to move to one of two possible nodes; for example, from node A, the only move is to node B or node D.



This data structure has 15 nodes (A to N and Z) but for future development a maximum of 25 nodes is specified. All nodes are unique.

The pseudocode on the next page can be used to add a node to the data structure. The procedure `AddToRobotData` uses the parameters `NewDataItem`, `ParentItem` and `ThisMove`.

The parameter `ThisMove` holds the move made to create this new item ('L' for LeftChild, 'R' for RightChild, 'X' for initial state/root), and the `ParentItem` parameter holds the value of the parent item which points to this `NewDataItem`.



To add node B as shown, the procedure call would be `AddToRobotData('B', 'A', 'L')`. The parameters used would be:

B, the new node

A, the parent node

L, the location of the child (which has an index of 2) is recorded in `LeftChild` of A.



The following pseudocode (available in PSEUDOCODE\_TASK\_3\_2.TXT) can be used to add a node to the data structure.

```

FUNCTION FindNode(NodeValue) RETURNS INTEGER
    Found ← FALSE
    CurrentPosition ← Root
    REPEAT
        IF RobotData[CurrentPosition].DataValue = NodeValue THEN
            Found ← TRUE
        ELSE
            CurrentPosition ← CurrentPosition + 1
        ENDIF
    UNTIL Found = TRUE OR CurrentPosition > 25
    IF CurrentPosition > 25 THEN
        RETURN 0
    ELSE
        RETURN CurrentPosition
    ENDIF
ENDFUNCTION

PROCEDURE AddToRobotData(NewDataItem, ParentItem, ThisMove)
    IF Root = 1 AND NextFreeChild = 1 THEN
        NextFreeChild ← RobotData[NextFreeChild].LeftChild
        RobotData[Root].LeftChild ← 0
        RobotData[Root].DataValue ← NewDataItem
    ELSE
        // does the parent exist?
        ParentPosition ← FindNode(ParentItem)
        IF ParentPosition > 0 THEN // parent exists
            // does the child exist?
            ExistingChild ← FindNode(NewDataItem)
            IF ExistingChild > 0 THEN // child exists
                ChildPointer ← ExistingChild
            ELSE
                ChildPointer ← NextFreeChild
                NextFreeChild ← RobotData[NextFreeChild].LeftChild
                RobotData[ChildPointer].LeftChild ← 0
                RobotData[ChildPointer].DataValue ← NewDataItem
            ENDIF
        ELSE
            IF ThisMove = 'L' THEN
                RobotData[ParentPosition].LeftChild ← ChildPointer
            ELSE
                RobotData[ParentPosition].RightChild ← ChildPointer
            ENDIF
        ENDIF
    ENDIF
ENDPROCEDURE

```





**Task 3.2**

Write code to implement `AddToRobotData` and `FindNode` from this pseudocode.  
 You may use the text file `PSEUDOCODE_TASK_3_2.TXT` as a basis for writing your code.

[7]

**Task 3.3**

Write a procedure `OutputData` which displays the value of `Root`, the value of `NextFreeChild` and the contents of `RobotData` in index order.

[6]

**Task 3.4**

The file `SEARCHTREE.TXT` contains the data for the search tree. Each row of the file contains three comma separated values, for example, the first row contains 'A', '0' and 'X'. The file is organised as:

```
NewDataItem, ParentItem, ThisMove
NewDataItem, ParentItem, ThisMove
...
<End of File>
```

There are a total of 20 lines in the `SEARCHTREE.TXT` file representing possible routes.

Write a main program to read the contents of this file and use `AddToRobotData` and `FindNode` to insert these routes into `RobotData`. Your program will then call the `OutputData` procedure.

[6]

[2]

**Task 3.5**

Write a recursive pre-order tree traversal that will display all valid routes from A to Z by following the routes described in `RobotData`.

[6]

[1]



- 4 A computer program can generate a simple Sudoku puzzle using a  $4 \times 4$  two-dimensional array.

An example of this puzzle is:

4	3	2	1
1	2	4	3
3	4	1	2
2	1	3	4

The first step to creating this puzzle is to develop a program to display the  $4 \times 4$  two-dimensional array as a grid. This program will display the grid as:

```

4 3 2 1
1 2 4 3
3 4 1 2
2 1 3 4

```

### Task 4.1

Create a program design that will declare, initialise and display the example puzzle shown. This design will:

- make use of top-down design
- include the data structure to represent the puzzle as a grid
- initialise the grid using the values shown
- make use of appropriate procedures and/or functions.

[6]

### Task 4.2

Write program code to display the puzzle designed in Task 4.1.

[5]

[1]

The puzzle is said to be valid if it follows these rules:

- It consists of four quadrants.
- The numbers in each quadrant must add up to ten.
- Each horizontal and vertical row of the puzzle must also add up to ten.
- No number can be repeated in the same row, same column or same quadrant of the puzzle.

A good strategy for creating puzzles is to start with a valid 'base' puzzle and perform transformations on it to create new puzzles.



You will write program code to create new valid puzzles.

Each puzzle created will have **two** randomly selected transformations, from a possible four, performed on it. The following are the four possible transformations that can be carried out.

Transformation	Explanation	
1	Swaps two rows in the same quadrants	<pre>       4 3 2 1       1 2 4 3       3 4 1 2       2 1 3 4       </pre>
2	Swaps two columns in the same quadrants	<pre>       4 3 2 1       1 2 4 3       3 4 1 2       2 1 3 4       </pre>
3	Swaps the top and bottom quadrant rows entirely	<pre>       4 3 2 1       1 2 4 3       3 4 1 2       2 1 3 4       </pre>
4	Swaps the left and right quadrant columns entirely	<pre>       4 3 2 1       1 2 4 3       3 4 1 2       2 1 3 4       </pre>



**Task 4.3**

Write additional program code, with brief **internal commentary** to identify each transformation.

The program code will:

- create a method of selecting, at random, two of the four possible transformations to be applied to the puzzle
- call a sub-program for each of the required transformations
- randomly select which rows will be transformed for transformations 1 and 2, for example, either the top or bottom two rows (for transformation 1) OR either the left-most or right-most two columns (for transformation 2) respectively
- display the puzzle before each transformation is applied and after the final transformation. Before each transformation, it will also display the name of the transformation being carried out. For example:

```
4321
1243
3412
2134
```

Transformation 1: Swaps two rows in the same quadrants

```
1243
4321
3412
2134
```

Transformation 4: Swaps the left and right quadrant columns entirely

```
4312
2143
1234
3421
```

[14]

[4]

