**Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Class:**\_\_\_\_\_\_\_\_\_\_\_\_\_



###### JURONG PIONEER JUNIOR COLLEGE

**JC 2 Preliminary Examination 2019**

**COMPUTING 9597/01**

**Higher 2**  **16 September 2019**

Paper 1 (Lab-based) **3 hours 15 minutes**

Additional materials: Cover Page

Electronic version of NUMBERS.txt data file

Electronic version of PATIENTS.txt data file

Electronic version of PUZZLES.txt data file

Electronic version of VISITORS.txt data 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.

The total number of marks for this paper is **100**.

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 all your work securely together.

This document consists of **10** printed pages.

**[Turn over**

|  |  |  |
| --- | --- | --- |
| **1** | The file VISITORS.txt contains the number of visitors to Singapore from 1978 to 2018. Each line of the text file is in the following format:  <year>,<month>,<number\_of\_visitors> |  |
|  | **Task 1.1**  Write program code to   * read the data from the file and store them in an appropriate data structure, * get user input to select the **year(s)** to display for **total number of visitors**, * ensure that user input for **start year** and **end year** are valid, * display the **year(s)** and **number of visitors**.   Sample output:   |  | | --- | | Start year : 1978 End year : 1980  Year Number of visitors 1978 1234567 1979 1542384 1980 1278652 | |  |
|  | **Evidence 1**  Your program code. | [11] |
|  | **Task 1.2**  Test your program with **four** relevant test data for user input. Copy the table with the following headings to show your test cases.   |  |  |  | | --- | --- | --- | | Test data | Purpose of test data | Expected results | |  |  |  | |  |
|  | **Evidence 2**  Completed table with **four** test cases.  Screenshots of running your program with the **four** test cases. | [4] |
|  |  |  |
| **2** | The incomplete pseudocode function below takes in an array as input. The array will then be sorted in ascending order and the results will be returned. Note that the start index of the array is 1. | |
|  | FUNCTION SomeSort(SomeList : ARRAY)  FOR Pointer ← 2 TO NumberOfItems  ItemToBeInserted ← SomeList[Pointer]  CurrentItem ← Pointer – 1  WHILE(SomeList[CurrentItem]>ItemToBeInserted AND CurrentItem>0)  SomeList[CurrentItem+1] ← SomeList[CurrentItem]  CurrentItem ← CurrentItem - 1  ENDWHILE  ...............  ENDFOR  RETURN ARRAY ENDFUNCTION | | |
|  | **Task 2.1**  Complete the missing line and write program code to implement the complete function. Sort the array given in the file NUMBERS.txt. You may copy and paste the array into your program code. |  |
|  | **Evidence 3**   * Your program code. * Screenshot showing the array before and after running the function. | [7] |
|  | **Task 2.2**  Write a bubble sort function that takes an array, sorts the array and returns the array. |  |
|  | **Evidence 4**  Your bubble sort function. | [4] |
|  | **Task 2.3**  Amend your program code in both functions to count and display the number of comparisons made when sorting the array. |  |
|  | **Evidence 5**   * Your amended program code. * Screenshot of running both sorting functions with number of comparisons displayed. | [4] |
| **3** | JP Medical Centre uses a priority queue to register patients who visit for medical attention. Upon registration at the medical centre, each patient will be assigned a priority number according to the urgency in seeking medical care. The urgent cases receive top priority and go directly to the front of the queue, whereas the minor cases are added to the bottom of the queue.  A priority queue is an extension of queue with the following properties.   * Every element has a priority associated with it. Higher priority has a larger number. * An element with high priority leaves the queue before an element with low priority. * If two elements have the same priority, they are served according to their order in the queue, i.e. the earlier element will be served before the later element (FIFO).   An example of operations on a priority queue is shown below:  **Initial state of priority queue**   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Data** | ‘jim’ | | ‘ben | | ‘ken’ | | ‘wayne’ | | ‘harry’ | |  | | **Priority** | 3 | | 3 | | 2 | | 1 | | 1 | |  | |  | **↑**  **front** |  | |  | |  | |  | | **↑**  **rear** |  |   **Insert ‘jenny’ with priority 3:** ‘jenny’ joins the priority queue ahead of ‘ken’ who has a priority of 2   |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Data** | ‘jim | | ‘ben | | *‘jenny’* | | ‘ken’ | | ‘wayne’ | | ‘harry’ | | | **Priority** | 3 | | 3 | | *3* | | 2 | | 1 | | 1 | | |  | **↑**  **front** |  | |  | |  | |  | |  | | **↑**  **rear** |   **Remove from priority queue:** ‘jim’ is removed from front of priority queue   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Data** | ‘ben’ | | ‘jenny’ | | ‘ken’ | | ‘wayne’ | | ‘harry’ | | | **Priority** | 3 | | 3 | | 2 | | 1 | | 1 | | |  | **↑**  **front** |  | |  | |  | |  | | **↑**  **rear** |  |   A **priority queue** abstract data type (ADT) is to be **implemented as a linked list** using object-oriented programming. Two classes Node and PQueue have been identified.   |  |  |  | | --- | --- | --- | | **Class:** Node | | | | **Identifier** | **Data Type** | **Description** | | **Properties** | | | | Data | STRING | The node data. | | Priority | INTEGER | Indicates priority of node.  Higher value has higher priority. | | Pointer | INTEGER | Pointer to next node in queue. |  |  |  |  |  | | --- | --- | --- | --- | | **Class:** PQueue | | | | | **Identifier** | | **Data Type** | **Description** | | **Properties** | | | | | ThisPQueue | ARRAY[10] OF Node | | The data for the priority queue. | | Front | INTEGER | | Index for front node of queue. | | Rear | INTEGER | | Index for rear node of queue. | | NextFree | INTEGER | | Index for the next unused node. | | **Methods** | | | | | Initialise | | PROCEDURE | * Create a new priority queue * Initialise Front and Rear to -1. | | JoinPQueue (NewItem:STRING, Priority:INTEGER) | | PROCEDURE | * Create a new node of Node class. * Assign NewItem and Priority passed as parameters to the Data and Priority attribute of Node. * Assign Node to the PQueue according to the priority of the node. | | LeavePQueue | | FUNCTION | * Remove Node from PQueue. * Return the data attribute of Node. | |  |
|  | The diagram shows the linked list with:   * the elements ‘ben’, ‘jenny’, ‘ken’, ‘wayne’, ‘harry’ in the priority queue * the unused nodes linked together   NextFree  -1  ………  -1  ‘harry’ 1  ‘ken’ 2  ‘ben’ 3  Rear  Front  ‘jenny’ 3  ‘wayne’ 1 |  |
|  | **Task 3.1**  Write program code to   * declare all the required identifiers for the Node and PQueue class as specified, including the methods Initialise, JoinPQueue, LeavePQueue, and * create the initial priority queue. |  |
|  | **Evidence 6:**  Your program code. | [25] |
|  | **Task 3.2**  Write a procedure OutputQueue which displays the value of Front, Rear, and NextFree and the contents of ThisPQueue in index order. |  |
|  | **Evidence 7:**   * Your program code for OutputPQueue. * Screenshot for displaying the initial priority queue. | [4] |
|  | **Task 3.3**  Write a main program to:   * Read from file PATIENTS.txt all the data items with its priorities into the priority queue by calling procedure JoinPQueue. * Output the priority queue by calling OutputPQueue. |  |
|  | **Evidence 8:**   * Your program code for task 3.3. * Screenshot showing the output from running program in task 3.3. | [5] |
|  | **Task 3.4**  Write additional code in your main program that prints a menu with the following options:  Patient Queue Menu   1. Add patient to PQueue 2. Remove patient from PQueue 3. Display PQueue 4. Exit program   Write program code for each option by calling the appropriate methods from the PQueue class. |  |
|  | **Evidence 9:**  Your program code for task 3.4. | [4] |
|  | **Task 3.5**  Test your main program by doing the following in order and display the priority queue:   |  |  |  |  | | --- | --- | --- | --- | | No. | Operation | Data | Priority | | 1 | Remove patient | - | - | | 2 | Add patient | Donny | 2 | |  |
|  | **Evidence 10:**  Screenshot for testing your program in task 3.5. | [2] |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **4** | You are to write a computer program to test the validity of classic Sudoku puzzles. | |
|  | The classic Sudoku puzzle involves a grid of 81 squares. The grid is divided into nine blocks, each containing nine squares. Each of the nine blocks has to contain all the numbers 1 to 9 within its squares. Each number can only appear once in a row, column or block. | |
|  | A 9 x 9 classic Sudoku puzzle can be represented using a two-dimensional array. An example of this puzzle is:  Each number 1 to 9 can only appear once in a row.  http://www.topcoder.com/i/srm/sudoku.gif  Each number 1 to 9 can only appear once in a block.  Each number 1 to 9 can only appear once in a column. | |
|  | The puzzle can be displayed in this way on the computer.   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 8 | 2 | 5 | 4 | 7 | 1 | 3 | 9 | 6 | | 1 | 9 | 4 | 3 | 2 | 6 | 5 | 7 | 8 | | 3 | 7 | 6 | 9 | 8 | 5 | 2 | 4 | 1 | | 5 | 1 | 9 | 7 | 4 | 3 | 8 | 6 | 2 | | 6 | 3 | 2 | 5 | 9 | 8 | 4 | 1 | 7 | | 4 | 8 | 7 | 6 | 1 | 2 | 9 | 3 | 5 | | 2 | 6 | 3 | 1 | 5 | 9 | 7 | 8 | 4 | | 9 | 4 | 8 | 2 | 6 | 7 | 1 | 5 | 3 | | 7 | 5 | 1 | 8 | 3 | 4 | 6 | 2 | 9 | | |
|  | **Task 4.1**  Write program code for a procedure displayboard that will take a parameter board and display a puzzle declared as a two-dimensional array.  Copy and paste array puzzle1 from the file PUZZLES.txt into your program code. Call procedure displayboard to display puzzle1. | |
|  | **Evidence 11:**   * Your program code for displayboard to display puzzle1. * Screenshot of displaying puzzle1 as a 9 x 9 Sudoku puzzle. | [5] |
|  | To test the validity of a Sudoku puzzle, each of the rows, columns and blocks can be checked to ensure that each number 1 to 9 only appears once. | |
|  | **Task 4.2**  Write program code for a function checkRow that will check all the nine rows of the puzzle to ensure that each number 1 to 9 only appears once. The function should take a parameter board and return a Boolean value. | |
|  | **Evidence 12:**  Your program code for checkRow. | [6] |
|  | **Task 4.3**  Write program code for a function checkColumn that will check all the nine columns of the puzzle to ensure that each number 1 to 9 only appears once. The function should take a parameter board and return a Boolean value. | |
|  | **Evidence 13:**  Your program code for checkColumn. | [6] |
|  | **Task 4.4**  Write program code for a function checkBlock that will check all the nine blocks of the puzzle to ensure that each number 1 to 9 only appear once. The function should take a parameter board and return a Boolean value. | |
|  | **Evidence 14:**  Your program code for checkBlock. | [8] |
|  | **Task 4.5**  Write program code to call the three functions checkrow, checkColumn, and checkBlock to test the validity of puzzle1, puzzle2, and puzzle3 given in the file PUZZLES.txt. Copy and paste these puzzles into your program code. Your program should first display the puzzle before printing statement(s) to show whether the puzzle is valid or not. If invalid, state whether the invalidity is due to the row, column or block. | |
|  | **Evidence 15:**   * Your program code for task 4.5 * Screenshot of running task 4.5 | [5] |

**END OF PAPER**