NANYANG JUNIOR COLLEGE
JC2 PRELIMINARY EXAMINATION

Higher 2

COMPUTING **9597/01**

Paper 1 **28 August 2019**

3 Hours 15 Minutes

Additional Materials: Pre-printed A4 paper
Removable storage device
Electronic version of `RAINFALL.txt` data file
Electronic version of `COUNTRIES.txt` data file
Electronic version of `BST.txt` text file
Electronic version of `INTERCEPT.txt` text 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** the 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.

1 The number of rainy days for each year and month is stored in the file `RAINFALL.txt`. The first line of the file contains the heading description for the data. Each line of data is stored in the format `YYYY-MM,99` where `YYYY` is the year, `MM` is the month and `99` is the number of days. Thus '`1982-01,10`' means there were 10 rainy days in the month of January, 1982.

You are required to write a program to:

- Read the data in the file.
- Calculate the total number of rainy days for each year by adding all the months' rainy days for that year.
- Create a new file `RAINFALLYEAR.txt`.
- Write the heading description in the first line as "Year,Rainy Days".
- For each subsequent line, write to the file in the format `YYYY,999` where `YYYY` is the year and `999` is the total number of rainy days (up to 3 digits) for that year.

Task 1.1
Write program code for this task.

Evidence 1: Your program code. [8]

Task 1.2
Write program code for a procedure `ShowMenu` to display the following menu:

```
1. Query total rainy days in any year
2. Query by year the month of highest rainy days
3. -1 to Exit
```

Evidence 2: Your program code. [2]

Task 1.3
Implement a program that displays `ShowMenu` and asks the user for their choice. Create functions `Query1` and `Query2` which corresponds to the menu selection option `1` and `2` respectively. When option `1` is selected, `Query1` should run and ask the user to input the year. `Query1` will return the total number of rainy days or a suitable message if data for that year is not available.

When option `2` is selected, `Query2` should execute and ask the user for the year. `Query2` will return the month with the highest number of rainy days in that year in words (e.g. January, August, or December) or a suitable message if data for that year is not available.

For both `Query1` and `Query2`, appropriate validation of the user input for year should be done. The program will display `ShowMenu` after each valid query until option `3` is selected.

Evidence 3: Your program code. [8]

Task 1.4
Design 3 test data which tests the functionality of your program.

Evidence 4: A screenshot for each test case you considered. Annotate the screenshot explaining the purpose of each test. [3]

2 The following is the algorithm for a recursive insertion sort on an array of size n.

1. Base Case: If array size is 1 or smaller, return.
2. Recursively sort first n – 1 elements.
3. Insert last element at its correct position in sorted array.

Task 2.1
Write program code for this algorithm to implement a recursive insertion sort function. Use the sample array data available from text file `COUNTRIES.txt` and paste this into your programming code. Your program should display the sorted items with each item shown per line.

Evidence 5: Your program code. [8]

Task 2.2
Amend your code to display the insertion process done during each recursive call. Each recursive call should display a line as follows:

`Element at position` *`n`* `is being inserted in position` *`j`*

Replace `n` and `j` to show the actual index value.

Evidence 6: Your amended program code. [3]

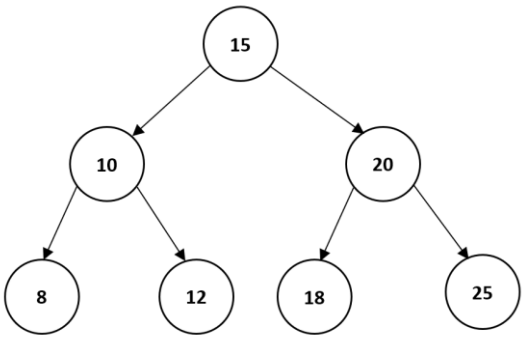
Evidence 7: One screenshot of the output. [2]

3 Create a binary tree Abstract Data Type (ADT) with commands to create a new tree, insert data items to the tree and print the tree.

The sequence of commands

Create a new tree
Add to tree (15)
Add to tree (10)
Add to tree (20)
Add to tree (8)
Add to tree (12)
Add to tree (18)
Add to tree (25)

would create the following binary tree:



The program to implement this ADT will use the classes Tree and Node designed as follows:

Tree
`root : Node`

`constructor()`
`add(newItem)`
`printTreeInOrder()`

Node

`key : INTEGER`
`left : Node`
`right : Node`

`constructor()` `insert(key : INTEGER)`

Task 3.1
Write program code to define the classes `Tree` and `Node`.

Evidence 8: Your program code. [16]

Task 3.2

- Write program code for a procedure `CreateTreefromArray` that accepts an array of unsorted unique integers passed in via a parameter.
- The procedure will read each integer in the array and construct a binary tree using your classes `Tree` and `Node`.
- Call `printTreeInOrder` to display the output (numbers shown will always be sorted).
- Test your program by copying the input data found in `BST.txt` into your code.

Evidence 9: Your `CreateTreefromArray` program code. [4]

Evidence 10: A screenshot of the output. [2]

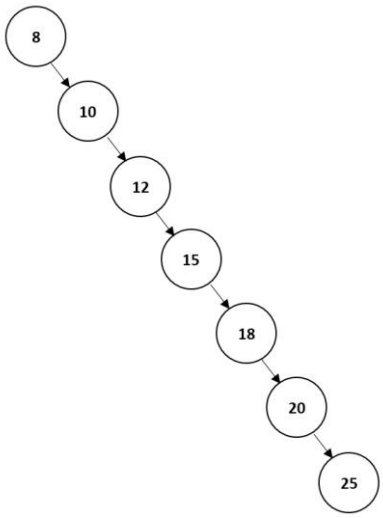
Task 3.3

A binary tree created from keys that are in ascending order will result in an unbalanced binary tree.

For instance, the sequence of commands

Create a new tree
Add to tree (8)
Add to tree (10)
Add to tree (12)
Add to tree (15)
Add to tree (18)
Add to tree (20)
Add to tree (25)

will result in a tree that looks as follows:



Amend procedure `CreateTreefromArray` so that the created tree from any input array of integers will be balanced where the number of items on the left and right subtree will roughly be divided equally (Hint: input array must first be sorted).

Evidence 11: Your amended program code. [6]

Task 3.4

Create a function `FindKthSmallest` that returns the k^{th} smallest element in your binary tree. If k = 5 the k^{th} smallest element will be 18. Your function should not need to use extra space (e.g. creating a new array) to solve the problem other than using a temp variable(s).

Evidence 12: Your program code for `FindKthSmallest`. [4]

Evidence 13: Produce a screenshot showing the retrieval of the 5^{th} smallest element from the tree created earlier. [2]

4 To encrypt a message, a keyword cipher is used. It is a form of monoalphabetic substitution where a keyword is used as the key. The key is used to determine the letter matchings of the cipher alphabet to the plain alphabet. Repeating letters in the key is removed. For instance, if the key used is `'SECRET'`, the cipher alphabet generated will be as follows:

Plain text: `A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`

Cipher Alphabet: `S E C R T A B D F G H I J K L M N O P Q U V W X Y Z`

After the key's unique letters is used up, the rest of the ciphertext letters are used in alphabetical order, excluding those already used in the key. Thus, to encode the word `'Attack'`, `'A'` is replaced with `'S'`, `'t'` is replaced with `'Q'`, and so on giving the encrypted word as `'SQQSCH'`.

Task 4.1
Write program code for a function to generate an array of the cipher alphabet given a key.

```
FUNCTION Cipher (NewAlphabet : ARRAY, Key : STRING)
```

The function has two parameters and returns the `NewAlphabet` array with the correct cipher alphabet based on the `Key` parameter.

Evidence 14: Your program code. [6]

Task 4.2
Write driver code that asks the user to enter a key that contains only letters, calls function `Cipher` and displays the cipher alphabet (all in uppercase) in one line. Do appropriate data validation on the input key.

Evidence 15: Your program code. [3]

Task 4.3
Design three suitable test cases and provide screenshot evidence for your testing.

Evidence 16: Annotated screenshots for each test data run. [3]

Task 4.4
Develop your program further to display the following menu:

1. Encode a message
2. Decode a message
3. -1 to Quit

Implement the menu options to allow a line of text to be encoded or decoded. For each option, ask for the cipher key and the message that is to be encoded or decoded. Test your program with the key `'TOPSECRET'` and the message `'I will score A for Computing'`.

Evidence 17: Your program code. [10]

Evidence 18: Screenshot(s) showing your output. [2]

Task 4.5

Frequency analysis is a common method used by code breakers to break monoalphabetic substitution ciphers. The first step is to analyse the coded message and construct a frequency table of all the letters appearing in the coded message.

Write a program that reads the content from the file `INTERCEPT.txt`. The text in this file contains a coded message. Construct a frequency table (ignore punctuation marks) as follows using a suitable data structure:

```
Ciphertext Letter: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Frequency:         5 2 ......
```

Your program will then display the table sorted by descending order showing the most used letter and its frequency first followed by the next highest and so on.

Partial sample output:

```
Ciphertext Letter:    S  O  G.....
Frequency:            88 85 67.....
```

Evidence 19: Your program code. [7]

Evidence 20: Screenshot of output. [1]