

# Hearts Assembly Technical Report

Benjie Eisen  
CSI-370 Computer Architecture  
benjie.eisen@mymail.champlain.edu

December 11, 2022

## What

The project I decided to create was the classic card game Hearts entirely in assembly. To create the game, I wrote down all of the parts of interactions I would need detailing how it would be played with the four players of one of which would be the human player.

## Why

The reason that I chose this project topic was that I used to really playing this game online and with my family. It was also a card game which is something that I have wanted to code up for a while. I decided to do a card game when go fish was brought up in class as one of the things that someone had done in the past and Hearts is similar but a bit more fun.

## Game Rules

The game Hearts is played with four players using a standard pack of 52 playing cards. The goal of the game is to get as few points as possible. When one of the players gets the agreed-upon amount of points, the player with the fewest points wins the game. The game is played in many rounds consisting of a passing phase and 13 tricks each. The players are dealt their cards and then pass three of them to the left in the first round, to the right in the second round, and straight across in the third round, and there is no passing for the fourth round (It repeats on subsequent rounds). The player with the 2 of clubs starts the first trick of the round and then each player subsequently plays a card in clockwise order. If the player has a card of the suit that starts the trick, they must play it. If the player does not have any of that suit then they may play a card of any suit. If you play the highest card in a trick of the starting suit, you get to start the next trick. You may not start a trick with a heart unless someone else has already played a heart on a previous trick. If you win a trick then you get points added to your score for that round. Each heart card is worth 1 point and the Queen of Spades is worth 13 points. If you somehow manage to win all the tricks with points and get 26 points for the round, you instead get 0 points that round and the other 3 players each get 26 points added to their totals (Remember points are bad).

## How

The way that I made the program was using x86.64 assembly. I used the MASM dialect because I made it on my Windows 10 laptop and that the language that Windows understands. The IDE that I used was Visual

Studio 2019. I originally also used C++ for the inputs and outputs of the programs but I changed that to only using assembly through the Windows APIs. I probably modularized it as there a ton of functions that just call other functions and it probably could have been condensed a bit better.

## Specifics

### I/O details

To print out strings and read in strings, I used Windows APIs. To take in strings I used ReadConsoleA to take in the user's input and store it in the Input Buffer to be converted from ascii characters into the card itself.

```
1  mov rcx, ConsoleInputHandle
2  lea rdx, InputBuffer
3  mov r8, 4
4  lea r9, InputBufferNum
5  call ReadConsoleA
```

To print out to the console I used the WriteConsoleA API. I made it into a macro because I used it so many times in many different places.

```
1  print MACRO string, length
2  mov rcx, ConsoleOutputHandle
3      lea rdx, string
4      mov r8, length
5      mov r9, 0
6      call WriteConsoleA
7  ENDM
```

I was then able to call it to print out a string by doing

```
1      print PlayMessage, 12
```

for example which would print out the PlayMessage string and it is 12 characters long. I also had a NewLine string which I could use to go to the next line when I needed to. The Console Input and Output handles are defined in the initialization by using the GetStdHandle API which gets the handles of both the input and output for the console

```
1  mov rcx, -11
2  call GetStdHandle
3  mov ConsoleOutputHandle, rax
4  mov rcx, -10
5  call GetStdHandle
6  mov ConsoleInputHandle, rax
```

The -11 is the device value for the default output device and -10 is the device value for the default input device.

### Functions

- There were many functions that were used in the making of this program. The first one is the main function that just calls the other functions.

```
1  _main PROC
2  call _initialize
3  mainLoop:
```

```

4  call _handInitialize
5  call _pass
6  call _play

```

- The next function is the hand initialize function which creates the deck and shuffles the cards in it. This is one of the few places I used random number generation.

```

1  shuffleLoop:
2  call _getRand
3  div r11
4  add rdx,rdi
5  mov al,[rbx+rdi]
6  xchg al,[rdx]
7  mov [rbx+rdi],al
8  inc rbx
9  cmp r11,rbx
10 jg shuffleLoop

```

- Some more functions that I made were bot and human play functions. For the human play it gets input from the user of which card to play. For the bot play it just uses the first card that it has that it is legal to play. The hand is randomized at the beginning of the round so it is random what card will be played. After that it guides through rest of the different parts of the process for a turn.
- An important function is the checkIfPlayable function. It takes in a card byte and also looks at the current player variable. It

```

1      playableLoop:
2  mov dh,0f0h
3  and dh,BYTE PTR [rdi]
4  cmp dl,dh
5  je playableReturn
6  inc rdi
7  loop playableLoop
8  jmp canPlay
9  playableReturn:

```

## Data Structures

The main data structure I used for this project was BYTE arrays that were 16 bytes long. Those were for the players' hands and usually only stored up to 13 cards except when passing at the beginning of the round when it can get up to 16 cards in their hands. I also used bytes to store the scores of the players(both for the round and between rounds). I used a 52 BYTE array at the beginning to initialize the hands but after that, it is not necessary until the next round starts and a new deck of cards needs to be generated.

## Challenges and Solutions

### Bot AI

Making an AI for the bot was one of the things that I was really struggling with. I ended up just removing all choice from the bot and it chooses the first card from its already shuffled up hand that it can play.

## Random Number Generation

Another problem that I had was with generating random numbers. I initially used rand and srand to generate random numbers but when I moved away from passing it through C++, I lost access to those C++ compiler-specific functions. I ended up using the rdtsc instruction in my getRand function which takes in the time stamp and mods that by the amount that I wanted to get between by taking it as an input to the function and outputting the random number

```
1  _getRand PROC ; (Max_Number)
2  push rcx
3  push rbx
4  xor rdx,rdx
5  mov rbx,rcx
6  rdtsc
7  xor rdx,rdx
8  nop
9  div rbx
10 mov rax,rdx
11 pop rbx
12 pop rcx
13 ret
14 _getRand ENDP
```

## Not having valid cards

Another problem that I had was that if I didn't have any cards of the starting suit, it showed that I didn't have any cards that I could play. If I don't have any cards of the correct suit than I should be able to play any of my cards without having a possibility to win the trick. I realized that it was caused by me not clearing out the c register before moving the amount of cards in my hand to the cl register.

```
1  xor rcx,rcx
2  mov cl,[r9] ;amount of cards in hand
```

This caused the loop to keep reading past that players hand and into other code which would always have bytes that make the cards invalid.

## Input validation

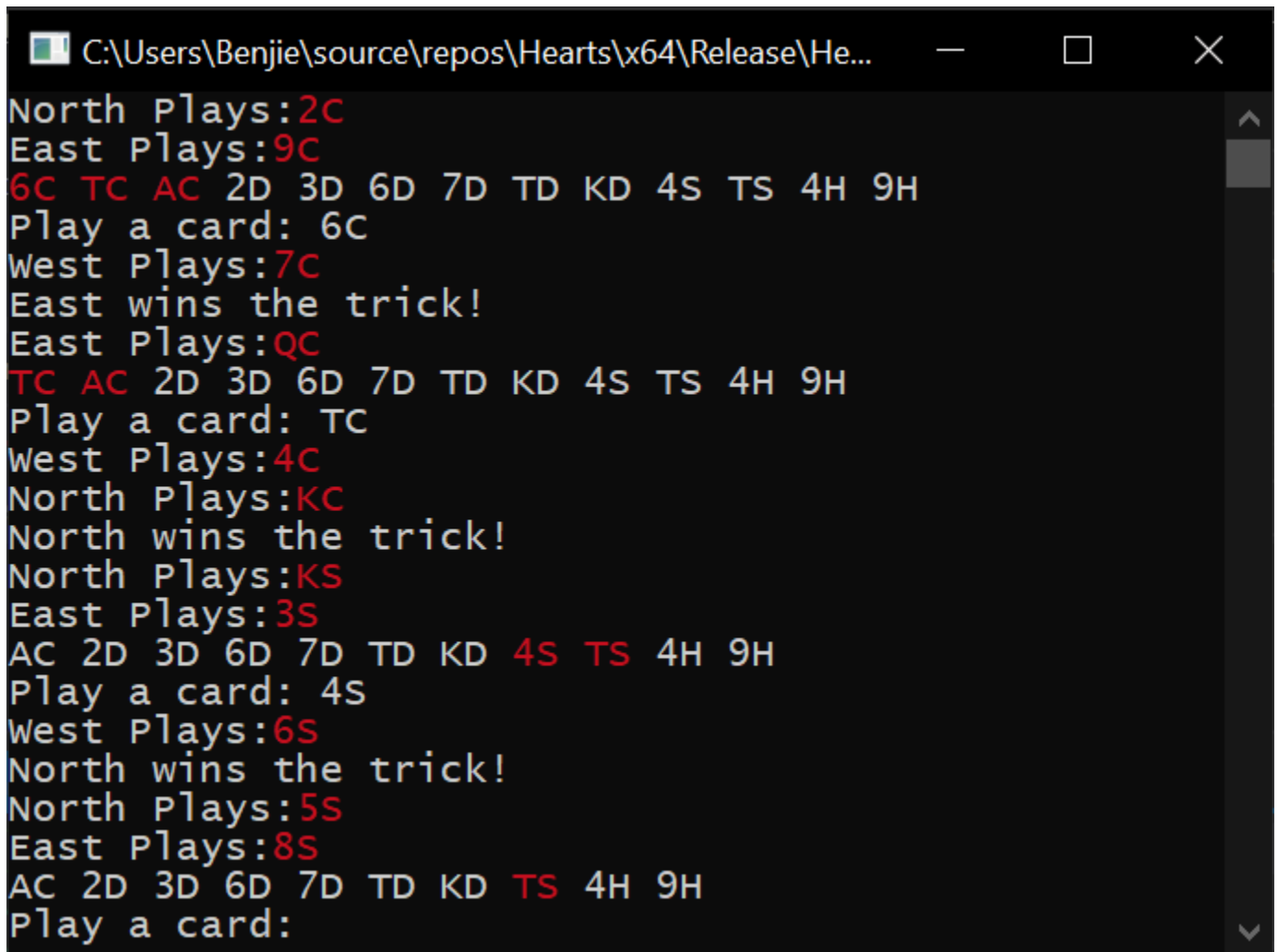
One other issue that I had was validating the input of a card name. This was for when the player wants to play a card from one the cards that they have in their hand. They have to type it in like RankSuit so for example the ten of clubs would be TC. It is case sensitive so they must type it in exactly like that. I used strings for the rank and suit that the validation function would search through to see if it is in those lists. The strings were uppercase except for the numbers.

```
1  Rank DB "23456789TJQKA",0
2  Suit DB "CDSH",0

1  call ReadConsoleA
2  mov rcx, 13
3  lea rdi,Rank
4  mov al,[InputBuffer]
5  REPZ SCASB
6  jne invalid
7  mov bl,3Eh
```

```
8  sub bl,cl
9  mov rcx, 4
10 lea rdi,Suit
11 mov al,[InputBuffer+1]
12 REPZ SCASB
```

## Other Visualizations



```
C:\Users\Benjie\source\repos\Hearts\x64\Release\He...
North Plays:2C
East Plays:9C
6C TC AC 2D 3D 6D 7D TD KD 4S TS 4H 9H
Play a card: 6C
West Plays:7C
East wins the trick!
East Plays:QC
TC AC 2D 3D 6D 7D TD KD 4S TS 4H 9H
Play a card: TC
West Plays:4C
North Plays:KC
North wins the trick!
North Plays:KS
East Plays:3S
AC 2D 3D 6D 7D TD KD 4S TS 4H 9H
Play a card: 4S
West Plays:6S
North wins the trick!
North Plays:5S
East Plays:8S
AC 2D 3D 6D 7D TD KD TS 4H 9H
Play a card:
```

Figure 1: Image of the game running after a few tricks

## References

- [1] Hall and Slonka, "Assembly Programming and Computer Architecture for Software Engineers". Prospect Press, 2017.
- [2] "Lists" - Overleaf Online Latex Editor, Date not provided  
<https://www.overleaf.com/learn/latex/Lists>
- [3] "Hearts". Bicycle Playing Cards, date not provided  
<https://bicyclecards.com/how-to-play/hearts>