Electronics and
Computer Science

UNIVERSITY OF Southampton

Referral

Module:	COMP1206: Programming 2
Assignment:	Referral
Lecturer:	Dr Jian Shi (Jian.Shi@soton.ac.uk)
Deadline:	4pm on 27th Aug 2021

Instructions

The Referral consists of three separate components:

	Component	**Details**
1)	Programming in Java	Hitori Application (60%)
2)	Understanding of Testing	Testing and report (20%)
3)	Programming in C	Programming task (20%)

These components are detailed on the following pages.

IMPORTANT: If you have any questions about the referral, please contact the ECS student office at ecs-studentoffice@soton.ac.uk, but for questions about the assessment instructions, please contact Dr Jian Shi at Jian.Shi@soton.ac.uk.

Submission

For each part, you should create:

1. Hitori.zip for the Hitori Application (part 1).
2. Testing.zip for the report and the unit testing code (part 2).
3. C.zip for the C programming task (part 3).

When you do the submission, please zip all these parts into one zip file and submit to the module Handin site.

Referral Part 1: Hitori Application

Hitori is a puzzle game where the player is presented with a square grid of cells. Each cell initially contains a number. The aim of the game is to eliminate cells (by blacking them out) until the following three constraints all hold:

1) There are no duplicate numbers in each row and column.
2) There are no blacked out cells next to each other (horizontally or vertically).
3) All white cells are connected to each other in a single component.

Below is an example of a Hitori puzzle (Figure 1) and its solution (Figure 2):

4	8	1	6	3	2	5	7
3	6	7	2	1	6	5	4
2	3	4	8	2	8	6	1
4	1	6	5	7	7	3	5
7	2	3	1	8	5	1	2
3	5	6	7	3	1	8	4
6	4	2	3	5	4	7	8
8	7	1	4	2	3	5	6

Figure 1: Hitori Puzzle

■	8	■	6	3	2	■	7
3	6	7	2	1	■	5	4
■	3	4	■	2	8	6	1
4	1	■	5	7	■	3	■
7	■	3	■	8	5	1	2
■	5	6	7	■	1	8	■
6	■	2	3	5	4	7	8
8	7	1	4	■	3	■	6

Figure 2: Solution of Puzzle

In this assessment, your objective is to design a JavaFX application for playing Hitori.

The application must meet the following **basic** requirements:

1) **Initial Puzzle**: On starting the application, an initial puzzle must be displayed to the user (for example the grid in Figure 1).
2) **Elimination**: The user must be able to eliminate (i.e., black out) cells by left-clicking on them.
3) **Reactivation**: The user must be able to reverse the elimination action by right-clicking on a blacked-out cell. This cell should then become white again.
4) **Reset**: There must be a button to reset the puzzle to the starting state (where no cells are blacked out). When clicking the button, a confirmation dialog must be shown to the user first, allowing the user to confirm or cancel this action.
5) **Mistake Detection**: When the user has violated constraints 2 or 3 through their actions, this should be highlighted immediately and clearly to the user.
6) **Win Detection**: When the puzzle has been solved (i.e., constraints 1-3 hold), the user should be notified immediately and clearly.

The application should also meet the following **extended** requirements:

7) **Resizability**: The application should be resizable, with the grid resizing appropriately to fill the available space.
8) **Loading**: There should be functionality to load new puzzles from a file. Files should be assumed to contain a text representation (in UTF-8) of the puzzle, where each line in the file corresponds to a line in the puzzle and adjacent cells are separated by spaces. For example, the puzzle in Figure 1 would be represented as:

```
4 8 1 6 3 2 5 7
3 6 7 2 1 6 5 4
4 8 1 6 3 2 5 7
3 6 7 2 1 6 5 4
2 3 4 8 2 8 6 1
4 1 6 5 7 7 3 5
7 2 3 1 8 5 1 2
3 5 6 7 3 1 8 4
6 4 2 3 5 4 7 8
8 7 1 4 2 3 5 6
```

You can download this file at:
https://secure.ecs.soton.ac.uk/notes/comp1206/fe/example.txt

Mark Scheme

6 marks are awarded for each of Requirements 1-7. Up to 18 marks are awarded for Requirement 8. The final mark is out of 60.

Submission

Please make sure you submit only the .java files required to run the application, compressed as a single .zip file called Hitori.zip. You should not rely on third-party packages (except for JavaFX). Please include a readme.txt file with brief instructions on how to compile and run the application, as well as any additional information that may help us run and test your application.

Referral Part 2: Testing and Report

For this part, you should do some testing for Part 1 and then write a report about that. In detail:

1. Use JUnit to conduct unit tests for requirements 5 and 6 in Part 1. Save your testing code in a separate .java file. Please note: if you failed to finish requirements 5 and 6, unit tests without the working code would also be accepted. **(weight 10)**

2. Write a short report (1-2 pages) to describe all the testing you have done for Part 1, this may include unit testing, integration testing, etc. **(weight 10)**

The final mark for this part is out of 20.

Submission

Please make sure you zip the testing code and the report (in pdf) to Testing.zip.

Referral Part 3: Programming in C

For this part, you should finish the following C programming task:

Consider the following structure

```
struct item
```

```
{
    int code;
    int quantity;
};
```

You must define **two** functions.

The first function `store` allocates space for an array of `n item` structures:

```
struct item *store(int n)
{
\\ function definition goes here
}
```

You must design the function so that it explicitly uses `malloc` and so that it returns values of the correct data type, and for all possible integer arguments (including invalid integer arguments).

The second function `average` computes and returns the average of the value of member `quantity` across `m item` structures:

```
double average (struct item *p, int m)
{
\\ function definition goes here
}
```

The function must return 0 for invalid arguments.

You should define the `average` function under the assumption that, when the function is called, space for storing structures must already have been allocated, and structures must have already been declared, initialised and stored in that space.

As an example, given three structures

```
struct item ONE = {1, 11};
struct item TWO = {2, 24};
struct item THREE = {3, 222};
```

function `store`, if correctly defined, should be used to allocate space to store structures `ONE`, `TWO` and `THREE`. These structures can then be stored in the allocated space. Function `average`, if defined correctly, can be called to compute the average value of member `quantity` across the three structures, i.e. (11 + 24 + 222) = 85.66.

Write your code in a file called `C.c`. Your code must include all the necessary header files, the definition of the structure `item` given above, the declaration and definition of the `store` and `average` functions. You file **must not** include any `main` routine. Submit a zipped version of your file called **C.zip.**

The final mark for this part is out of 20.