HWA CHONG INSTITUTION
C2 PRELIMINARY EXAMINATION 2017

COMPUTING
Higher 2

11 September 2017 **Paper 1 (9597 / 01)** **0815 -- 1130 hrs**

Additional Materials:
Electronic version of `WEBLOG.txt` data file
Electronic version of `CUPS-SOLD1.txt` data file
Electronic version of `CUPS-SOLD2.txt` data file
Electronic version of `TEAMS.txt` data file
Electronic version of `RESULTS.txt` data file
Electronic version of `SCORES.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 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 web log, WEBLOG.txt, keeps track of the date and time the server is being accessed, and the client that accessed it. The client is identified by either the host name or internet address. The format of WEBLOG.txt is as follows:

```
<host name>|<DD/MMM/YYYY:HH:MM:SS>
```

or

```
<internet address>|<DD/MMM/YYYY:HH:MM:SS>
```

Below is a sample of WEBLOG.txt:

```
199.72.81.55|01/Jul/1995:00:00:01
unicomp6.unicomp.net|01/Jul/1995:00:00:06
199.120.110.21|01/Jul/1995:00:00:09
burger.letters.com|01/Jul/1995:00:00:11
```

The client address (host name or internet address) can appear multiple times, depending on how many times it accessed the server. Similarly, the date it accessed the server can be duplicated too because it can access the server many times in a day.

A summary report, SUMMARY.txt, is to be generated to list the **unique** clients (hosts name/internet address) and the corresponding list of **unique** dates that it has accessed the server, from the web log, WEBLOG.txt.

Below is a sample of SUMMARY.txt:

```
199.72.81.55          01/Jul/1995,03/Jul/1995,04/Jul/1995
Buger.letters.com     01/Jul/1995,05/Jul/1995
Computing.com         01/Jul/1995,02/Jul/1995,03/Jul/1995
```

Task 1.1
Write a procedure ReadLog() to read WEBLOG.txt, using a suitable data structure(s), and prepare the log information to create SUMMARY.txt.

Evidence 1
Your ReadLog() procedure code. [4]

Task 1.2
Write a procedure ProcessLog() to generate SUMMARY.txt.

Evidence 2
Your ProcessLog() procedure code. [4]

Evidence 3
Screenshot of SUMMARY.txt after running the program. [1]

Task 1.3
Add code to your Task 1.2 program to output to screen the highest number of days the server was accessed by any client, and the corresponding client(s). Below is a sample of screen output:

```
Highest frequency (days): 3
Accessed by:
199.72.81.55
Computing.com
```

Evidence 4
Your program code for Task 1.3. [4]

Evidence 5
Screenshot of the program output. [1]

2. The files `CUPS-SOLD1.txt` and `CUPS-SOLD2.txt` contain the daily total number of cups of coffee sold at a café over a period of 50 days. The task is to read the daily sales from the file and to allow a search for a specific sales figure.

You will program two different search algorithms:

- Linear Search
- Binary Search

Task 2.1
Write program code that repeatedly:

- displays a menu with the following options:

```
1. Read file
2. Linear Search
3. Binary Search
4. End
```

- calls an appropriate procedure/function depending on the user's option.

Evidence 6
Program code for Task 2.1. [3]

Task 2.2
Write the program code for a procedure to implement menu option 1.

Evidence 7
Program code for menu option 1. [2]

Options 2 and 3 will perform a search for a specified sales figure and display a message to indicate if it is found or not.

Task 2.3
Write program code as a function to implement the linear search for menu option 2. The function will return `0` if the sales figure is found, or `-1` if it is not found.

The program will:

- Input a sales figure
- Use the function `LinearSearch`
- Report whether or not this sales figure was found.

Use `CUPS-SOLD1.txt` to test your program code.

Run the program two times. Use the following inputs: `167` and `405`.

Evidence 8

- Program code for menu option 2.
- Screenshots showing the output from menu option 2. [5]

Task 2.4

Write program code as a function to implement the binary search for menu option 3. The function will return `0` if the sales figure is found, or `-1` if it is not found.

***Hint: Do not use any pre-defined sort function e.g. sort() in your program code**.

The program will:

- Input a sales figure
- Use the function `BinarySearch`
- Report whether or not this sales figure was found.

Use `CUPS-SOLD1.txt` to test your program code.

Run the program two times. Use the following inputs: `366` and `123`.

Evidence 9

- Program code for menu option 3.
- Screenshots showing the output from menu option 3. [10]

Task 2.5

Amend your program code for the linear search function so that if the specified sales figure exists, the number of day(s) with this sales figure is also reported.

Use the file `CUPS-SOLD2.txt` to test your program.

Evidence 10

The amended program code for menu option 2. [3]

Task 2.6

Study the contents of `CUPS-SOLD2.txt` and then devise a set of **three** test cases with the sales figures to be used for testing the amended code.

Evidence 11

A screenshot for each test case you considered. Annotate the screenshot explaining the purpose of each test. [3]

3. An application is to be created to store a Football League table data. The team names are stored in the file `TEAMS.txt`. The results of the football matches are provided in file `RESULTS.txt`.

Each match data takes up one line, for example: `MadUnited 2 Chelsand 1`

That is, `MadUnited` won `Chelsand`, scoring two goals and conceding one goal, or Chelsand lost to Mad United, scoring one goal and conceding two goals.

The League table that needs to be created has the following information:

Team	P	W	D	L	GF	GA	GD	Points
MadUnited	4	3	1	0	8	3	5	10
Chelsand	4	2	1	1	10	7	3	7
Lovepool	5	2	1	2	4	6	-2	7
. . .								
. . .								

Legend:
P – games played
W – games won
D – games drawn
L – games lost
GF – goals for (scored against opponents)
GA – goals against (goals conceded by team)
GD – goal difference, i.e. GD = GF – GA
Points – computed based on 3 points per win, 1 point per draw and zero points per loss

Task 3.1
Write program code for a procedure `CreateUpdateFile` which does the following:

- the program reads the **first** match results from `RESULTS.txt`
- appends to the results of each team to a text file `NEWFILE.txt` with the following information: team name, result of match (W/D/L), goals for (GF), goals against (GA)
- for e.g. the data "`MadUnited 2 Chelsand 1`" will result in the following two records to be appended to `NEWFILE.txt`.

```
MadUnited,W,2,1
Chelsand,L,1,2
```

Evidence 12
Your `CreateUpdateFile` program code. [5]

Task 3.2
Amend your `CreateUpdateFile` program code from Task 3.1 so that all the match results are read from `RESULTS.txt`, and the `NEWFILE.txt` updated accordingly.

Evidence 13
Your program code for the amended procedure `CreateUpdateFile`. [2]

Evidence 14
2 screenshots of `NEWFILE.txt` showing first 10 and last 10 records. [2]

Task 3.3
Write program code for a function `ComputeTeamStat` which does the following:

- receives a team name as a parameter
- the function searches the file `NEWFILE.txt` for all occurrences of that team
- calculates and outputs the team's league table information,
  e.g. for team '`MadUnited`', it may output the following:

Team	P	W	D	L	GF	GA	GD	Points
MadUnited	4	3	1	0	8	3	5	10

Evidence 15
Your `ComputeTeamStat` program code. [6]

Evidence 16
A screenshot showing the output for the team `Everlong`. [1]

Task 3.4
Write program code for a procedure `GenerateTable` which does the following:

- reads the data from files `TEAMS.txt` and `NEWFILE.txt`
- and makes use of the function `ComputeTeamStat` from Task 3.3
- to output the complete league table information ordered by the team with the highest points first. If two or more teams having the same points, team having more goal difference (GD) will be listed first.

Evidence 17
Your `GenerateTable` program code. [7]

Evidence 18
A screenshot showing the output for the complete League table. [2]

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.

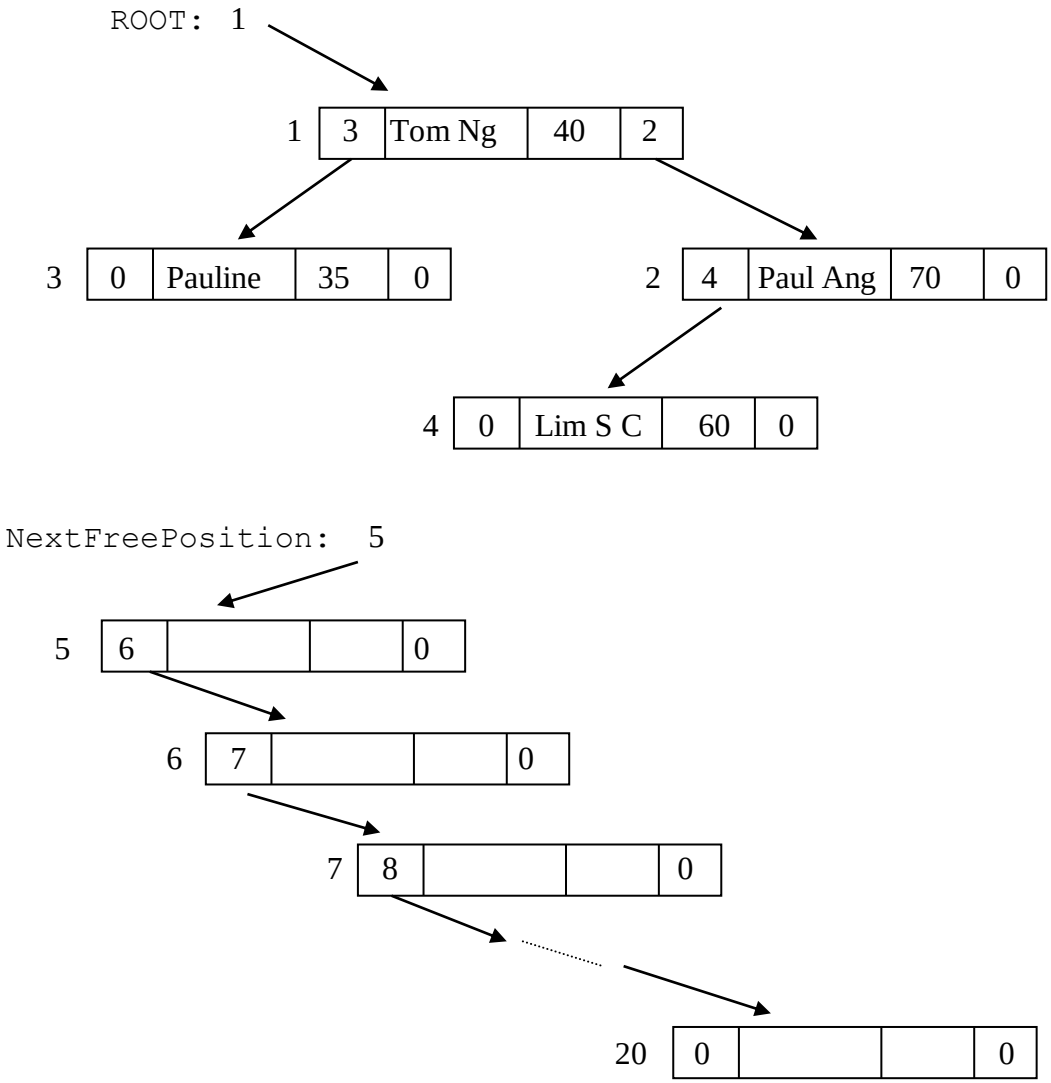
The program will use a user-defined type `Node` for each node defined as follows:

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 19
Your program code for Task 4.1. [8]

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

Evidence 20
Your program code for Task 4.2. [8]

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 21
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 `AddNodeToTree`.
- Your program will then call procedure `OutputData`.

Evidence 22
Your program code for Task 4.4. [4]

Evidence 23
Screenshot showing the output from running the program in Task 4.4. [4]

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

Include a call to the procedure from your main program.

Evidence 24
Your program code for Task 4.5. [4]

Evidence 25
Provide a screenshot showing students' names and scores in descending scores order. [2]