Candidate Name: _____ CT Group: _____

Index no. _____

PIONEER JUNIOR COLLEGE
JC 2 PRELIMINARY EXAMINATION

H2 COMPUTING **9597/1**

Paper 1

Tuesday **12 SEPTEMBER 2017** **3 hour 15 min**

TIME 0800 - 1115

Additional Materials: Removable storage device
Electronic version of `TEMPERATURE.txt` data file (Qn.1)
Electronic version of `COLLEGE.txt` data file (Qn 3)
Electronic version of `BITS.txt` data file (Qn 4)
Electronic version of `EVIDENCE.docx` file

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.

This question paper consists of **8** printed pages (inclusive of this page).

1 The text file `TEMPERATURE.txt` contains the mean daily temperature from January 1982 to June 2017. Each line of record is in the format

`<year>-<month> <mean daily temperature>`

For examples, `1982-01 29.8` and `2017-06 31.6`

Task 1.1

Write program code that will

- read the data from the text file,
- calculate the average of all the available monthly temperatures by year, by totalling up the temperatures for each month of that year and divided by the number of months,
- output the average temperature for that year (from 1982 to 2017), in 3 decimal places.
- Use the following format:

```
Year      Average temperature
1982            31.442
1983            31.750
⋮   ⋮             ⋮   ⋮
⋮   ⋮             ⋮   ⋮
2017            31.217
```

Evidence 1: Program code for Task 1.1. **[9]**

Evidence 2: Screenshot of running program. **[1]**

Task 1.2

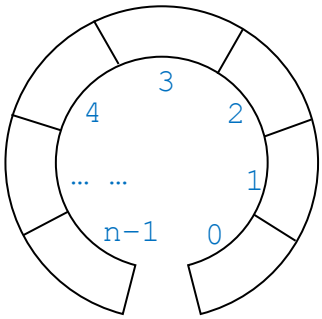
Sort by temperature and output the average temperature by year, in ascending order of temperature. If temperatures are the same, then display the later year before the earlier year. For example, if year 2017 and 1989 have the same temperature, then display 2017 before 1989.

Evidence 3: Program code for Task 1.2. **[7]**

Evidence 4: Screenshot of running program. **[1]**

[18 marks]

2 A circular queue is to be implemented with a fixed size array of n elements, indexed from `0` to `(n - 1)`.



Task 2.1

Following good programming practice, write program code for procedures

- `setup_queue` to set up circular queue which allows user to input the value of `n`,
- `enqueue` to add an element to the queue,
- `dequeue` to remove an element from the queue.

Evidence 5: Program code for Task 2.1 for `setup_queue`, `enqueue`, `dequeue`. **[12]**

Task 2.2

Write program code for a main procedure to display a menu with these options:

1. Set up queue
2. Add to queue
3. Remove from queue
4. Display queue
5. Exit

Write additional code to implement menu options 1, 2 and 3 using procedures from Task 2.1.

Also write code to implement

- option 4 to display the contents of the queue and its pointers as shown in diagram below,
- option 5 to exit program.

The diagram shows the result of option 4 to display contents of queue and pointers for `n=5` with 3 items `'fig'`, `'lemon'` and `'cane'` in the queue:

```
      Queue
   |   fig     | <- Front pointer
   |  lemon    |
   |   cane    | <- Rear pointer
   |           |
   |           |

  Number in queue: 3
```

Evidence 6: Program code for Task 2.2 for main procedure, display queue and exit program. **[8]**

Task 2.3

Test run your program from using the following input:

Test run 1
• `n = 8` • Add 5 words to queue in order: `'Mary', 'had', 'a', 'little', 'lamb'` • Display queue

Test run 2
• `n = 3` • Add 4 words to queue in order: `'The', 'quick', 'brown', 'fox'` • Remove from queue once • Display queue

Evidence 7

Screenshots of test runs 1 and 2. **[2]**

[22 marks]

3 A linked list of nodes is used to store data for a college. The data include name of student and exam mark.

The linked list Abstract Data Type (ADT) has commands to create a new linked list, add data items to the list and display the list.

The program to implement this ADT will use the classes **Node** and **LinkedList** as follows:

Node
name : STRING mark : INTEGER nextPtr : INTEGER
constructor() setName(name : STRING) setMark(mark : INTEGER) setNextPtr(ptr : INTEGER) getName() : STRING getMark() : INTEGER getNextPtr() : INTEGER

LinkedList
nodes : ARRAY OF **Node** head : INTEGER
constructor() addInOrder(name, mark) print()

In the `Node` class, `name` and `mark` store a student's name and exam mark respectively, while `nextPtr` is a pointer to the next node.

In the `LinkedList` class, `head` is a pointer to the first node in the linked list.
When the linked list has no data, `head` will be set to –1.
Data added to the linked list will be stored in alphabetical order of name.
The print method will output for each node, in array order, the data and pointer of each node.

Task 3.1

Write program code to define the classes `Node` and `LinkedList`.

Evidence 8: Program code for Task 3.1. **[20]**

Task 3.2

Write code to create a linked list object in the main program, read from data file `COLLEGE.txt` and add in all the data items, and print the array contents.

The file contains name and mark of each student in the following format:

`<name>|<mark>`

Sample record: `Jenny Tan|49`

Evidence 9

Program code for Task 3.2.

Screenshot of running Task 3.2. **[5]**

Task 3.3

Write code for a method `countNodes` to count the number of nodes used for the data in the linked list.

Evidence 10: Program code for Task 3.3 `countNodes`. **[3]**

Task 3.4

Another method `sortByMark` is to be added to the `LinkedList` class to sort the linked list in descending order of exam mark.

Write program code to implement this method.

Test your program code by sorting the linked list from Task 3.2 in descending order of mark.

Evidence 11

Program code for Task 3.4 `sortByMark`.

Screenshot of running Task 3.4 `sortByMark`. **[8]**

Task 3.5

Write another method `displayByMark` to display the list of students in descending order of mark by traversing the sorted linked list from Task 3.4.

Evidence 12

Program code for Task 3.5 `displayByMark`. **[4]**

[40 marks]

4 Perform binary addition of positive numbers.

Task 4.1

Write program code for a function that takes two binary numbers and returns the result after addition of these two binary numbers.

```
FUNCTION addBinary(bin1, bin2: STRING) RETURN result: STRING
```

Evidence 13

Program code for Task 4.1 `addBinary`. **[10]**

Task 4.2

Write a main procedure that

- repeatedly ask user to input binary numbers until user input '`XXX`' to exit program;
- perform validation of binary numbers;
- add these binary numbers using the function `addBinary`;
- output result of these addition.

Sample output:

```
Enter binary number 1 (XXX to exit) : 1
Enter binary number 2 (XXX to exit) : 10
Enter binary number 3 (XXX to exit) : XXX
Result is 11
```

Evidence 14

Program code for Task 4.2 main procedure. **[8]**

Evidence 15

Screenshot of running program with user input of the following six binary numbers in order: **`1000, 1010, 1100, 0101, 1111, 0011`**. **[2]**

These binary numbers can be found in the file `BITS.txt`.

[20 marks]

~ END OF PAPER ~