Project 1

Title

Hearts\*

\*A text-based computer version of the

classic card game –

for educational purposes only

Course

**CIS-17A C++ - Programming Objects**

Section

**48593**

Due Date

**November 11, 2021**

Author

**Sonya Smalley**

# Introduction

The game Hearts is played using a standard 4 suit card deck and is best played with 4 players. However, modifications can be made to the deck to allow other numbers to play. Generally, playing with more than 5 players requires a second deck of cards to be added. There are ***many*** variations on this game, and I could find no two websites that agreed exactly on the rules. Therefore, I used the rules described at bicyclecards.com and on Wikipedia as the basis for my game, since the rules given at the two sites were very similar. No variations were used for this game.

The object of this game is to be the player with the lowest score. Each heart is worth 1 point and the Queen is worth 13 points. All other cards are worth 0 points. The entire deck is dealt out, and the player with the 2 of Clubs (2C) leads the first round with that card. If the 2C card has been removed to accommodate some a game with 3 or 5 players, the 3C starts the round. During a round, each player must follow suit if possible. If a player is void of the led suit, a card of any other suit may be led. However, a heart or the Queen of Spades (QS) may not be discarded during the first round. The highest card of the suit led wins the trick and leads the next round. Hearts may not be led until a heart or the QS has been discarded. The QS may be discarded at any time.

# Summary

## Stats

The game is 811 lines long. There are three structures: Card, Deck, and Player, with Card nested within both Deck and Player. There are 20 functions withing the program (not counting main), three of which should probably have been further partitioned as they are a bit large. The main function contains 15 variables, not including temporary variables within loops. There are two files used in the program for tracking the tricks won during each round. A text file records the full names of the cards (i.e. “Queen of Spades”), while a binary file records the structure of the cards (which includes the full text name, face and suit designations, face value, card position, and point value.

## Development – General Info & Challenges

This game took somewhere between 30 and 36 hours to program. Unfortunately, I ran out of time and was not able to implement three features that I had planned on having. The first was that I did not complete the validation of the user card choices. Namely, I wasn’t able to add validation to prevent them from playing hearts or the QS card at times that are forbidden by the rules. The second feature that I was unable to add was variable difficulty levels that adjust the programmed playing strategies appropriately. As it is, all players use the same strategies, which are ones that I feel cumulatively add up to an average playing style. Such players would be familiar with the game and neither overly sophisticated nor too easily beaten. The third I completely forgot until I was finishing this report. It is that I forgot about shooting the moon! If a player takes all the hearts and the QS for their tricks, they earn no points and everyone else earns 26 points.

I had never programmed a card game simulation, so I wasn’t sure how to deal with all the different aspects of cards and card games. Just dealing cards out to players seemed hugely problematic from my newby programmer perspective. I imagined having to randomly select each card and then use a conditional statement to determine if that card had already been given to a player, then to repeat the process if it had been until an unused card was selected. This seemed like a stupid idea and just contemplating it (particularly for a game like hearts where all cards are dealt out) had me worried that I would end up with nightmares where I was endlessly pulling a single card from a near endless pile of fresh decks of cards. So, I wandered through our class github and through the CIS-5 class github until I found Dr. Lehr’s CardDeckShuffle program. Then I hit myself on the forehead, rolled my eyes at myself for once again not seeing the obvious, and ***finally*** started programming for this project. The first thing I wrote was a near copy of the CardDeckShuffle program (see Hearts Deal Deck).

I encountered another stall when I tried to move from shuffling and dealing to sorting the players hands. I realized that if I left the cards as chars or strings all identified in a single 1d array, then sorting them by their names would be problematic (aka unnecessarily complicated) since Q comes after K in the alphabet. Also, I’d have to search through each element to identify the suit. Too much work. Too many things going on. I want to learn to program things that are efficient, not overly complicated. So I had to redo my cards. I already knew that I was probably going to have move towards putting them into structures and structures within structures in order to meet the requirements for this project. I therefore changed how the cards were handled in the file.

My next set of challenges came when I added a function to let the user choose which card to play. The first problem at this point was processing the user input. I wanted to allow the user to enter the values for a card’s suit and face value (such as 2C for Two of Clubs or QS for Queen of Spades). However, by this point, a card’s suit value was held in an enum Suit variable. This of course meant that the user could not directly input their suit choice into the suit variable, which is not really a problem. But I had used a short variable to temporarily hold the suit choice, thinking that I could directly assign the short’s value to the enum. I had forgotten that this is not possible, and it took me a little while to figure out the problem. The other challenge at this point was one that I often run into. Basically, my Boolean statements when I was validating the user’s card choice were causing all kinds of fun things to happen: infinite loops, approving invalid entries, rejecting valid entries, etc. Some of the things I did wrong with the Boolean statements to cause these issues: I added an else statement that allowed invalid choices to slip through, I used an and statement when I needed an or statement, I didn’t specifying conditions for my final else statement, and I was not being specific enough a couple of times.

# Description

## Sample Input/Output

Except at the end of the game, after play is over, the cards are represented by two characters. The first is for the card face, and the second is for the card suit. Both are described in Table 1 on the following page.

The game will periodically pause to allow the user time to read the generated output. The user must press the Enter key to continue with the game. When the user is prompted to choose a card to discard, the user must enter the characters for the card’s face and suit. For example, QS, Qs, qS, and qS are all acceptable inputs for choosing the Queen of Spades. Entering 2C would indicate that the user wants to discard the Two of Clubs card into the trick pile.

Table 1: Face and Suit Characters and Names

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Face Character | Face Name |  | Suit Character | Suit Name |
| 2 | Two |  | C | Clubs |
| 3 | Three |  | D | Diamonds |
| 4 | Four |  | S | Spades |
| 5 | Five |  | H | Hearts |
| 6 | Six |  |  |  |
| 7 | Seven |  |  |  |
| 8 | Eight |  |  |  |
| 9 | Nine |  |  |  |
| T | Ten |  |  |  |
| J | Jack |  |  |  |
| Q | Queen |  |  |  |
| K | King |  |  |  |
| A | Ace |  |  |  |

## Flowchart

Diagram

Description automatically generated

Diagram

Description automatically generated

## Pseudocode

Structures

Cards – 6 member variables

Deck – 2 member variables

Player – 5 member variables

Global constants

Enum Suits (5) declared in Cards.h

Short Size Name declared in Cards.h

Function prototypes

Set random number seed

Declare variables and initialize some

Display introduction to the game if the user so chooses

Determine if the user wants to see the deck and other players’ hands during play

Initialize major variables

Create an array of Players

Fill the Deck with Cards

Deal the Deck of Cards out to the Players

Determine the size of Player hands

Determine the number of rounds that will be played

Create an array to track player points and determine the winner

Shuffle the deck, deal the cards, organize players’ hands, and determine who goes first

**Loop to play the game one round at a time**

Open the Tricks files

The player who has the 2C or who won the last trick leads and discards the 2C on the first round or the card of their choice

The user can choose which card to discard

Loop to validate input

Programmed Players play 2C or choose a card

If it’s not round 1, Player’s hand is searched for the lowest card in the suit led

If they don’t have a card in the suit, Player’s hand is searched for the QS

If they don’t have the QS, Player’s hand is searched for the AS or the KS

If they don’t have the AS or KS, Player’s hand is sorted by face value, then searched for the highest card in a suit other than the one led, starting with any hearts in the hand

All of the cards played in the round are recorded in a temporary trick array, a text file, and a binary file

Discarded cards are “nulled” to unreal values in the Players’ hands

Determine which Player won the trick and will lead the next round

Record points earned

Close the Tricks files

Return who won the trick

Open the Tricks binary file

Loop by Player to retrieve and display the tricks won by each Player

Record the total points won by each Player in the Winner array

Note which Player(s) had the lowest points

Close the Tricks binary file

Loop to determine and announce winner(s)

Deallocate memory

End the game

## Variables

Table 3: Major Variables and Their Locations in Code

|  |  |  |
| --- | --- | --- |
| Concept | Datatype | Location in Code |
| **Card Member Variables** |  |  |
| name | 1D char array | Card.h file & Line 166 |
| face | Char | Card.h file & Line 212 |
| suit | Char | Card.h file & Line 213 |
| order | Short | Card.h file & Line 214 |
| value | Short | Card.h file & Line 215 |
| point | Short | Card.h file & Line 218 |
| **Deck Member Variables** |  |  |
| nCards | Short | Deck.h file & Line 208 |
| cards | Dynamic Card array | Deck.h file & Line 210 |
| **Player Member Variables** |  |  |
| name | String | Player.h file & Line 346 |
| hand | Dynamic Card array | Player.h file & Line 372 |
| trick | Card | Player.h file & Line 443 |
| points | Short | Player.h file & Line 477 |
| trkRnds | Vector<short> | Player.h file & Line 478 |
| **Files** |  |  |
| trkText | Fstream | Line 406 |
| trkBin | Fstream | Line 154 |
| **Main** |  | Lines 154-176 |
| nShufl | Short | Line 102 |
| nPlayrs | Short | Lines 86-90 |
| hndSize | Short | Line 88 |
| first | Short | Line 111 |
| round | Short | Line 134 |
| maxRnd | Short | Line 130 |
| deck | Deck | Line 87 |
| players | Dynamic Player array | Line 86 |
| broken | Bool | Line 134 |
| winner | Dynamic Short array | Lines 172-174 |
| minPts | Short | Line 174 |
| choice | Char | Line 80 |
| showAll | Bool | Line 81 |

## Concepts

Table 3: Concepts Used and Their Locations in Code

|  |  |
| --- | --- |
| Concept | Location in Code |
| **Pointers/Memory Allocation** |  |
| Pointer Variables | Line 67 |
| Arrays/Pointers | Line 93 |
| Function Parameters | Lines 32 & 314 |
| Memory Allocation | Line 90 |
| Return Parameters | Lines 30 & 277 |
| **Char Arrays and Strings** |  |
| C-Strings | Lines 245 to 270 |
| Strings | Lines 336-337 |
| **Structured Data** |  |
| Arrays | Line 289 |
| Nested | Line 364 |
| Function Arguments | Lines 31 & 298 |
| Function Return | Lines 30 & 277 |
| Pointers | Line 64 |
| Enumeration | Lines 217-223 |
| **Binary Files** |  |
| Function Parameters | Lines 34-35 & 399-400 |
| Multiple Files | Lines 406-408 |
| Binary Files | Lines 154-176 |
| Records with Structures | Lines 154-176 |
| Random Access Files | Lines 154-176 |
| Input/Output Simultaneous | Line 154 |

# References

[CardDeckShuffle](https://github.com/ml1150258/2021_Summer_CSC_CIS_5/tree/master/Projects/Project2/CardDeckShuffle) – A program by Dr. Mark Lehr demonstrating generating a deck of cards and shuffling the deck (click link to see the code in github).

MarkSort – Dr. Mark Lehr designed this easy-to-remember sort function and shares it with his students (it’s my go-to sort because it’s quick, easy to remember, and suitable for my purposes).

# Program

There are eleven versions of this game, beginning with Hearts Deal Cards, and running through versions of it, Hearts Basic, Hearts Add User, and Hearts Add Binary Files. Hearts vFinal is the final version. Screenshots are included below:

Text, application

Description automatically generated