Candidate Name: _____ CT Group: _____

Index no. _____

# PIONEER JUNIOR COLLEGE
## JC 1 PROMOTIONAL EXAMINATION

**COMPUTING H2** **9597/01**

**Paper 1** **21 Sep 2016**

**3 hours 15 min**

Additional Materials: Removable storage device
Electronic version of `RACE.txt` data file
Electronic version of `CITY.txt` data file
Electronic version of `COUNTRY.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.

The total mark for this paper is **85**.

Copy and paste required evidence of program listing and screen shots into the `EVIDENCE.docx` document.

**At the end of the examination, save your `EVIDENCE.docx` as a pdf file using filename `yourname_EVIDENCE.pdf` in your removable storage device.**

---

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

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

Each record has the following format:

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

A sample record is:

```
2225,SIN,C. Kang,10.77
```

**Task 1.1**

Write program code to find and display the

- **number of runners** who recorded a timing of more than 11 seconds, and
- **list of 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. **[10]**

**Evidence 2:** Screenshot of running program code for task 1.1. **[1]**

**Task 1.2**

Write program code to display the **top 10 runners** in order of race time. 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. **[8]**

**Evidence 4:** Screenshot of output of running program code for task 1.2. **[1]**

***[20 marks]***

**2.** A pseudocode algorithm for a binary search on an array CITY is shown below. Array is sorted by name of city in ascending order. It has an initial subscript 1 and final subscript MAX. The algorithm can be made clearer.

```
SET element_found to FALSE
SET low_element to 1
SET high_element to MAX
DOWHILE (NOT element_found) AND (low_element <= high_element)
   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
   OUTPUT CITY(index)
ELSE
   OUTPUT "Not found"
ENDIF
```

**Task 2.1**

Write program code for this algorithm and improve on clarity. Use the sample array data available by reading from the file CITY.txt.

**Evidence 5:** Your program code for task 2.1. **[5]**

**Task 2.2**

Amend your program code to:

- get user to input city to be searched,
- use the binary search algorithm to search for the city,
- loop through the code to ask for user input of another city,
- keep looping until user input “XXX”.

**Evidence 6:** Your program code for task 2.2. **[6]**

**Evidence 7:** Produce screenshots of running your program code, by searching for `Istanbul` and `Aberdeen`. **[2]**

**Task 2.3**

Write the binary search algorithm as a recursive function. Explain using comment lines, on your choice of parameters passed into recursive function and the return value.

**Evidence 8:** Your program code for task 2.3. **[7]**

***[20 marks]***

**3.** A program is to be written to store names of countries in alphabetical order using a linked list. The program will use nodes implemented as instances of the class **`ListNode`**. The user-defined **`ListNode`** is defined as follows:

| **Class: `ListNode`** | | |
|---|---|---|
| **Properties** | | |
| **Identifier** | **Data Type** | **Description** |
| `Name` | `STRING` | The node's value for a country name |
| `Pointer` | `INTEGER` | The pointer for the node |

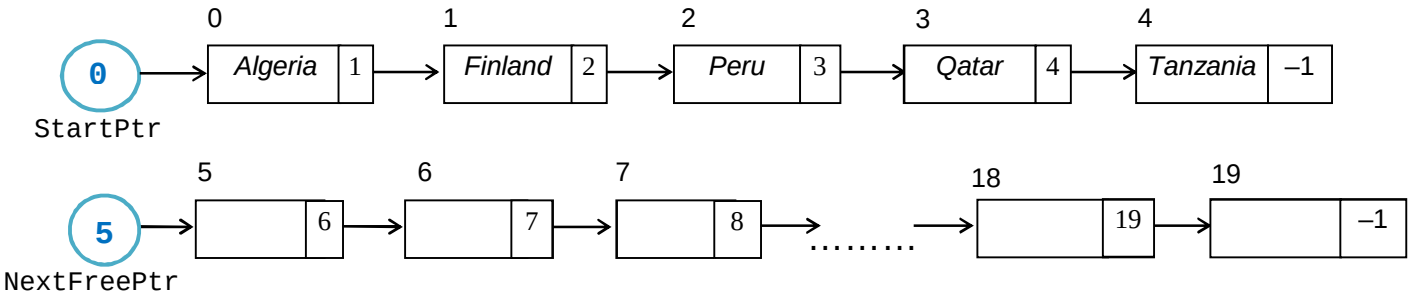
A linked list is implemented as an instance of the class `LinkedList`. The class `LinkedList` has the following properties and methods:

| **Class: `LinkedList`** | | |
|---|---|---|
| **Properties** | | |
| **Identifier** | **Data Type** | **Description** |
| `Node` | `ARRAY[20] of ListNode` | The linked list data structure – data value (`Name`) and pointers.<br>Array index starts at 0.<br>Dataset has a maximum of 20 nodes. |
| `StartPtr` | `INTEGER` | Index position of the node at the start of the linked list |
| `NextFreePtr` | `INTEGER` | Index position of the next unused node |
| **Methods** | | |
| **Identifier** | | **Description** |
| `Initialise` | `PROCEDURE` | Sets all node data values to empty string.<br>Sets pointers to indicate all nodes are unused and linked.<br>Initialise values for `StartPtr` and `NextFreePtr`. |
| `DisplayList` | `PROCEDURE` | Display the current state of pointers and array contents. |
| `InsertNode` | `PROCEDURE` | Add a new data item to the linked list. |
| `DeleteNode` | `PROCEDURE` | Remove a data item from the linked list. |
| `IsEmpty` | `FUNCTION RETURNS BOOLEAN` | Test for empty linked list. |

| | | |
|---|---|---|
| `IsFull` | `FUNCTION RETURNS BOOLEAN` | Test for no unused nodes. |
| `OutputAllNodes` | `PROCEDURE` | Output all data in alphabetical order |

The diagram shows the linked list with:

- the 5 countries added, and
- the unused nodes linked together.




**Task 3.1**

Write program code for the classes `ListNode` and `LinkedList`, including the `Initialise`, `DisplayList`, `IsEmpty` and `IsFull` method. The code should follow the specification given.

Do not write the `InsertNode`, `DeleteNode` and `OutputAllNodes` procedures yet.

**Evidence 9:** Your program code for task 3.1. **[12]**

**Task 3.2**

Write code to create a `LinkedList` object in the main program. Run `DisplayList` procedure to show the content of the array contents and state of pointers.

**Evidence 10:** Screenshot confirming all values after initialisation of the `LinkedList` object (task 3.2). **[3]**

**Task 3.3**

The following `InsertNode` algorithm will add a new data item to the linked list. The algorithm uses the variables below:

| Identifier | Data Type | Description |
|---|---|---|
| NewItem | STRING | New data item pass into the function |
| NewNodePtr | INTEGER | Array index position of the new node inserted |
| PreviousPtr | INTEGER | Previous array index position during list traversal |
| CurrentPtr | INTEGER | Current array index position during list traversal |

```
PROCEDURE InsertNode(NewItem)
    Node[NextFreePtr].Name ← NewItem
    NewNodePtr ← NextFreePtr
    NextFreePtr ← Node[NextFreePtr].Pointer
    //find insertion point
    PreviousPtr ← StartPtr
    CurrentPtr ← StartPtr
    WHILE CurrentPtr <> NULL AND Node[CurrentPtr].Name < NewItem
        PreviousPtr ← CurrentPtr
        CurrentPtr ← Node[CurrentPtr].Pointer
    ENDWHILE
    IF PreviousPtr = StartPtr
        THEN // insert new node at start of list
            Node[NewNodePtr].Pointer ← StartPtr
            StartPtr ← NewNodePtr
        ELSE //insert new node between PreviousPtr and CurrentPtr
            Node[NewNodePtr].Pointer ← Node[PreviousPtr].Pointer
            Node[PreviousPtr].Pointer ← NewNodePtr
    ENDIF
ENDPROCEDURE
```

You may use the pseudocode as a basis for the writing of your code. You should check that there is available nodes for use before inserting into the linked list.

**Evidence 11:** Program code for `InsertNode` procedure. **[8]**

**Task 3.4**

Write code to use the `InsertNode` method by reading in all the text from the file `COUNTRY.txt`.

**Evidence 12:** Program code for task 3.4. **[3]**

**Evidence 13:** Display the state of array content and pointers after running your program code in task 3.4. **[1]**

**Task 3.5**

Write code to implement the `DeleteNode` method for the `LinkedList` class that will remove a country name from the linked list. The method should:

- get a country input by the user,
- check for the country in the linked list,
- delete the country if exists in the linked list,
- output message on whether country is deleted or not in linked list.

You should check that the linked list is not empty before deleting.

**Evidence 14:** Program code for `DeleteNode` method. **[10]**

**Task 3.6**

Write program code that will use the `DeleteNode` method and delete these countries from the linked list: ***`Albania, Zambia, Singapore`***

**Evidence 15:** Screenshots for running task 3.6 and display the state of array content and pointers after running your program code in task 3.6. **[3]**

**Task 3.7**

Write program code to implement the `OutputAllNodes` method for the `LinkedList` class that will access all the data in the linked list and output them in alphabetical order.

**Evidence 16:** Program code for `OutputAllNodes` method. **[4]**

**Evidence 17:** Screenshot of running `OutputAllNodes` program code in task 3.7. **[1]**

***[45 marks]***

**~~~ END OF PAPER ~~~**