CE4703 Computer Software 3
ED5071 Computer Engineering Fundamentals

Assignment 2: User-Defined Data Types and Dynamic Data Structures

1 Overview

This is a group project with groups of 3-5 students (groups with less than 3 students will not be accepted). Please add the names and ID numbers of your group members to the wiki page on SULIS (please add the members of your group also to the comments at the beginning of each of your source/header files). You can also leave messages there if you are looking for additional group members or for a group to join.

The task is to write a simple card game - please follow the instructions given below carefully: any solution not following these instruction will be deemed a failed project and will receive 0 marks.

The rules are as follows: Each player gets an equal number of cards - remaining cards are put "face down" in a deck on the table (the hidden deck). The top card of the hidden deck is turned face up. Each player in turn puts down a card from his hand - this card must match the topmost open card either in suit or rank (value). Thus, if the top card is Spade-Five a player can play any other card of suit Spade or any other suit with rank Five (Club-Five, Heart-Five, Diamond-Five or again Spade-Five). If a player is unable to play a card, s/he is picking up a card from the hidden deck and the game moves to the next player. Players that pick up a card cannot play this card until it is their turn again. The game is finished if one of the players has no cards left on his/her hand. If the hidden deck is empty before any player has an empty hand, the played cards are taken (the last played card is left as played deck), shuffled and then used as the new hidden deck.

2 Instructions

Please follow the instructions given below carefully: any solution not following these instruction will be deemed a failed project and will receive 0 marks.

2.1 Environment

Your solution needs to compile on a Windows machine using Visual Studio Community. Please provide your solution as zip archive that contains your entire Visual Studio Solution.

2.2 Data Design & Implementation

To facilitate the game, you initially are asked to design two user-defined data types, one for cards and another for a deck of cards. Once your data types are implemented, you can start developing the game. Your data types should be as general as possible, i.e. they should be useable for any type of card game that uses standard decks of cards (52 cards, suits Club, Spade, Heart and Diamond, for each suits ranks Two through to Ten, Jack, Queen, King, Ace).

Please make sure to add Doxygen comments to your source files. In addition to documenting all the functions, please make sure to also include a description of your data types Card ad CardDeck in your doxygen comments.

2.2.1 Data Type "Card"

Your data type "Card" needs to support all cards in a standard deck of cards: Use a structure (`struct`) to model a single card and use two separate enumeration types (`enum`) - one to model the suit of cards (Club, Spade, Heart, Diamond) and the other to model the rank of cards (Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King, Ace).

Make sure to add doxygen comments that explain how your data type Card is working.

2.2.2 Data Type "CardDeck"

For the purpose of this game, use standard packs of cards (52 cards, suits Club, Spade, Heart and Diamond, for each suit ranks Two through to Ten, Jack, Queen, King and Ace). A "CardDeck" is a collection of 0, 1 or more cards. Make sure that your data type "CardDeck" supports any number of packs of cards: At creation time of a "CardDeck" the user can indicate how many (complete) packs of cards are to be placed in the "CardDeck". Thus, your implementation cannot use a fixed size array - you must use a dynamic data structure (dynamic memory allocation).

“CardDeck” Operations
Think about what kind of operations are usefull for a deck of cards (e.g. shuffle, sorting, adding or removing a card (either to/from top or at random location) and so on). For each operations, add a function that implements the operation.

Your data type “CardDeck” must support sorting of cards - pick any of the sorting algorithms discussed in the lecture and implement it to sort the cards in your card deck.

Make sure to add doxygen comments that explain how your data type CardDeck is working.

2.3 Game Development

Implement the game outlined above for two players as follows:

- Follow the K&R coding style.
- Prompt the user to enter the number of packs of card (each of 52 cards) to be used in the game.
- Create an initialised and shuffled CardDeck for the hidden deck (with the number of packs of cards as specified by the user).
- Create three empty CardDecks - one for each player and one to hold the played cards (played deck).
- Deal out 8 cards to each player by picking them from the top of the hidden deck - give one card to each player in turn, i.e. the first card in the shuffled deck goes to the first player, the second card to the second player, the third card again to the first player and so on.
- After all cards are dealt, sort the card deck for each player and print the cards of each player to the screen (please use a separate line for each player).
- Finally, simulate the game:
  - Remove the topmost card from the hidden deck, print it to the screen and move it to the played deck.
  - If player one has a matching card, let player one play this card (remove it from his hand and move it to the played deck).
  - If player one has no matching card, s/he picks the topmost card from the shuffled deck. Add this to the player’s card deck and use your chosen sorting algorithm to sort the player’s card deck.
  - In either case, display a text message (i.e. “Player one played card *xxx*” or “Player one picks card *xxx* from the shuffled deck”) and print all the cards of the player to the screen.
  - Afterwards, do the same for player two.

  - Keep iterating until the game is over (one player has no cards left).
  - If at any point in the game the hidden deck is empty, transfer the played cards to the hidden deck and shuffle it. However, the last played card should remain in the played deck.

3 Marking

The marks for this programming project are distributed as follows:

Well formatted source code for your data type "Card" and all operations (distributed over suitable files):	10
Well formatted source code for your data type "CardDeck" and it's operations (distributed over suitable files):	20
Implementation of sorting algorithm for "CardDeck":	15
Implementation of the game:	30
Complete Doxygen documentation:	25
Penalties:	
Insufficient comments:	Up to -30%
Poor code format/Not following K&R coding style:	Up to -30%
Poor file structure	Up to -15%
Not using dynamic dynamic data structure	-67%
Bad coding style (e.g. using `goto` or global variables)	Up to -50%
Total	**100**[1]

4 Deadline & Submission

Deadline for submission of your project solution is 17:00h on Friday, 01.12.2023 (week 12). Please submit your solution as a single zip archive via the module's Brightspace page - only one group member should submit the solution (result comments will be returned to all group members individually). If multiple solutions are submitted for a group, I will only mark the one that I find first (I assume that all submissions are the same!).

A complete solution consists of the following:

- Implementation of data type "Card"
- Implementation of data type "CardDeck"

[1]Will be scaled down to 20% for module marking.

- Implementation of the game.
- Generated Doxygen documentation for the program.

Please ensure that your archieve **contains all** deliverables - I cannot accept any modifications to your submission after the deadline. I recommend to copy your zip file to some location, extract it and re-compile and run your application from that new location.

5 Hints & Comments

- All input/output should be console based - no need for a GUI.
- There is no need to implement any form of intelligence/strategy - simply play the first card that matches.
- You **must** use a dynamic data structure for your ADT CardDeck - use of simple arrays is not sufficient!
- Comment your code with Doxygen:
  - Each file has comments at the top, outlining the content/purpose of the file, author(s) and last date of change.
  - Each function should be preceeded by comments.
  - You need (non-Doxygen) comments inside your functions explaining your code.
- Follow the K&R coding style.
- A shuffled deck of cards can be created by the following algorithm:
  1. Create an ordered deck with N cards (N = M * 52 where M is the number of packs of cards used).
  2. Create a second empty deck.
  3. Get a random number r that has a value between 1 and N.
  4. Remove the r-th card from the ordered stack (number of cards is reduced to N-1) and insert it at the beginning of the second stack.
  5. Decrement N by one.
  6. Repeat steps 3 to 5 until N equals 0.