HWA CHONG INSTITUTION
C2 PRELIMINARY EXAMINATION 2016

COMPUTING
Higher 2

30 August 2016 **Paper 1 (9597 / 01)** **0815 -- 1130 hrs**

Additional Materials:
Electronic version of `FRUITS.txt` data file
Electronic version of `VEHICLE.dat` data file
Electronic version of `PSEUDOCODE_TASK_4_4.txt` 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** questions.

The maximum mark for this paper is 100.

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` and fasten your printed copy securely together.

1. A ***palindrome*** is an integer that reads the same backwards and forwards – so 6, 11 and 121 are all palindromes, while 10, 12, 223 and 2244 are not (even though 010=10, we don't consider leading zeroes when determining whether a number is a palindrome).

A ***fair and square*** number is an integer that is a *palindrome* **and** the *square of a palindrome* at the same time. For instance, 1, 9 and 121 are fair and square (being palindromes and squares, respectively, of 1, 3 and 11), while 16, 22 and 676 are **not** fair and square: 16 is not a palindrome, 22 is not a square, and while 676 is a palindrome and a square number, it is the square of 26, which is not a palindrome.

Task 1.1
Write program code with the following specification:

- Input two integers -- the endpoints of an interval e.g. `100 1000`
- Output all fair and square numbers, if any, in the interval (inclusive of endpoints).
- Output a count of the number of fair and square numbers in the interval.

Evidence 1:
Your program code for Task 1.1. [8]

Evidence 2:
Produce three screenshots showing the output of `1 4`, `10 120` and `100 1000` by the user. [3]

Task 1.2:
Write program code to output the first 10 positive fair and square numbers.

Evidence 3:
Your program code for Task 1.2. [3]

Evidence 4:
Screenshot of output. [1]

2. The data file `FRUITS.txt` contains a list of fruit names.

Task 2.1
Write program code to sort the fruit names in `FRUITS.txt` in descending order using insertion sort.

Evidence 5:
Your program code for Task 2.1. [7]

Evidence 6:
Screenshot of output. [1]

Task 2.2
Write program code to insert fruit names in `FRUITS.txt` to a binary search tree using a linked structure.

Evidence 7:
Your program code for Task 2.2. [8]

Task 2.3
Write program code to display the binary search tree contents in preorder, inorder and postorder.

Evidence 8:
Your program code. [6]

Evidence 9:
Screenshot of output. [3]

3. A transport company has a number of vehicles which can carry passengers. Each vehicle is classified either as a bus or as a coach. All vehicles have a registration number and have a certain number of seats for the passengers. A bus can have a maximum number of standing passengers, but a coach is not allowed to carry any standing passengers. Some of the coaches are fitted with seat belts, but seat belts are never fitted in a bus.

The transport company currently has a data file, `VEHICLE.dat`, stores the following data:

- `RegNo` is used to uniquely identify a particular vehicle. A typical vehicle registration number comes in the format "SBA1234":
    - S – vehicle class ("S" stands for a private vehicle)
    - BA – alphabetical series ("I" and "O" are not used to avoid confusion with "1" and "0")
    - 1234 – numerical series
- `NoOfSeats` is the maximum number of seats for passengers that each vehicle can carry.
- `VehicleType` is the type of the vehicle and can take one of the two values: '`B`' for a bus and '`C`' for a coach.

`VEHICLE.dat` has the following structure:

```
<NumberOfRecords>
<RegNo>|<NoOfSeats>|<VehicleType>
<RegNo>|<NoOfSeats>|<VehicleType>
...............
...............
<RegNo>|<NoOfSeats>|<VehicleType>
```

NumberOfRecords is the number of records in the file.

Task 3.1
Complete the test case table with the addition of **three** more invalid vehicle registration numbers. The reasons for their invalidity should be different.

The return value is a code as follows:
- 0 – valid registration number
- 1 – the registration number was not 7 characters
- you will use other integer numbers for other invalid cases.

Test Number	`RegNo`	Return value	Explanation of the test case
1	`SBA1234`	0	Valid registration number
2			
3			
4			

Evidence 10:
The completed test case table. [3]

Task 3.2
Write program code for a function to validate a registration number. The function header has the format:

```
FUNCTION ValidateRegNo(ThisRegNo : STRING) RETURNS INTEGER
```

Write a program to:

- Input a registration number by the user
- Validate the input using the function `ValidateRegNo`
- Output a message describing the validity of the input.

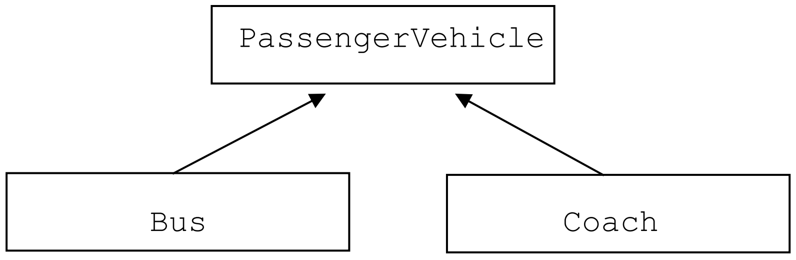
Evidence 11:

- Program code for the function `ValidateRegNo` [4]
- **Three** screenshots showing the testing of Test Numbers 2, 3 and 4.. [3]

Additional data now needs to be stored about the vehicle:

- `MaxStanding` is the maximum number of standing passengers that a vehicle classified as of type '`B`' can carry.
- `SeatBeltsFitted` is a field that indicates whether a vehicle of type '`C`' has fitted with seat belts.

The program design to process data about the vehicles is to be implemented with object-oriented programming with the following three classes:



Task 3.3
Write program code for the three classes shown.

Evidence 12:
Program code for the three classes. [6]

Task 3.4
The data in `VEHICLE.dat` does not currently contain the following additional data:

- `MaxStanding`
- `SeatBeltsFitted`

Write code to read a record from `VEHICLE.dat` and write the updated record to `UVEHICLE.dat`.

As the data on each vehicle is read it should be displayed and the user should be allowed to input the additional data required.
The user should be prompted to input the data item appropriate to the type of vehicle.

The number of standing passengers is never more than 15.
For a '`C`' type vehicle the `MaxStanding` field should contain a zero value.
The `SeatBeltsFitted` field should contain an appropriate value.

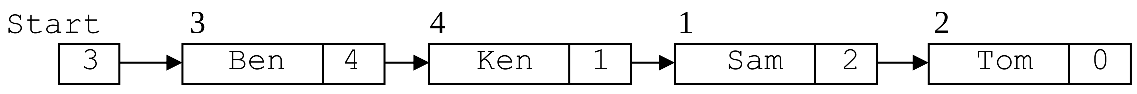
You are expected to make use of the classes you designed in Task 3.3.

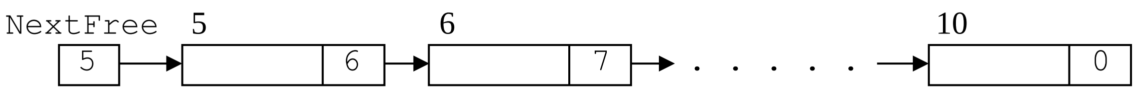
Evidence 13:
Program code for Task 3.4. [10]

Evidence 14:
Screenshot showing the contents of `UVEHICLE.dat` from running the program. [2]

4. A program is to be written to represent and implement a linked list of nodes. Each node contains a string data value and a pointer. The pointers link the data items in alphabetical order.

The unused nodes are linked as shown below. The first unused node is the position where the next new data item is to be stored.





The diagram shows the linked list with:

- the names `Sam`, `Tom`, `Ben` and `Ken` (added in that order).
- the unused nodes linked together.

The program will use a user-defined type `ListNode` for each node defined as follows:

Identifier	Data Type	Description
`DataValue`	`STRING`	The node data
`PointerValue`	`INTEGER`	The node pointer

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[10] OF ListNode`	The linked list data structure – data values and pointers. The array index starts at 1. For testing purposes, the dataset has a maximum of 10 items.
`Start`	`INTEGER`	Index position of the node at the start of the linked list.
`NextFree`	`INTEGER`	Index position of the next unused node.

Methods		
`Initialise`	`PROCEDURE`	Sets all node data values to empty string. Set pointers to indicate all nodes are unused and linked. Initialise values for `Start` and `NextFree`.
`AddNode`	`PROCEDURE`	Add a new data item to the linked list.
`RemoveNode`	`PROCEDURE`	Remove a data item from the linked list.
`Display`	`PROCEDURE`	Display the current state of pointers and the array contents.
`IsEmpty`	`BOOLEAN FUNCTION`	Test for empty linked list.
`IsFull`	`BOOLEAN FUNCTION`	Test for no unused nodes.

Task 4.1
Write program code that repeatedly:

- displays a menu with the following choices:
  1. Add an item
  2. Remove an item
  3. Output all pointers and data values
  4. Exit
- calls an appropriate procedure depending on the user's choice.

Evidence 15:
Program code for Task 4.1. [5]

Task 4.2
Write program code for the classes `ListNode` and `LinkedList`. Including the `Initialise`, `IsEmpty`, `IsFull` and `Display` methods. The code should follow the specification given.
Do not attempt to write the methods `AddNode` and `RemoveNode` at this stage.

Evidence 16:
Program code for the `ListNode` and `LinkedList` classes [10]

Task 4.3
Write code to create a `LinkedList` object in the main program.
Run the program and select menu choice 3 to confirm the initial values of the pointers and data values when the linked list is empty.

Evidence 17:
Screenshot confirming all values after initialisation of the `LinkedList` object. [2]

Consider the AddNode method. The following pseudocode adds a new data item to the linked list. The algorithm uses the variables below:

Identifier	Data Type	Description
NewItem	STRING	New data item input by the user
Found	BOOLEAN	Flags to TRUE when the position at which to insert the new item has been found
Current	INTEGER	Current array index position during list traversal
Pervious	INTEGER	Previous array index position during list traversal
Temp	INTEGER	Temporary storage of pointer value

```
PROCEDURE AddNode
      INPUT NewItem
      Node[NextFree].DataValue = NewItem

      Temp = NextFree
      NextFree = Node[NextFree].PointerValue


      # traverse the list to find the position to
      # insert the new item
      Found = False
      Current = Start
      Previous = 0

      WHILE NOT Found AND Current != 0
            IF NewItem > Node[Current].DataValue
                  THEN
                        # move on to the next node
                        Previous = Current
                        Current = Node[Current].PointerValue
                  ELSE
                        Found = True
            ENDIF
      ENDWHILE

      IF Previous = 0
            THEN
                  # new item will become the start of the list
                  Start = Temp
                  Node[Temp].PointerValue = Current
            ELSE
                  # new item is between previous and current
                  Node[Previous].PointerValue = Temp
                  Node[Temp].PointerValue = Current
      ENDIF
ENDPROCEDURE
```

Note: The above pseudocode is available in the text file PSEUDOCODE_TASK_4_4.txt

Task 4.4
Write code to implement the `AddNode` method for the `LinkedList` class.

You may use the text file `PSEUDOCODE_TASK_4_4.txt` as a basis for the writing of your code.

The main program should check each time that the `LinkedList` object is not full before using the `AddNode` method.

Run the program as follows:

- Menu choice 1 four times, inputting the data values:
  `Sam`, `Tom`, `Ben`, `Ken` in that order.
- Menu choice 3 to display.

Evidence 18:
Your program code for Task 4.4. [5]

Evidence 19:
Screenshot showing the pointers and the addition of the four nodes to the linked list. [2]

Task 4.5
Write code to implement the `RemoveNode` method for the `LinkedList` class.

The main program should check each time that the `LinkedList` object is not empty before using the `RemoveNode` method. Node removed from the linked list should be returned to the `NextFree` list.

Run the program as follows:

- Menu choice 1 four times, inputting the data values:
  `Sam`, `Tom`, `Ben`, `Ken` in that order.
- Menu choice 2 two times, inputting the data values:
  `Sam`, `Ben` in that order.
- Menu choice 3 to display.

Evidence 20:
Your program code for Task 4.5. [6]

Evidence 21:
Screenshot showing the pointers and data values after the addition of the four nodes followed by the removal of two nodes to the linked list. [2]