University of Huddersfield
Department of Informatics
In-course Assignment Specification

Module Code and Title: CIT2112: Studio 1	
Assignment No. and Title: Portfolio I	
Assessment Tutor: Dr Minsi Chen	**Weighting Towards Module Grade:** 40%
Date Set: Thursday 28/09/2017	**Submission Date:** Thursday 14/12/2017

Learning Outcome(s) covered in this assignment:
Knowledge and Understanding

1. Have a sound knowledge of basic scripting and programming concepts
2. Be aware of typical software structures and design/development procedures

Abilities

1. Design appropriate technical solutions to a problem or brief, including recognising the risks/safety issues related to their safe operation
2. Construct prototype software
3. Test and evaluate a developed software prototype

Level of Collaboration:
This is an individual assignment.
Collaboration is not allowed.

Overview

The Autumn term assessment for CIT2112 consists of one programming problem.

Problem: The Game (Max. 70 pts)

The Game is a cooperative card game designed by Steffen Benndorf (see `https://boardgamegeek.com/boardgame/173090/game`). The game can played by 1-5 players using a draw pile consisting of 98 cards numbered from 2 to 99. The objective is to dispense as many cards as possible from the draw deck and hands into four designated play piles.

Your aim is to implement The Game as a text-based console application using the C/C++ language. In order to complete this task, you should first read the Mechanics and Rules to familiarise yourself with the game itself.

Mechanics and Rules

At the start of the game, players shuffle the deck of 98 cards to form the draw deck. Each player is dealt an initial hand of cards based on the following table:

No. Players	**Hand size**
Solo	8
2	7
3-5	6

Table 1: Hand size in relation to the number of players.

There are four **play piles** onto which cards can be placed during a player's turn. Two play piles are labelled "1"; the other two are labelled "100". In each play pile, only the top card is visible to the players.

In each turn, a player must place **at least** TWO cards from his/her hand into the play piles. A player is free to choose which play pile to place a card into. However, the placement of a new card must conform to the following rules:

- The play piles labelled "1" only allow a player to place cards in an ascending order, i.e. the new card being place must be greater than the card at the top of the pile
- Conversely, the play piles labelled "100" only allow a player to place cards in an descending order
- A player can place a card in the reverse order IF and ONLY IF the value difference between the new card and the top card in a pile is exactly 10. E.g. if the top card in a descending pile is 50, you can place a card with value 60 into that pile.

When a player's turn ends, he/she replenishes one's hand up to the designated hand size by drawing from the top of the draw deck. The next player will then start his/her turn. N.B. A player can end a turn at all will as long as a minimum of two cards have been played. There is no need to play as many cards as possible from a hand unless doing so is advantageous to winning the game.

The game ends when one of the following conditions are met:

- All cards including those in players' hands and the draw deck have been played. (Bravo,you have aced the game!)
- The active player is unable to play at lease two cards. The game ends immediately in this case!

The score is calculated at the end of the game by counting the number of remaining cards in the draw pile and in the players' hands.

Implementation

You must implement the Game as a text-based application using the C/C++ language. The application should offer an intuitive interface that enables a player to see the current state of the game. For example, the top card on each of the four play piles, the cards in a player's hand and the draw pile should be display on the screen. Please note, a graphical user interface is not required.

To help you better approach this task and structure your program, I have broken down the game flow into three phases. Your code should be organised accordingly thus maintaining a logical flow and self-explanatory structure.

Setup (Initialisation)

1. Reshuffle the initial draw pile of 98 cards
2. Deal each player an initial hand of cards
3. Empty the four play piles

Game Loop (Game Turn)

1. The active player choose a card to play
2. The active player choose a play pile to play the chosen card into
3. If at least two cards have been played, the active player can either continue or end his/her turn
4. When the active player ends his/her turn, replenish the hand up to a designated number of cards
5. Start a new turn for the next player if this is a multi-player game

End of Game

1. Check if the end of game conditions are met (you may allow players to voluntarily end a game)
2. Calculate and display the score on the screen
3. Ask if players want to start a new game or quit the program

The grading of your solution will be based on the following requirements.

- Basic Requirements:

  - Use appropriate data structures to represent a draw pile, a hand of cards and the four card stack
  - Implement the basic game flow including Setup, Game Loop and the end of game
  - Functions and variables must be named using a consistent standard
  - You must comment your code to explain what they do
  - Use of an appropriate and consistent naming convention for functions, variablesand structs/classes

- Advanced Requirements:
  - Implement local multiplayer supporting 2 - 5 players. Please note: in a multiplayer game, a player is NOT allowed to tell other players what are in their hand!
  - Create a simple computer controlled player
  - Store all cards played into each of the play pile

Deliverable and Submission

Your solution should consist of the following items:

- All source file(s) to compile and run your game
- A ONE page document in PDF format explaining how to compile and play the game

To submit your work, you must package the abovementioned items into a single ZIP file. The ZIP file must follow the naming convention `STUDENTID_theGame.zip`, N.B. this is case-sensitive. Please substitute `STUDENTID` with your own ID.

The packaged solution must be uploaded online via the designated UniLearn submission point by **23:59 Thursday 14/12/2017**. Your tutor will inform you when the submission point goes live and provide you with further instruction.

Grading Criteria

Grades	Criteria
80-100	The submitted work is exceptional and shows a comprehensive awareness of the dimensions of the topic. The solution demonstrates a level of novelty and offers an extension to existing techniques. Discussion and analysis are presented in a scholarly and scientific manner. Accurate data and strong evidences are used to support the discussion of any novel approach.
70-79	The submitted work is excellent and shows a full awareness of the dimensions of the topic. The solution is highly optimised and demonstrates individuality and creativity. Discussion offers insightful information and are well presented. Analysis is supported by accurate data and resources. **Correctness:** Your provided solutions are not only correct for all test cases, but also demonstrate additional features with high complexity and a certain degree of originality. **Techniques and Methods:** Your implementation is functional and fully optimised; you are fully aware of the runtime resource management and computational efficiency; the code base demonstrates some more advanced usage of C programming language and is extensible. **Code:** Excellent use of C style coding incorporating advanced features of C programming language; new functionalities and their implementation fully adhere to code reusability and extensibility.
60-69	The submitted work demonstrates some advanced techniques and includes other relevant aspects that shows a clear awareness of the dimensions of the topic. The implementation is efficient and scalable. Discussion and analysis are critical and well structured; data and resources are accurate. **Correctness:** Solutions and results are completely correct for all cases. **Techniques and Methods:** Your implementation is functional; you are fully aware of the runtime resource management and computational efficiency; the code base demonstrates some advanced usage of C language features and is extensible. **Code:** Very good use of C style coding with logical and coherent functions; the code is highly reusable and extensible.

50-59	The submitted work demonstrates advanced features and the implementation. The solution evidences a clear understanding of the problems and solutions, but it lacks in depth analysis and discussion. **Correctness:** Solutions and results are correct for a majority of cases, and only exhibit minor flaws and limitations in rare cases. **Techniques and Methods:** Your implementation is functional; some consideration was made to runtime and resources efficiency by using adequate data structure and algorithms; the code base demonstrates a good level of procedural programming concept. **Code:** Good use of C style coding; implementation is clear to the reader; little redundancy in code and good code reusability.
40-49	The submitted work is functional and exhibiting very limited results required for the pass grade. Although sufficient understanding in the topic is shown, it lacks analysis. **Correctness:** Solutions and results exhibit major flaws and limitations. **Techniques and Methods:** Your implementation is functional; little consideration was made to runtime and resource; the code base is not extensible. **Code and Structure:** Poor use of C style coding; implementation is difficult to follow for the reader; noticeable redundancy in code.
35-39	The submitted work is clearly deficient. It does not meet all the requirements for the pass grade set in the assignment specification. The work does not evidence a sufficient understanding in the subject.
0-34	The submitted work is not relevant to the topic. There is little evidence that attempts were made to complete the work to the minimum standard.

Programming Resource

This assignment does not depend on any API and can be solely completed by using the standard C/C++ library. For details of functions and facilities from the standard C/C++ library, please refer to `http://www.cplusplus.com`.