HWA CHONG INSTITUTION
C2 PRELIMINARY EXAMINATION 2015

COMPUTING
Higher 2

31 August 2015 **Paper 1 (9597 / 01)** **0815 -- 1130 hrs**

Additional Materials:
Electronic version of `TOP_TEAM.txt` data file
Electronic version of `CIPHER.txt` data file
Electronic version of `PHRASES.txt` data file
Electronic version of `SCORES.txt` data file
Electronic version of `PSEUDOCODE_TASK_4_2.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** 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 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.

1. A program is to process the total points of football league teams based on number of wins, draws and losses. The program can be run every day.

The program reads from file `TOP_TEAM.txt` the current highest points and team name from running the program on previous days.

The program specification is to:

- input **up to** four team names (max. 12 characters long) each with the total number match wins, draws and losses so far.
- calculate the total aggregate point for each team based on 3 points for every win, 1 point for every draw and 0 points for every loss.
- display on screen:
  - the highest point with the team name for today
  - a message saying whether or not the highest point today beat the current team with the highest aggregate points.
- update the file `TOP_TEAM.txt` if a higher aggregate point was computed today.

Task 1.1
Write program code for this task.

Evidence 1:
Your program code for Task 1.1.

[8]

Task 1.2
Draw up a set of test data which tests the functioning of your program. Consider carefully all cases which could occur for both the data input and the two processing requirements.

Evidence 2:
A screenshot for each test case you considered. Annotate the screenshot explaining the purpose of each test.

[7]

2. The Russian peasant algorithm is an alternative method to perform multiplication of whole numbers by consecutive application of doubling numbers, halving numbers and addition.

Consider the multiplication of `57` by `86` (= `4902`):

Write each number side by side:

57	86

Double the first number and halve the second number (by performing integer division by 2 i.e. drop the remainder).

If the second number or halving result is even, cross out this entire row. Keep doubling, halving, and crossing out until the halving result is 1.

~~57~~	~~86~~
114	43
228	21
~~456~~	~~10~~
912	5
~~1824~~	~~2~~
3648	1

Add up the remaining numbers in the first column. The total is the product of the original numbers.

```
  114
  228
  912
+ 3648
 ----
  4902
```

Task 2.1
Write a function to implement the Russian peasant multiplication algorithm. Test your function with the numbers `50` and 22.

Evidence 3:
Your program code for Task 2.1.

[7]

Evidence 4:
Screenshot showing the output from running the program.

[1]

The Russian peasant algorithm is actually related to binary numbers. Doubling a decimal number is to shift its binary equivalent left, while halving a decimal number is to shift its binary equivalent right. Consider the multiplication of `13 by 12 (= 156)`

In Binary		In Decimal	
~~1101~~	~~1100~~	~~13~~	~~12~~
~~11010~~	~~0110~~	~~26~~	~~6~~
110100	0011	52	3
1101000	0001	104	1

```
110100 + 1101000 = 10011100 = 156 (in decimal)
```

Task 2.2
Write a function `DecToBin()` to convert a decimal number to its binary equivalent.

Evidence 5:
Your program code for Task 2.2.

[3]

Task 2.3
Implement the Russian peasant multiplication algorithm using binary numbers. Your program should accept 2 decimal numbers as input, convert them to binary, and then perform the necessary operations to output a binary string as the result. You should make use of your `DecToBin()` function in Task 2.2.

Evidence 6:
Your program code for Task 2.3.

[8]

Evidence 7:
One screenshot showing the output from running the program code.

[1]

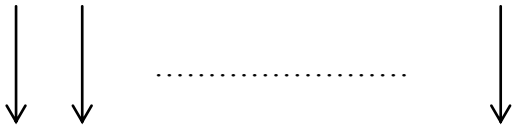
3. A message is encrypted and passed between two parties. To decrypt the message, a "key" is applied. Both the sending and receiving parties hold the key which enables them to encrypt and decrypt the message.

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:

Plain text alphabet: abcdefghijklmnopqrstuvwxyz



Cipher text alphabet: APLEBCDFGHIJKMNOQRSTUVWXYZ

'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 3.1
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 8:
Your program code for Task 3.1.

[8]

Task 3.2
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 9:
Your program code for Task 3.2.

[3]

Evidence 10:
Screenshot for running Task 3.2.

[1]

Task 3.3
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 11:
Your program code for Task 3.3.

[6]

Task 3.4
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 12:
Your program code for Task 3.4.

[3]

Evidence 13:
Screenshot for running Task 3.4.

[1]

Task 3.5
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 `cipher`, and returns the encrypted message as a string. Parameter `message` is the message to be encrypted while parameter `cipher` is the cipher text.

Evidence 14:
Your program code for Task 3.5.

[4]

Task 3.6
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 15:
Your program code for Task 3.6.

[3]

Evidence 16:
Screenshot for running Task 3.6.

[1]

4. The task is to store a dataset of students' names and test scores (max size of 20 students) as a binary tree structure. The text file `SCORES.txt` stores the students' names and test scores in the following format:

```
<student Name>|<Score>
```

All test scores are integer values in the range 0 to 100 inclusive.

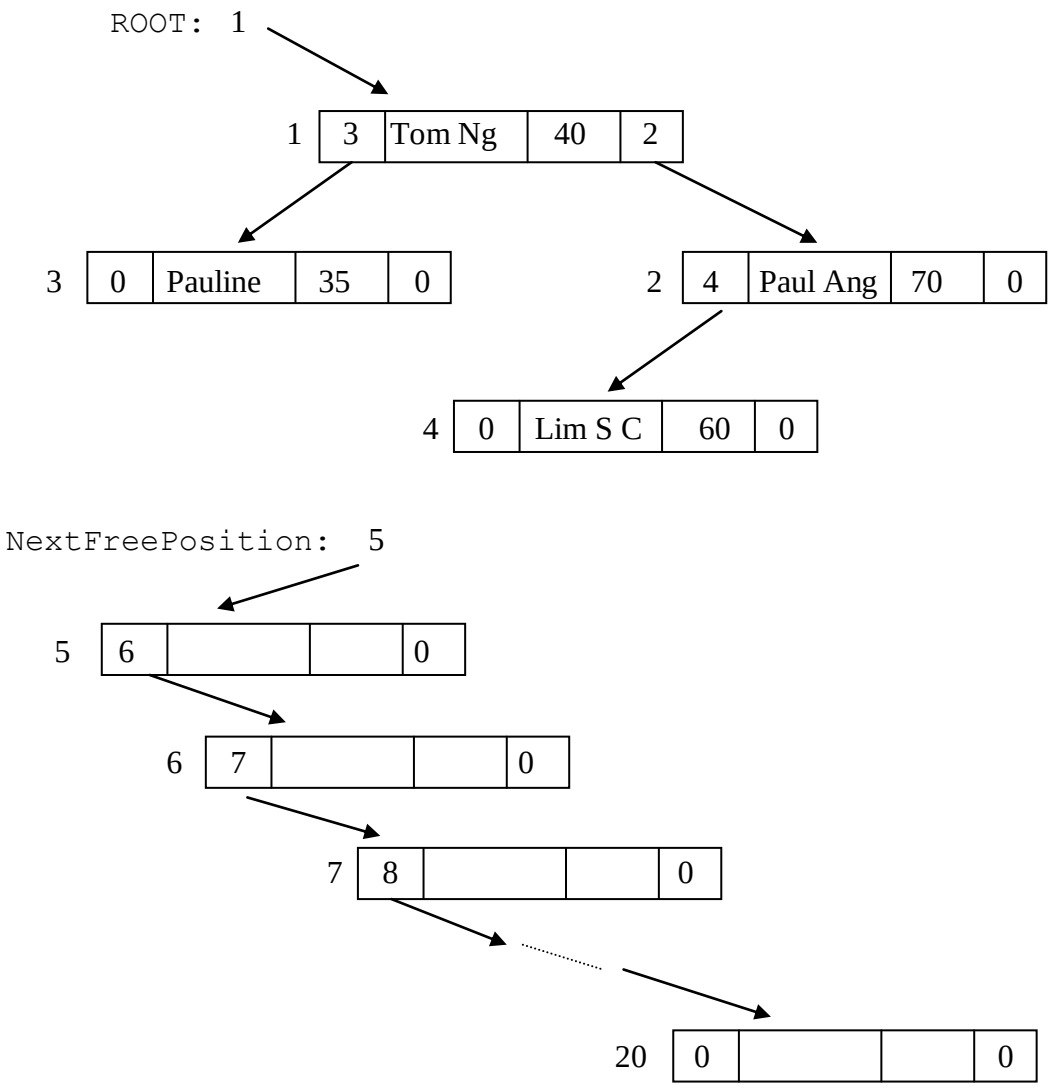
The program will use a user-defined type `Node` for each node defined as follows:

Identifier	**Data Type**	**Description**
`LeftP`	`INTEGER`	The left pointer for the node
`Name`	`STRING`	The name of the student
`Score`	`INTEGER`	The score of the student
`RightP`	`INTEGER`	The right pointer for the node

A linked list is maintained of all the unused nodes which do not form part of the tree. The first available node which is used for a new student is indicated by `NextFreePosition`. Nodes in the unused list are linked using their left pointers.

The binary tree and linked list are implemented using variables as follows:

Identifier	**Data Type**	**Description**
`ThisTree`	`ARRAY[20]: Node`	The tree data
`Root`	`INTEGER`	Index for the root position of the `ThisTree` array
`NextFreePosition`	`INTEGER`	Index for the next unused node



The diagram shows the binary tree with the students' scores 40, 70, 35 and 60 (added in that order) and linked list of unused nodes after the four students' scores have been added.

Task 4.1
Write the program code to declare all the required variables and create the initial linked list which contains all 20 nodes.

Add statement(s) to initialize the empty tree.

Evidence 17:
Your program code for Task 4.1.

[8]

The following (incomplete) pseudocode inserts a student's name and his/her score into the binary tree structure.

The `LastMove` variable holds the direction of the previous traversal move as follows:

X -- no move yet made
L -- move was to the left
R -- move was to the right

```
PROCEDURE AddNodeToBinaryTree(NewName, NewScore)

IF Root = 0
    THEN
      Root ← NextFreePosition
    ELSE
      //traverse the tree to find the position for the new value
      CurrentPosition ← Root
      LastMove ← 'X'
      REPEAT
          PreviousPosition ← CurrentPosition
          IF NewScore < ThisTree[CurrentPosition].Score
            THEN
              //move left
              LastMove ← 'L'
              CurrentPosition ← ThisTree[CurrentPosition].LeftP
            ELSE
              // move right
              LastMove ← 'R'
              CurrentPosition ← ThisTree[CurrentPosition].RightP
          ENDIF
      UNTIL CurrentPosition = 0
ENDIF

IF LastMove = 'R'
    THEN
      ThisTree[PreviousPosition].RightP ← NextFreePosition
    ELSE
      ThisTree[PreviousPosition].LeftP ← NextFreePosition
ENDIF

NextFreePosition ← ThisTree[NextFreePosition].LeftP


ENDPROCEDURE
```

Note: The above text is available in the text file `PSEUDOCODE_TASK_4_2.txt`

Task 4.2
Write non-recursive code to implement the `AddNodeToBinaryTree` procedure that will add a new node with student's name and score into the binary tree structure.

You may use the text file `PSEUDOCODE_TASK_4_2.txt` as a basis for the writing of your code.

The given pseudocode is incomplete as:
- it does not initially test that there is free node available for a new student
- it does not assign `NewName` and `NewScore` to the data fields of the `ThisTree` array

Add these requirements to your program solution.

Evidence 18:
Your program code for Task 4.2.

[6]

Task 4.3
Write a procedure `OutputData` which displays the value of `Root`, the value of `NextFreePosition` and the contents of `ThisTree` in index order.

Evidence 19:
Your program code for Task 4.3.

[5]

Task 4.4
Write a main program to:

- construct a binary search tree using the data provided in the text file `SCORES.txt` by calling procedure `AddNodeToBinaryTree`.
- Your program will then call procedure `OutputData`.

Evidence 20:
Your program code for Task 4.4.

[3]

Evidence 21:
Screenshot showing the output from running the program in Task 4.4

[5]

Task 4.5
Write a recursive procedure `RankList` to output the students' names and scores in descending scores order.

Include a call to the procedure from your main program.

Evidence 22:
Your program code for Task 4.5.

[6]

Evidence 23:
Provide a screenshot showing students' names and scores in descending scores order.

[2]

-- END OF PAPER ---