2011 SCHOLARSHIP EXAMINATION

PRACTICAL SECTION

The University of Waikato
Te Whare Wānanga o Waikato

DEPARTMENT	Computer Science
COURSE TITLE	Year 13 Scholarship
TIME ALLOWED	Six hours with a break for lunch at the discretion of the supervisor
NUMBER OF QUESTIONS IN PAPER	Three
NUMBER OF QUESTIONS TO BE ANSWERED	Three
GENERAL INSTRUCTIONS	Candidates are to answer ALL THREE questions. All questions are important. Answer as much of each question as you can. Plan your time to allow a good attempt at each question, but be aware that Question 3 is the most difficult and will take considerably longer than the others.
SPECIAL INSTRUCTIONS	Please hand in listings, notes and answers to written questions, and a CD/DVD with your program/computer work for each question. Please make sure that a copy of each program is printed, or stored as a plain text file. You cannot assume that the examiner has available any special software that might be required to read your files. Candidates may use any text or manual for reference during the examination.
CALCULATORS PERMITTED	Yes

1. **Ladder Tournament?** (Spreadsheet Use)

In this question you are asked to use a spreadsheet to do calculations and to display the results. We expect that the spreadsheet will be used for all calculations unless the question states otherwise - you will be marked down for performing calculations by hand and directly entering the results. Your work will be marked on three criteria.

(i) The accuracy of your results.

(ii) The skill you show in making use of the capabilities of the spreadsheet.

(iii) The presentation of your results. We have deliberately not provided any instructions concerning layout or formatting and example graphs may lack labels and proper scales.

A 'Tournament' is a pattern of games designed to rank players or teams. The simplest is a knockout tournament where players or teams are randomly paired for a first round of matches, and the winners of those matches go on to a second round, etc. This continues until one winner emerges. Knockout tournaments are suited to special sporting events.

A 'Ladder Tournament' is a system that can be used for an ongoing set of games over a long period of time. It is often used by sports clubs to rank their members. The idea is that all the players are listed in a column on a notice board. At any time a lower ranked player can challenge a higher ranked player to a match. If the lower ranked player wins, their position on the list is swapped with that of the person they defeated. Over a long period of time the list should end up with the best player at the top, the worst at the bottom and everyone else sorted appropriately in between.

For example, consider the ladder on the left below. Helga is on the top. If Aline challenges Helga and wins they swap places as shown on the right. Note that the new ranking may not be exactly right. It might be that Helga is a better player than Leeanne, but she will have to challenge Leeanne to prove it.

1. Helga Hambright	1. Aline Ascencio
2. Leeanne Lum	2. Leeanne Lum
3. Aline Ascencio	3. Helga Hambright
4. Zora Zinck	4. Zora Zinck

One problem with a simple ladder is that it shows no history. Your task is to develop a spreadsheet that maintains a Ladder Tournament. You have been provided with two files. The first (names.txt) is a list of names in order from the top of the ladder (best player) to the bottom. The names are in the order chosen at the time the ladder was first started. There are 20 names in the list. For simplicity we have just included each person's first name. The first few lines look like this:

```
Adrianna
Nichol
Mozella
```

(Question 1 – continued on next page)

The second file (matches.csv) has one line for each match played so far. Each line has the name of the defender, the name of the challenger and the outcome ("win" if the challenger won, "loss" if they lost). The items on each line are separated by commas – hence the file being a .csv or comma separated value file. The first few lines look like this.

```
Rhona,Cristopher,win
Nichol,Wilburn,win
Haywood,Reyes,win
```

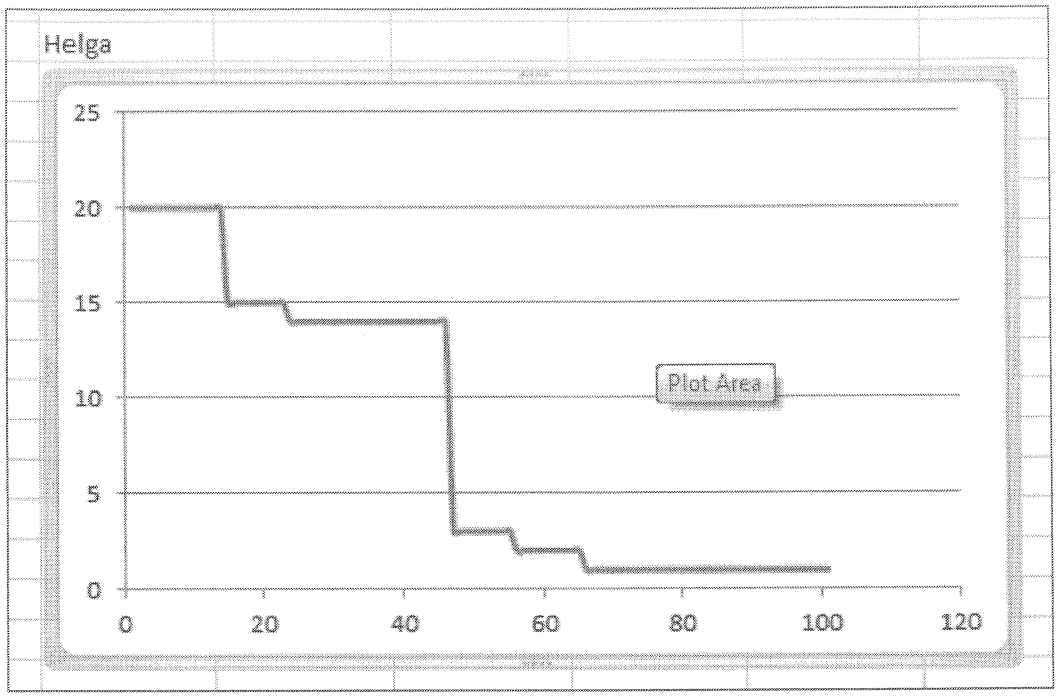
Step One: Make a spreadsheet. The sample below suggests a layout you might use. Across the top are the matches played, showing challenger, defender and result. Below are columns with the initial ladder and then, under each match, the new ladder resulting from the match (with one pair of names swapped if there has been a win).

Defender		Rhona	Nichol	Haywood	Adrianna
Challenger		Cristopher	Wilburn	Reyes	Haywood
Result		win	win	win	win
1	Adrianna	Adrianna	Adrianna	Adrianna	Haywood
2	Nichol	Nichol	Wilburn	Wilburn	Wilburn
3	Mozella	Mozella	Mozella	Mozella	Mozella
4	Mariko	Mariko	Mariko	Mariko	Mariko
5	Wilburn	Wilburn	Nichol	Nichol	Nichol
6	Edna	Edna	Edna	Edna	Edna
7	Haywood	Haywood	Haywood	Reyes	Reyes
8	Jamila	Jamila	Jamila	Jamila	Jamila
9	May	May	May	May	May
10	Lindsey	Lindsey	Lindsey	Lindsey	Lindsey
11	Aimee	Aimee	Aimee	Aimee	Aimee
12	Rhona	Cristopher	Cristopher	Cristopher	Cristopher
13	Cristopher	Rhona	Rhona	Rhona	Rhona
14	Norine	Norine	Norine	Norine	Norine
15	Thomasena	Thomasena	Thomasena	Thomasena	Thomasena
16	Reyes	Reyes	Reyes	Haywood	Adrianna
17	Zora	Zora	Zora	Zora	Zora
18	Aline	Aline	Aline	Aline	Aline
19	Leeanne	Leeanne	Leeanne	Leeanne	Leeanne
20	Helga	Helga	Helga	Helga	Helga

At the right will be the current ladder. Note that this will be a very wide spreadsheet; only the first few columns are shown

(Question 1 – continued on next page)

Step Two: Find some way of drawing a graph of place on the ladder for a particular person. The screen shot below shows a cell with the name of a player (top left), and a graph showing the player's place on the ladder over time. In this version the Y axis shows position on the ladder (1 to 20). No effort has been made to format this chart well – for example it is upside down (as Helga's position on the ladder improves, the line falls).



CONTINUED

2. **Dates and Weights** (Careful and Accurate Programming)

Your programming work in this question will be assessed on two criteria.

(a) Completeness and accuracy of the program.

(b) Good presentation. That is, it should make good use of programming language facilities, be well organised, neatly laid out, and lightly commented.

Your program should be coded using an agreed programming language. Producing a solution using a spreadsheet or similar tool is not acceptable

The weight of a human baby at birth varies over quite a wide range. A researcher is studying birth weights. Their first task is to explore the available statistics. You have been asked to write a program to help. The available data provides the birth weight in grams and the birth week (number of weeks since conception) for a number of babies.

Some babies are just naturally large or small. However, one reason for low birth weight is premature birth. Babies born early are smaller. Our researcher is only interested in babies born close to their due dates, in particular born in the 37th, 38th, or 39th week after conception.

- Your program should read the birth week and weight for a number of babies.
- It should calculate and display the average weight of those born in weeks 37, 38 and 39; ignoring those born outside that time.
- It should display the weights of all babies born in the specified weeks whose weight was less than 2600 grams.
- It should display the weights of all babies born in the specified weeks whose weight was greater than 3800 grams.

For example, given the data for 10 babies shown in the table to the right:

Birth Week	Weight(grams)
37	3791
37	3893
37	3897
40	3041
39	2625
38	3737
40	2929
39	3378
38	2149
36	2250

Your program should produce output that looks something like this:

```
Number of weights averaged = 7
Average weight = 3352
Small baby weights: 2149
Large baby weights: 3893 3897
```

3. **Connect Four** (Problem Solving and Programming)

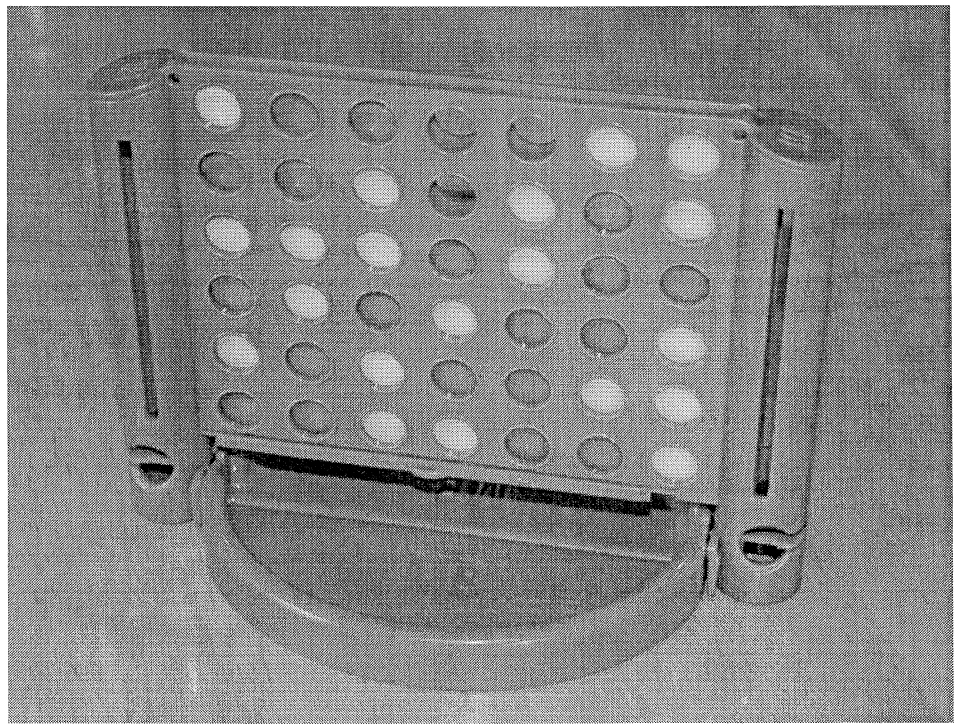
Your programming work in this question will be assessed on two criteria:

a) Your approach to the problem. We will be looking at your work for evidence that you found good ways of storing the necessary data, and devised algorithms for finding and displaying the requested results. ***Please hand in any notes and diagrams which describe what you are attempting to program, even if you don't have time to code or complete it.***

b) The extent to which your program works and correctly solves the problem.

"Connect Four (also known as Captain's Mistress, Four Up, Plot Four, Find Four, Four in a Row, and Four in a Line) is a two-player game in which the players first choose a color and then take turns dropping their colored discs from the top into a seven-column, six-row vertically-suspended grid. The pieces fall straight down, occupying the next available space within the column. The object of the game is to connect four of one's own discs of the same color next to each other vertically, horizontally, or diagonally before one's opponent can do so. There are many variations on the board size, the most commonly used being 7×6, followed by 8×7, 9×7, and 10×7.

The game was first sold under the famous Connect Four trademark by Milton Bradley in February 1974. Their game looks like this:" (Text and image from Wikipedia.)



Your task is to write a program to read and display the state of a Connect Four game in text format, to decide if the game has been won or not, and if not to decide whether the game is close to a win condition.

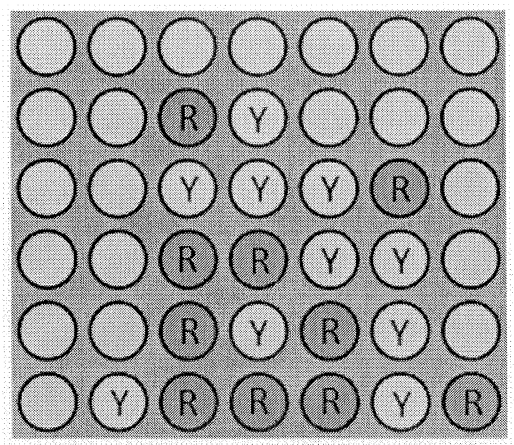
You should develop your program in stages as outlined below.

(Question 3 – continued on next page)

Stage 1: The first step is to be able to read the state of a game into memory, ready for display or analysis. You can assume that the game has 6 rows and 7 columns, that counters are yellow and red, and that red plays first. One way of handling input is to read a line with the numbers of the columns used on each play. For example the input

7 6 5 6 4 2 5 4 3 6 3 5 3 5 4 4 6 3 3 4

means that red played in column 7 (columns numbered from the left as 1 to 7), then yellow played in column 6, then red in 5, and so on. The input above should generate this game state (example from Wikipedia).



A Connect Four game in progress. With red to move, there are two winning moves that yellow can play in response: at the top and on the far right.

Stage 2: Write instructions to display the game state in a text form: For the game state example above, your display could look like this:

```
- - - - - - -
- - R Y - - -
- - Y Y Y R -
- - R R Y Y -
- - R Y R Y -
- Y R R R Y R
```

Stage 3: Write instructions to examine the game state you have read and determine whether or not the game has been won. Your program should output a message saying if red has won or yellow has won or if neither has yet won. The requirement for a win is four counters of the same colour in a line horizontally, vertically or diagonally. You may assume that only one player has a line of four in a game state.

Stage 4: Write instructions to examine a game state (where the game has not yet been won) and decide if the player to play next can prevent their opponent from winning on the following turn. For example, in the game shown above, red is next to play. Whatever red plays, it is possible for yellow to win on the following turn.

Stage 5: Can you extend your solution to stage 4 to decide if a win can be guaranteed after each player has two more turns?