

**CSE1301 Individual Project**  
**Marked during Practical Sessions 11 and 12 (25 marks)**

This project requires you to solve problems and write software independently, and to integrate various aspects of C programming that you have learned this semester.

***IMPORTANT: most of the work related to this project will be done independently outside the pracs. The project will be marked during Pracs 11 and 12, but students should NOT rely on the prac time to complete the project. Students should use these pracs to complete the questions in the corresponding prac sheets.***

**PART 1: (8 marks)**

Produce a design document for the individual project. Your document should describe

- how data are represented in the program. You will need to describe all the variables and data structures used in your program;
- each of the software modules (functions) shown in the form of a structure chart;
- the control and data coupling of the modules.

The document should also include

- the program input/output (possibly with an operation of the system using the supplied executable code, i.e., a session with the system, including input and output),
- algorithms for each module, and
- test data for each function and for the overall program.

***Important:***

- You should discuss your ideas with your lecturer, tutor or demonstrator before you start coding, in order to make sure you understand correctly the requirements of the project.
- You should show your design document to your demonstrator before you start coding.
- During the design process, you may find that some requirements are not precise. As a result, you may make some assumptions regarding the operating conditions of your system. You should validate these assumptions with your demonstrator.

***Any assumptions should be stated clearly at the beginning of your design document.***

**PART 2: (17 marks)**

Code, document and test each function. Demonstrate the workings of your program.

***Requirements:***

- You are responsible for designing a program which addresses the requirements of the project.
- You should design your algorithms and your overall program so that your code is efficient and clean.
- You should test the components of your program separately, but you should end up with one working program at the end.

**SUBMISSION [TO BE HANDED IN AT THE BEGINNING OF THE PRAC]**

**PART 1:** To be marked by your demonstrator during Practical Session 11. **Make sure you retain a copy of the design document.**

**PART 2:** To be marked by the demonstrator during Practical Session 12. You are required to have your design document with you during marking.

## PROJECT SPECIFICATIONS

The aim of the project is to produce a program with which a human player can play the game “*Bejeweled*”. You may know this (quite popular) game from your own home computer. It is a single player game that is played against time. If you do not know the game, you can play an online version at <http://www.popcap.com/> and you can find many other implementations online.



Bejeweled is played on an 8 x 8 board (see above). Initially the whole board is filled with one stone per cell and the types of stones are chosen randomly. The task of the player is to swap adjacent stones such that three or more stones of the same type are directly adjacent in a row or column (diagonals do not count). Note that only such swaps are allowed that produce a row or column of three or more similar stones. A swap that does not achieve this cannot be executed. Swaps can only be in a column or in a row, but not diagonal.

If a swap changes the board such that at least three similar stones are directly adjacent in some row or column these stones will immediately disappear. The gaps that these leave behind will be filled by the remaining stones. Imagine the board standing upright. A gap is filled by letting the stones in the same column “fall” down into the gap. This of course produces gaps in the top row(s) of the board which will be filled with new random stones.

Consider the following example: Let the fields on the board can be numbered from (1,1) to (8,8). We will call the stone types “A”, “B”, ..., “F”. On the board below a swap of (8,8) with (7,8) would produce three “C” in (7,6)-(7,8). These would vanish, column 7 would be moved three fields down and (7,1) to (7,3) would be filled with new random stones. In contrast, a swap of, say, (1,1) and (1,2) is not permissible.

A	B	B	A	C	F	C	D
D	D	B	E	E	F	C	A
A	B	C	F	E	C	D	A
F	F	E	B	A	C	A	D
F	A	B	E	D	A	A	B
E	E	F	D	A	B	C	B
C	F	E	A	D	D	C	E
A	D	F	E	C	B	E	C

→swap (8,8) with (7,8)→

A	B	B	A	C	F	A	D
D	D	B	E	E	F	E	A
A	B	C	F	E	C	B	A
F	F	E	B	A	C	C	D
F	A	B	E	D	A	C	B
E	E	F	D	A	B	D	B
C	F	E	A	D	D	A	E
A	D	F	E	C	B	A	E

It is, of course, possible that a single swap causes three similar stones to be adjacent in a row as well as in a column. In this case the stones in the row as well as in the column disappear. It is also possible that the “falling down” of the stones causes a chain reaction by producing further chains.

Each stone removed from the board counts 10 point. Note that we use simplified scoring rules. As the game is played against time, the final score is the quotient of the number points divided by the total playing time in seconds.

For the purpose of this project, you may assume a board size of 6 \* 6 rows and columns and only 4 different types of stones. You will have to simulate the board using simple ASCII characters. Instead of filling the board with stones, you should use ASCII characters like "\$", "e", "#" and "?".

This is a single player game. Your program prompts the user for a move and executes the move (displaying the updated board) if the move is permissible. If the move is not permissible it should be rejected. The user can input a move by typing in the "from" coordinates and the "to" coordinates. Once the move has been executed, your program should display the score obtained from the move as well as the cumulative total score. The player can also choose to quit the game at any time, at which point the final score will be displayed.

To implement timing functions and for the random number generation you will need to use library functions. The short program below illustrates which to use and how to call them. Note that you must #include additional libraries (see header of program).

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

int main () {

    time_t time1, time2 /* a structure for times declared in the library (see K&R, App B11) */
    unsigned int seed;
    int max; /* number of random numbers to generate */
    int i; /* counter for random numbers */

    time1=time(NULL); /* get the current time */
    printf("input an unsigned int as a random seed\n");
    scanf("%u", &seed);
    /* a random number generator must be seeded. This means it must be supplied
       with a start value for a unique series of random numbers
    */

    time2=time(NULL); /* get the new current time */

    double diff = difftime(time2, time1); /* use library function to calculate time difference */
    printf("You made me wait %.0f seconds.\n How many random numbers do you want?", diff);

    scanf("%d",&max);
    srand(seed); /* now seed the random generator */

    for (i=1; i<=max; i++) {
        int myRand = rand() / (RAND_MAX/6);
        /* rand() generates random integers between 0 and RAND_MAX
           RAND_MAX is declared in the library
           so here we rescale for the number to be between 0 and 5
        */
        printf("Here is a random number between 0 and 5: %d\n", myRand);
    }

    return 0;
}
```

## DETAILED MARKING SCHEME

### PART 1 (1.5 + 1.5 + 1 + 2 + 2 = 8 marks)

The design document should include the following sections:

**Data Section (1.5 marks).** This section should discuss how data are represented in the program. It should include a list of the main data structures, and a description of the members of each structure. You should specify explicitly the actual C code for declaring structures and types. You should also describe how each structure should be used.

**Functions Section (1.5 marks).** You should apply the software engineering principles described in the lectures to structure the system into independent modules (represented by functions or sets of function). You should provide a description of the purpose and functionality of each module, and represent the control coupling of the modules in a structure chart. Label the chart with the name of the actual C function implementing each module. The chart should also be annotated to show the data coupling of the modules. Significant data used as parameters and return values should be among those described in the data section of the document.

**Program I/O Section (1 mark).** This section describes the input and output of your program and the user interface. A simple text-based user interface, such as that used in the example program provided, will do, but you need to describe clearly the options available to the user.

**Algorithms Section (2 marks).** You should provide algorithms for the modules that you must write or modify in the system. The algorithms can be described with pseudo-code or flow-charts. Note that you are not required to provide algorithms for functions that are fully supplied.

**Test Data Section (2 marks).** You should provide a list of data for testing each function that you must write or modify, as well as the program as a whole. The test data depend on your algorithms. **Each test set should include valid, invalid and boundary data.** You may also need to discuss the limitations of your program with regard to invalid data.

### PART 2 (2+8+1.5+1.5+2+2 = 17 marks)

The marks are broken down as follows:

Does the code compile?	2 marks
Does the code work correctly?	8 marks
Does the code satisfy the project specifications?	2 marks
Does the code match the design?	2 marks
Is the code written in good programming and layout style?	2 marks
Is the code well documented?	1 mark

Note: Only 1 mark for documentation, since much is provided with the supplied code.