Candidate Name: _____ CT Group: _____

Index no. _____

PIONEER JUNIOR COLLEGE
JC 2 PRELIMINARY EXAMINATION

COMPUTING H2 **9597/01**

Paper 1 **17 September 2014**

3 hours 15 min

Additional Materials: Removable storage device
Electronic version of `RACE.csv` data file
Electronic version of `CITY.csv` data file
Electronic version of `phrases.txt` data file
Electronic version of `cipher.txt` data file
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 screen shots into the `EVIDENCE.docx` document.

At the end of the examination, print out your `EVIDENCE.docx` and fasten your printed copy securely together.

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

1. At the Commonwealth Games, the timings for the heats of 100m race are recorded in a file `RACE.csv`.

Each record has the following format:

```
<runnerID>,<country code>,<name of runner>,<race time>
```

A sample record is:

```
2225,SIN,Kang,10.77
```

Task 1.1

Write program code to find and output the ***number of runners*** who recorded a timing of more than 11 seconds, and ***list these runners*** on the screen along with their full records under this heading:

```
Runner ID       Country       Name       Race Time
```

Evidence 1:

Your program code for task 1.1. [6]

Evidence 2:

Screenshot of output. [1]

Task 1.2

Write program code to display the top 10 runners in order of race time. Runners with the same race time will have the same rank. The fastest runner will be displayed first, under this heading:

```
Runner ID       Country       Name       Race Time
```

Evidence 3:

Your program code for task 1.2. [7]

Evidence 4:

Screenshot of output. [1]

2. A pseudocode algorithm for a binary search on an array `CITY` is shown below. This array stores records of city, and its country and population. Array is sorted by name of city. It has an initial subscript 1 and final subscript `MAX`. The algorithm can be improved to make it clearer and more efficient.

```
Set element_found to false
Set low_element to 1
Set high_element to MAX
DOWHILE (NOT element_found)
   index = (low_element + high_element)/2
   IF CITY(index) = input_value THEN
      Set element_found to true
   ELSE
      IF input_value <= CITY(index) THEN
         high_element = index – 1
      ELSE
         low_element = index + 1
      ENDIF
   ENDIF
ENDDO
IF element_found = true THEN
   Print index
ELSE
   Print "sorry"
ENDIF
```

Task 2.1

Write program code for this algorithm and improve on clarity and efficiency. Include the sample array data available by reading from the file `CITY.csv`. If a city is found after searching, display the full record of the city, which includes country and population.

Evidence 5:

Your program code. [6]

Evidence 6:

Produce a screenshot of running your program code, by searching for **`Istanbul`** and **`Aberdeen`**. [2]

Task 2.2

Write the binary search algorithm as a recursive function. Using comment lines, explain on your choice of parameters passed into the recursive function and return value, if any.

Evidence 7:

Your program code. [7]

3. Implement a linked list to store names of runners and their best running times in seconds, in ascending order of running time. The runner with the fastest timing is stored at the first node while the runner with the slowest timing is stored at the last node. A linked list of free nodes is also implemented with a maximum size of 20 nodes.

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

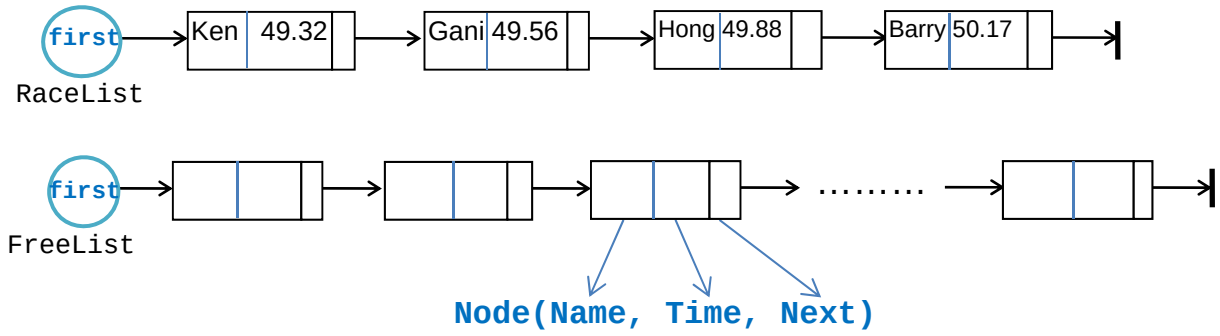
Identifier	Data Type	Description
Name	STRING	The name of the runner
Time	FLOAT	The best running time of the runner
Next	POINTER	The pointer to the next node

The program will also use another user-defined type **LinkedList** for each linked list. It contains a **first** pointer that points to the first node of the linked list and makes use of **Node** for its nodes.

The user-defined type **LinkedList** contains methods as follows:

Method	Description
Display	To display the contents of the linked list in order
AddFirst	To add a new node as first node of linked list
RemoveFirst	To remove first node of linked list
AddLast	To add a new node as last node of linked list
RemoveLast	To remove last node of linked list
Empty	To return Boolean True if linked list is empty

The diagram shows **two** linked lists – **RaceList** and **FreeList**.
RaceList contains a dataset of four nodes. Each node contains a *name*, a *running time*, and a *pointer* to the next node.
FreeList is a list of free nodes available for **RaceList** to store data, where the maximum number of nodes is 20.



Task 3.1

Write program code to create **Node** and **LinkedList**, and initialise an empty linked **RaceList**, and **FreeList** of **20** nodes. Ensure all identifiers and methods specified above are created.

Evidence 8:

Your program code for task 3.1. [17]

Evidence 9:

Screenshot of running method to display **RaceList** and **FreeList** on screen. [1]

Task 3.2

Write code to implement a method **AddInOrder** that will add a new node with data into **RaceList** in ascending order of running time. Node added to **RaceList** should be taken from **FreeList**.

Evidence 10:

Your program code for task 3.2. [11]

Task 3.3

Test your program using the following data items input in the order shown and run method to display **RaceList** and **FreeList** on screen.

Order of input	Name	Running Time
1	Barry	50.17
2	Gani	49.56
3	Hong	49.88
4	Ken	49.32

Evidence 11:

Provide screenshot for task 3.3. [2]

Task 3.4

Write code to implement a method **RemoveNode** that will remove the node that contains data specified by user to be removed from **RaceList**. Node removed from **RaceList** should be returned to **FreeList**.

Evidence 12:

Your program code for task 3.4. [8]

Task 3.5

Test your program by removing **Gani** from **RaceList** and run method to display **RaceList** and **FreeList** on screen.

Evidence 13:

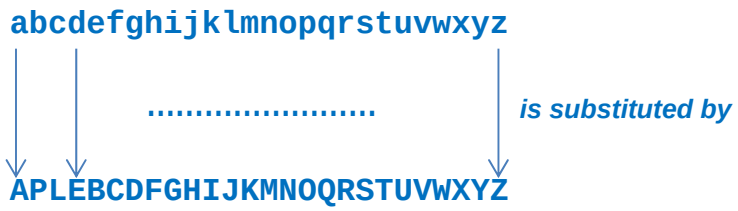
Provide screenshot for task 3.5. [1]

4. A message is encrypted and passed between two parties. To decrypt the message, a "key" is applied. Both the sending and receiving parties hold the key which enables them to encrypt and decrypt the message.

An approach of cryptography is the simple substitution cipher, a method of encryption by which each letter of a message is substituted with another letter. The receiving party deciphers the text by performing an inverse substitution.

The substitution system is created by first writing out a *phrase*. The *key* is then derived from the *phrase* by removing all the repeated letters. The *cipher text* alphabet is then constructed starting with the letters of the *key* and then followed by all the remaining letters in the alphabet.

Using this system, the phrase "**apple**" gives us the *key* as "**APLE**" and the following substitution scheme:



'a' will be substituted by 'A', 'b' will be substituted by 'P', 'c' will be substituted by 'L', 'd' will be substituted by 'E', 'e' will be substituted by 'B', and so on.

Task 4.1

Write program code for a function to create cipher text using the following specification:

```
FUNCTION CreateCipher (phrase : STRING) : STRING
```

The function `CreateCipher` has a single parameter `phrase` and returns the cipher text alphabet as a string.

Evidence 14:

Your program code for task 4.1. [8]

Task 4.2

Write program code for a procedure `CreateCipherTest` which does the following:

- read the phrases from file `phrases.txt`
- create cipher text for each of the phrases
- display each phrase and cipher text on the screen as follows:

```
Phrase: apple
Cipher text: APLEBCDFGHIJKMNOQRSTUVWXYZ
... ...
... ...
```

Evidence 15:

Your program code for task 4.2. [3]

Evidence 16:

Screenshot for running task 4.2. [1]

Task 4.3

Write program code for a function to decrypt a message using the following specification:

```
FUNCTION Decrypt (enc_message:STRING, cipher:STRING) : STRING
```

The function `Decrypt` accepts parameters `enc_message` and `cipher`, and returns the decrypted message as a string. Parameter `enc_message` is the encrypted message, and parameter `cipher` is the cipher text alphabet.

Evidence 17:

Your program code for task 4.3. [6]

Task 4.4

Write program code which does the following:

- read the phrase and encrypted message from file `cipher.txt`
- cipher text is generated from `CreateCipher` function
- message is decrypted from `Decrypt` function
- display decrypted message on the screen together with the phrase and encrypted message

```
Phrase: ...
Encrypted message: ...
Decrypted message: ...
```

Evidence 18:

Your program code for task 4.4. [3]

Evidence 19:

Screenshot for running task 4.4. [1]

Task 4.5

Write program code for a function to encrypt a message using the following specification:

```
FUNCTION Encrypt (message:STRING, cipher:STRING) : STRING
```

The function `Encrypt` accepts parameters `message` and `phrase`, and returns the encrypted message as a string. Parameter `message` is the message to be encrypted while parameter `cipher` is the cipher text.

Evidence 20:

Your program code for task 4.5. [4]

Task 4.6

Write program code which does the following:

- encrypt the message: "`do not give up!`"
- use the phrase: "`skyhigh`"
- generate cipher text from `CreateCipher` function
- message is encrypted using `Encrypt` function
- encrypted message is displayed on screen as follows:

```
Phrase: skyhigh
Encrypted Message: ...
```

Evidence 21:

Your program code for task 4.6. [3]

Evidence 22:

Screenshot for running task 4.6. [1]

END OF PAPER