THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

2013 SCHOLARSHIP EXAMINATION

PRACTICAL SECTION

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. **Group of Groups?** (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 graded 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.

Some sports competitions are easy to organize. In the case of a marathon, for example, all competitors can begin running at the same time (roughly) and the person to cross the finish line first is obviously the winner. Sports in which games are played between two players or teams are not so easy however. There are all kinds of ways of organizing a series of matches to try to ensure that the best team is finally the winner. Inevitably, the exact draw of teams, times and matches is likely to influence the outcome. The football world cup final, soon to be played in Brazil, is no exception. A sequence of matches has been carefully planned. Teams start by competing in groups of four, then winners and runners up in each group proceed to a final knockout competition. Coming up with that sequence involved much negotiation and thought. In this exercise we are going to consider an alternative tournament arrangement. The alternative involves more teams and many more matches than the real world cup, and is favoured by the people who make money from ticket sales and broadcasting rights.

One way of evaluating a tournament arrangement is to experiment with different possible results from each of the games and in each case look at the final result to see if it seems fair. The Spreadsheet is an excellent piece of software to use in such 'what if' experiments. You have been asked to put together a spreadsheet which works out results for the new 'Group of Groups' style of tournament. The idea is to have the spreadsheet display the draw (which teams play each other) for the tournament; allow a person to enter match results; and present the outcomes. It should be easy to update the data and look at the consequences of different match outcomes.

For our spreadsheet we will assume that there are 50 teams taking part in our 'Group of Groups' tournament. The teams will be divided into five groups of ten. The competition will proceed in two parts. In the first part, each team will play all the other teams in their group. (Each team will play 9 matches. In total there will be 45 matches played in each group). In the second part, the winner and runner-up in each of the five groups will be put together to form a group of ten finalists. That group will play in the same pattern as the groups in the first part (again each finalist team will play 9 matches, for a total of 45). The winner is the team that wins the most games. We will assume there are no ties.

It should be possible for an experimenter to enter teams into groups in any order they like, but for testing the software we will start with the teams in the order listed below. The teams taking part are:

Spain, Argentina, Germany, Italy, Colombia, Belgium, Uruguay, Brazil, Netherlands, Croatia, Portugal, Greece, USA, Switzerland, Russia, Chile, England, Bosnia, Ivory Coast, Ecuador, Mexico, Sweden, Denmark, Ghana, France, Ukraine, Montenegro, Algeria, Slovenia, Hungary, Romania, Czech Republic, Costa Rica, Peru, Panama, Venezuela, Nigeria, Mali, Norway, Honduras, Paraguay, Japan, Serbia, Cape Verde, Albania, Tunisia, Austria, Iran, Turkey, and Egypt.

So you can assume group one will be Spain to Croatia, group two will be Portugal to Ecuador, etc.

(Question 1 – continued on next page)

It may not be possible to have the spreadsheet perform all calculations automatically. Your task is to perform as much as possible using spreadsheet calculations. You should not use any macros, or built-in programming language – just use spreadsheet operations and the built-in functions. Where the user must carry out some operations by hand – possibly copying parts of the spreadsheet from one area to another, possibly doing sorting operations – please provide instructions.

Step One

For each group of teams in part one, make a table, showing the matches played by each team, and providing places in which results can be entered.

Step Two

For each group display the sorted results, with the winner first and others listed below in order of descending results. It does not matter in what order you display tied teams (teams with the same score).

Step Three

Transfer the winning teams from each group to form a final group. As in step one, there should be a table into which part two match results can be placed.

Step Four

Display the final sorted results for the part two group.

Step Five

Write instructions for the user of your spreadsheet. This should explain how to enter team names, how to read the draw and enter match results, and any operations that need to be performed by hand. You can write this on paper, or use a text editor, or a word processor.

2. **Cash Flow** (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.

Keeping track of family finances can be difficult. Even when your income is sufficient to pay all your bills, you still have to be careful to have money available at the right times. For example, mortgage payments are usually made once a month, whereas wages may be paid every two weeks. This means that you cannot rely on wages being paid into your bank account just before each mortgage payment is due. If there is a month in which your wages are paid the week after your mortgage is due, you must make sure that you have kept enough money from previous wage amounts to cover the mortgage payment.

Your task is to write a computer program to check that a bank account always has enough money to cover mortgage payments. The program will be run on January the first each year to check cash flow for the year. The account has been set up especially for mortgage payments and a part of the owner's wages are transferred into the account every pay day. The program should start by asking for three pieces of information:

The starting bank account balance.
The amount of the mortgage payment per month
The amount of wage money transferred into the account every two weeks

It should then keep track of money in the account throughout the year.

Mortgage payments are made on the last day of each month. If the program keeps track of time by counting days, and day 1 is the 1st of January, then as January has 31 days, February has 28, March has 31, etc; mortgage payments will be made on day 31, day 59, day 90, etc. You may assume that this is not a leap year, so February always has 28 days.

Wages are paid every 14 days starting on day 14; so payments occur on days 14, 28, 42, etc. Payments are made before withdrawals, so you can assume that money that is paid on a particular day is available for making a mortgage payment on that same day.

Your program should display the remaining balance at each mortgage payment, and stop when a payment cannot be made.

An interaction with your program might look like this (underlined text is entered by the user of the program:

```
Starting balance  >> 155
Mortgage payment  >> 1000
Wages transferred >> 462

Mortgage paid on day 31 leaving $79 in the account
Mortgage paid on day 59 leaving $3 in the account
Mortgage could not be paid on day 90
```

3. **Shotcube** (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.***

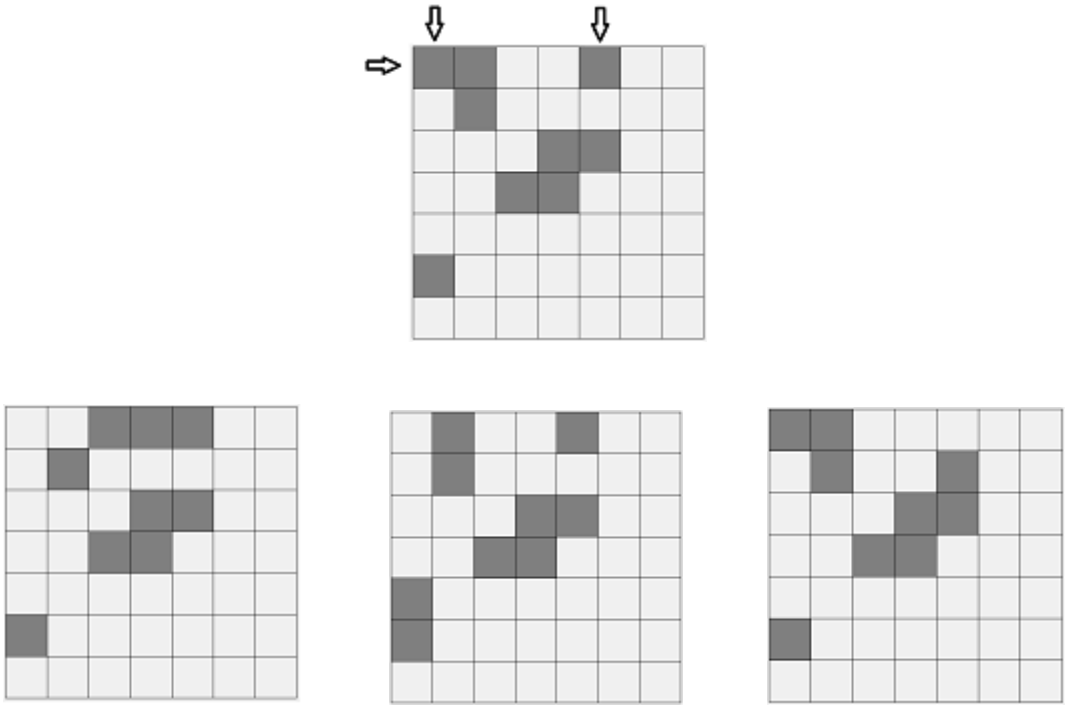
(b) The extent to which your program works and correctly solves the problem.

You may find that the programming language you use makes it difficult to work as shown in the example implementations below. If this is the case, feel free to build your program in a way that suits your way of working.

Namco Bandai's video game 'Tales of Graces' introduces a puzzle minigame known as Shotcube. In this game you have 9 cubes placed on a 7 by 7 square grid. The objective is to rearrange the cubes so that they all lie in a 3-by-3 square. In the video game, this square must be in the exact centre of the grid, but for this problem, the 3-by-3 square may be anywhere on the grid.

The only way to rearrange the cubes is to "shoot" at them from the outside of the grid. If there is a cube on an edge (or corner) of the grid, you may shoot it to push it in a straight line, in which it will keep moving until it hits another cube, at which point it stops. You may only shoot a cube if there is another cube in its path which stops it. If you shoot a cube in a direction such that one or more cubes are immediately behind it, all of those cubes will move in unison until the farthest cube hits another cube (after having travelled at least one grid square), at which point all of them stop.

Consider the grid below. The arrows mark the three legal shots; no other shots are legal, either because there is no cube on the edge of the grid, or because there is no cube present to stop the motion of the shot cube(s). The three grids below that indicate the result of shooting the first row to the right, the first column down, and the fifth column down.



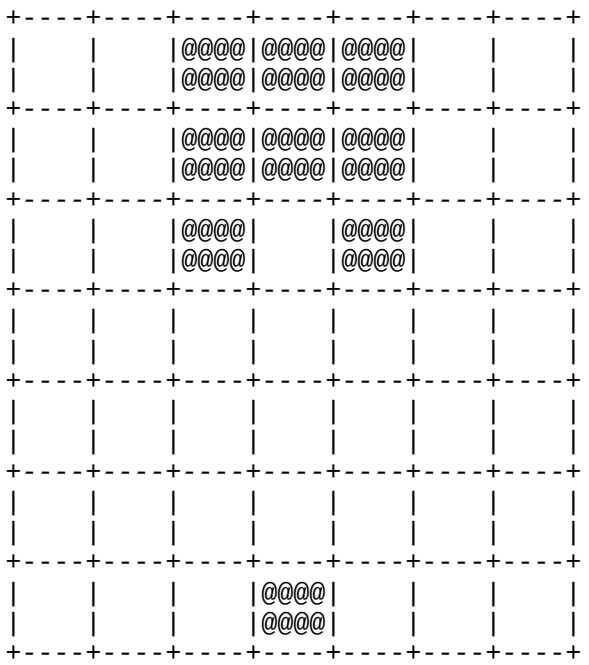
(Question 3 – continued on next page)

Your task is to write a program to supervise a game of shotcube. It should allow a player to enter a game layout and then make moves. If the player tries to make an illegal move, then the program should state that the move is illegal. If the game reaches a point at which no further moves are legal, it should announce that the player has lost. If the game reaches a point at which the nine cubes form a square (anywhere on the grid), then the program should announce that the player has won.

Stage 1: The first step is to be able to read game layout into your program. You are free to do this however you like, but your program must have a way to enter new games. A way of doing this at the command line is to have an interaction something like this (underlined text entered by the program user):

```
Enter locations for the 9 cubes
as row(1-9) followed by column(1-9)
  Cube 1:  1 3
  Cube 2:  1 4
  Cube 3:  1 5
  Cube 4:  2 3
  Cube 5:  2 4
  Cube 6:  2 5
  Cube 7:  3 3
  Cube 8:  3 5
  Cube 9:  7 4
```

Stage 2: Write instructions to display the game grid. Using the console, it is possible to produce a clear display. Here is an example. The exact format is up to you.



Stage 3: Allow the user to enter a move. You can choose how a move should be entered.

Stage 4: Perform the move and display the updated grid.

Stage 5: Check for the game having been won or lost.