Project 1

Hearts (v3)

CIS-17A

Stephanie Peacock

November 12, 2023

Introduction

Title : Hearts

The game of Hearts is usually a four player game. Each player is dealt 13 cards (dividing the deck equally). The player's intent is to score as low as possible. Each hand consists of thirteen tricks, in which each player plays one card. The lead player sets the suit for the trick, and remaining players must match the suit, if possible, else they can play the card of their choice.

The player with the two of clubs plays the first trick of each hand, usually point cards are not allowed during that trick, nor are hearts allowed to lead unless a heart has already been played. The winner of the trick (player with the highest card of the lead card suit) plays the first card of the next trick. This continues until all cards have been played.

Players score their hands as follows:

Any Heart card is worth one point. Queen of Spades is worth 13 points

If a player manages to collect all 14 point cards (for a total of 26 points) they score 0 points for the hand, and all remaining players are scored 26 points. This is called "Shooting the Moon."

Players play until an agreed upon total point count is reached, usually 50 or 100. Once one player reaches that point, the game is over, and the player with the lowest point tally is considered the winner.

Summary

Project size: 659 lines

<u>Major Variables:</u>

- Global variable: const int DECK = 52
- Structure CARDS
  1. int cards[13]
- Enum Order
  1. FIRST
  2. SECOND
  3. THIRD
  4. FOURTH

- Structure Player
  1. char *name //C-String to hold player name
  2. Order order //enumerated type
  3. Cards hand //nested structure
  4. int choice //to hold the choice
  5. bool match //used for scoring and suit enforcement
  6. int tScore //current trick score
  7. int score //current game score

- Structure Show
  1. string pshow[13]
  2. string show[DECK]

My Modifications from CIS5 v3:

I greatly simplified several sections, most importantly the trick function and the section of code enforcing the player to follow suit (it is located in the play function now). The changes to trick in particular saved me almost 600 lines of code.

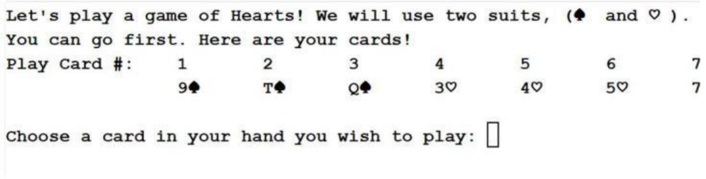
CSC/CIS 17A Project 1 Check-Off Sheet

Chapter	Section	Concept	Points for	Location in	Comments
			Inclusion	Code	
9		**Pointers/Memory Allocation**			
	1	Memory Addresses			
	2	Pointer Variables	5	Throughout	Player, name, deck, show
	3	Arrays/Pointers	5	Throughout	Char, int, structure arrays
	4	Pointer Arithmetic			
	5	Pointer Initialization			
	6	Comparing			
	7	Function Parameters	5	throughout	Most functions
	8	Memory Allocation	5	40,42,61,170	p, p.name, deck
	9	Return Parameters	5		Did not use
	10	Smart Pointers			
10		**Char Arrays and Strings**			
	1	Testing			
	2	Case Conversion			
	3	C-Strings	10	Hearts.h - 21	p.name
	4	Library Functions			
	5	Conversion			
	6	Your own functions			
	7	Strings	10	Hearts.h – 31,32	pfv. show, pfv.show
11		**Structured Data**			
	1	Abstract Data Types			
	2	Data			
	3	Access			
	4	Initialize			
	5	Arrays	5	37, 40, 116	Show, Player, Win
	6	Nested	5	Hearts.h - 23	Cards nested in Player (p.hand.cards)
	7	Function Arguments	5	throughout	Most functions
	8	Function Return	5	195	int indx = linSrch()
	9	Pointers	5	35,36, most functions	Deck[], p[], p.name[], most functions
	10	Unions ****			
	11	Enumeration	5	throughout	FIRST-FOURTH: deal() play() playCard()
12		**Binary Files**			
	1	File Operations			
	2	Formatting	2		**Did not use**
	3	Function Parameters	2	176	Deal function
	4	Error Testing			
	5	Member Functions	2		Getline(fv,pfv[0].show[i]
	6	Multiple Files	2	114, 169, 175	Saved.bin, cards.bin, cards.txt
	7	Binary Files	5	114, 169	Saved.bin, cards.bin
	8	Records with Structures	5	114-154	Read saved, overwrite saved, verify
	9	Random Access Files	5	118, 140, 145	seekg(0), seekp(0)
	10	Input/Output Simultaneous	2	114-154	Read saved, overwrite saved, verify
	Total		100		

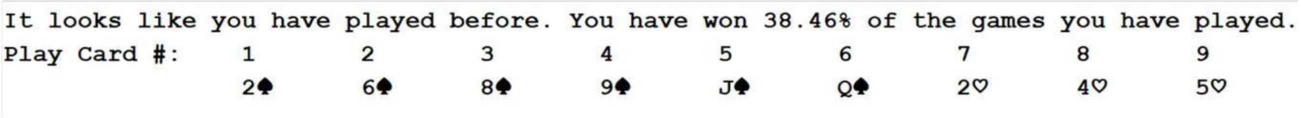
Game Play Screenshots

Some screenshots are from previous versions, the major changes are included as new screenshots.

The game begins with an introduction that changes, depending upon if the player has ever played or not. If they have not, they are given a brief introduction to the game. If the player has played previously, they are told what percentage of games they have won so far.

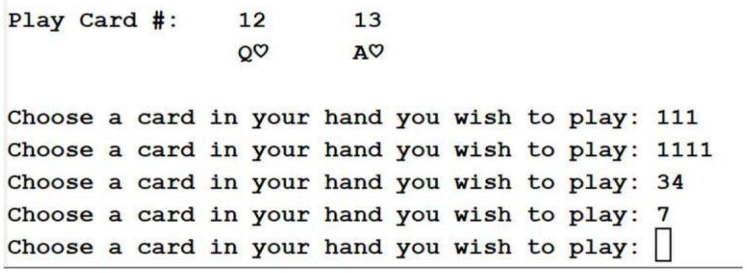


```
Let's play a game of Hearts! We will use two suits, (♠  and ♡ ).
You can go first. Here are your cards!
Play Card #:    1        2        3        4        5        6        7
                9♠       T♠       Q♠       3♡       4♡       5♡       7

Choose a card in your hand you wish to play: ▯
```



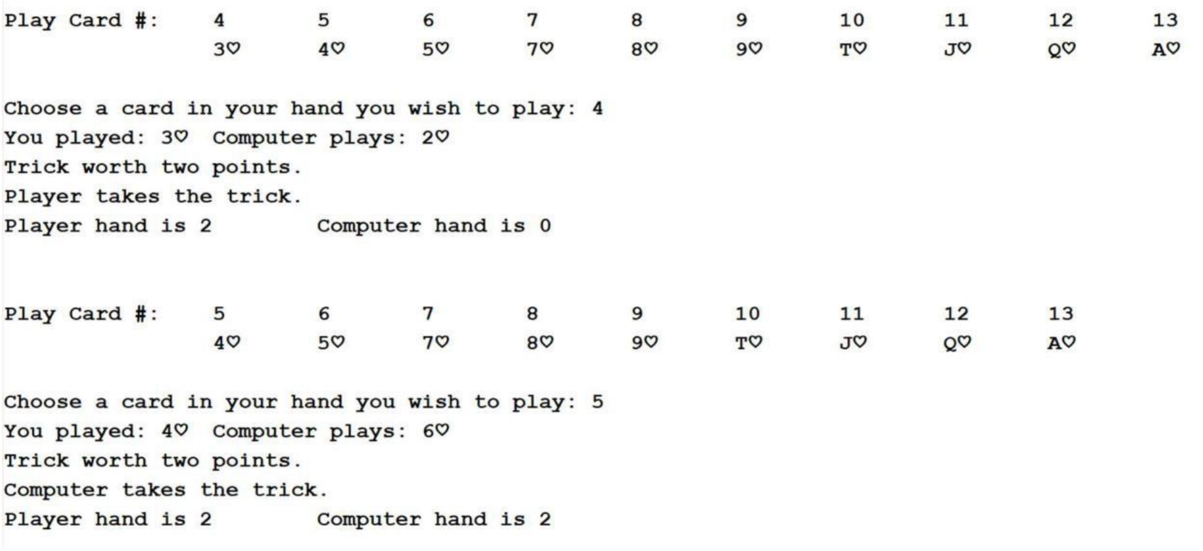
```
It looks like you have played before. You have won 38.46% of the games you have played.
Play Card #:    1       2       3       4       5       6       7       8       9
                2♠      6♠      8♠      9♠      J♠      Q♠      2♡      4♡      5♡
```

Following the introduction, the player is shown their hand, and asked to choose a card. If they do not choose a card currently in their hand, or choose a card that has been previously played, they will be prompted to reenter their card.



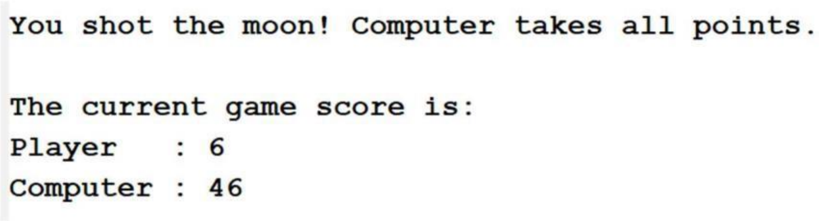
```
Play Card #:    12      13
                Q♡      A♡

Choose a card in your hand you wish to play: 111
Choose a card in your hand you wish to play: 1111
Choose a card in your hand you wish to play: 34
Choose a card in your hand you wish to play: 7
Choose a card in your hand you wish to play: ▯
```

As each trick is played, both player cards are shown, then scored. Played cards are removed from the player's visible hand, so they don't try to re-play them.



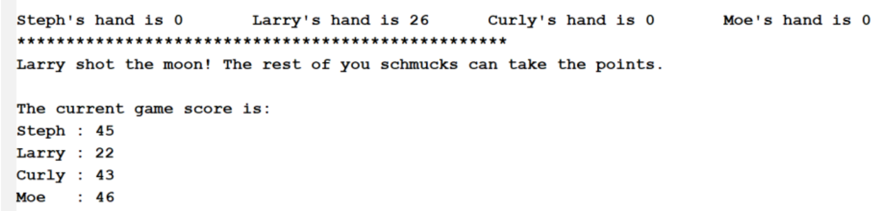
```
Play Card #:    4       5       6       7       8       9       10      11      12      13
                3♡      4♡      5♡      7♡      8♡      9♡      T♡      J♡      Q♡      A♡

Choose a card in your hand you wish to play: 4
You played: 3♡  Computer plays: 2♡
Trick worth two points.
Player takes the trick.
Player hand is 2        Computer hand is 0


Play Card #:    5       6       7       8       9       10      11      12      13
                4♡      5♡      7♡      8♡      9♡      T♡      J♡      Q♡      A♡

Choose a card in your hand you wish to play: 5
You played: 4♡  Computer plays: 6♡
Trick worth two points.
Computer takes the trick.
Player hand is 2        Computer hand is 2
```

If a player manages to shoot the moon, they are given a bonus message, and points are allocated to the computer instead.

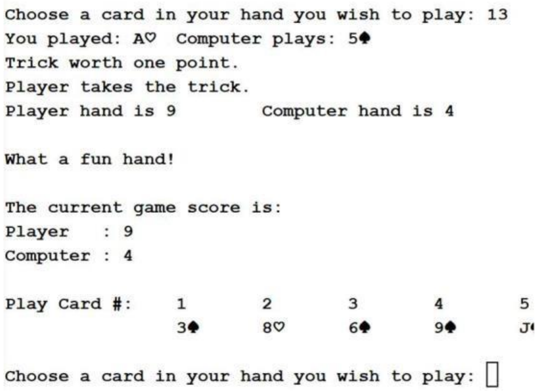


```
You shot the moon! Computer takes all points.

The current game score is:
Player   : 6
Computer : 46
```

I added a fun message if one of the Stooges manages to shoot the moon!



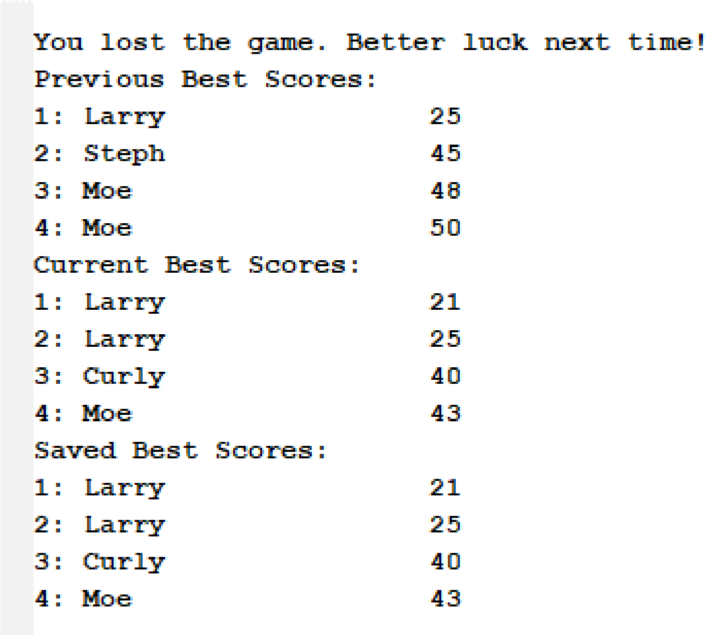
```
Steph's hand is 0       Larry's hand is 26      Curly's hand is 0       Moe's hand is 0
**************************************************
Larry shot the moon! The rest of you schmucks can take the points.

The current game score is:
Steph : 45
Larry : 22
Curly : 43
Moe   : 46
```



```
Choose a card in your hand you wish to play: 13
You played: A♡  Computer plays: 5♠
Trick worth one point.
Player takes the trick.
Player hand is 9        Computer hand is 4

What a fun hand!

The current game score is:
Player   : 9
Computer : 4

Play Card #:    1       2       3       4       5
                3♠      8♡      6♠      9♠      J

Choose a card in your hand you wish to play: ▯
```

The player can't always shoot the moon, however, so the usual scoring for the hand is displayed.

After the game is finished we check the saved file to view the previous best scores, then compare the 4 current game scores to the 4 previous. The best 4 of all 8 are listed as the current and written to the save file. Lastly, we read in the save file to verify the data transferred correctly.



```
You lost the game. Better luck next time!
Previous Best Scores:
1: Larry                25
2: Steph                45
3: Moe                  48
4: Moe                  50
Current Best Scores:
1: Larry                21
2: Larry                25
3: Curly                40
4: Moe                  43
Saved Best Scores:
1: Larry                21
2: Larry                25
3: Curly                40
4: Moe                  43
```

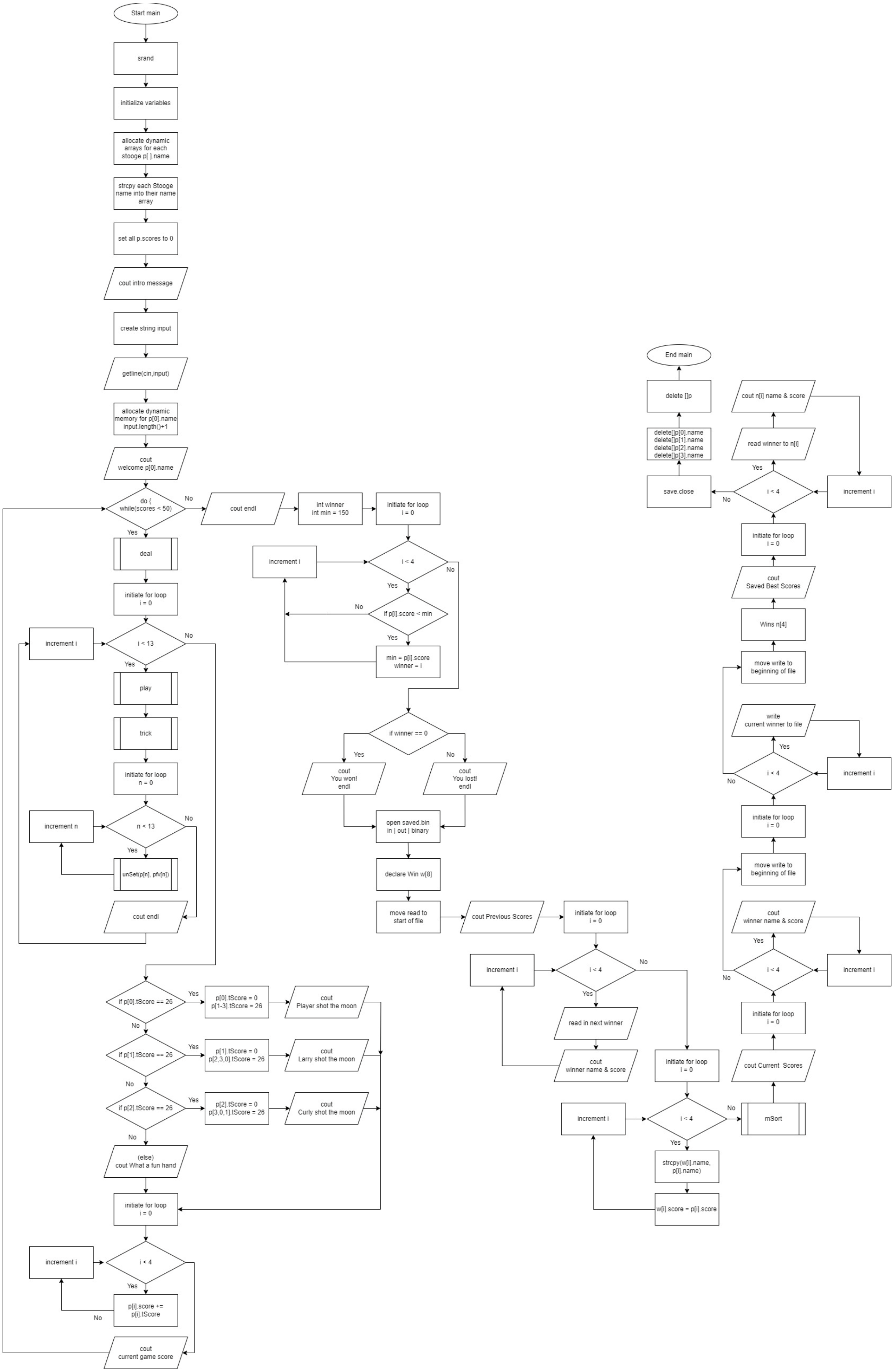
Pseudocode

```
//Initialize system libraries
//User Libraries
//Begin Main
    //Setting Random Number Seed
    //Initialize variables/input values
    //Create the structures
    //Make the NPC name arrays and fill them
    //set all game scores to 0
    //Cout Intro
    //create string to hold input
    //get the player name – getline(cin,input)
    //allocate memory for name array
    //copy input into name array
    //start the game loop
    //do{
        //Deal the cards  - deal(cards,fv,deck,p,DECK,pfv)
        ***BEGIN DEAL***
                        //open cards.bin
                        //allocate memory for new dynamic deck array
                        //read cards.bin into deck array
                        //close the file
                        //open cards.dat to get face values
                        //initiate a loop to run for whole deck (52)
                            // getline(fv,pfv[0].show[i])
                        //endl
                        //close the file
                        //shuffle()
                            ***BEGIN SHUFFLE***
                            //initiate loop run 52 times
                                //n = I + random number between 1-52
                                //swap deck[i] & deck[n]
                                //swap pfv[i] & pfv[n]
                            ***END SHUFFLE***
                    //initiate loop to pass out cards – run 13 times
                        //p[0] gets cards i & pfv[0] gets show i
                    //p[0] gets cards i+13 & pfv[0] gets show i+13
                    //p[0] gets cards i+26 & pfv[0] gets show i+26
                    //p[0] gets cards i+39 & pfv[0] gets show i+39
                        //sort all the hands with mSort()
                            ***BEGIN MSORT***
                            //initiate outermost loop n to increment players (run 4 times)
                                //initiate middle loop j – run 13 times
                                    //initiate inner loop i – run 13 times
                                        //if player's current j card > current i card
                                            //create temp and set to current player's card[i]
                                            //card[j] = card[i]
                                            //card[i] = temp
                                            //create temp string and set to current pfv.pshow[i]
                                            //pshow[j] = pshow[i]
                                            //pshow[i] = temp
                            ***END MSORT***
        //find the player order
  //int indx = linSrch(deck,DECK)
            ***BEGIN LINSRCH***
            //declare indx, set to 0
            //declare bool found, set to salse
            //while still cards left & not found
                //if current indx == 1
                    //set found to true & exit loop
                //else increment indx
            //return indx
            ***END LINSRCH***
        //if     (indx < 13)
    //p[0].order = FIRST; p[1].order = SECOND; p[2].order = THIRD; p[3].order = FOURTH
        //else if(indx > 12 && indx < 26)
    //p[1].order = FIRST; p[2].order = SECOND; p[3].order = THIRD; p[0].order = FOURTH
        //else if(indx > 25 && indx < 39)
    //p[2].order = FIRST; p[3].order = SECOND; p[0].order = THIRD; p[1].order = FOURTH
        //else
    //p[3].order = FIRST; p[0].order = SECOND; p[1].order = THIRD; p[2].order = FOURTH
        //initiate loop to reset the match status & tScores – run 4 times
        //p[i].match = false;
        //p[i].tScore = 0;
        //Delete the deck array since it's not needed until redeal
  ****END DEAL***
//begin Hand loop – repeat 13 times
        // play(p,pfv)
    **BEGIN PLAY***
  //set min higher than possible - used to find valid range
    //loop 4 times so each player goes
            //if player’s turn
      // print() player's cards
      ***BEGIN PRINT***
        //show choice options
        //if card 0 == 0 print empty string - else print 1
        //repeat for all 13 cards (add 1 to else value each time)
        //move to next line
        //begin loop to print card face values – run 13 times
          //cout current card face value
                **END PRINT***
      //set valid to false
            //run until valid choice is found – do {
            // ask for player choice
                //read in player choice
        // Begin validation - 2clubs validation if player is first
        //while FIRST & player’s first card is 2 clubs & choice is not 2 clubs
                    //ask for 2 clubs
                    //read in new choice
                    //if 2 clubs selected
            //set choice to valid & match to true (exit do while loop)
                //if player is first, doesn’t have 2 clubs, & selected card is !=0
                    //valid = true, match = true (exit do while loop)
        // if player not first, begin suit validation
                // Stooge that was first played CLUBS
                    // if player played CLUBS
                            //valid = true, match = true (exit do while loop)
        //begin loop to check for matching suit – run 13 times
                                    //if current card < min
                                            //set min to current card value
                            //if there is a valid match that wasn't played
                                    //while valid card not chosen
                                                    //prompt for valid choice
                                            //cin new choice
                                            //if new choice is matching suit
                                            //valid = true, match = true (exit do while loop)
                                    //else if no valid match accept any choice not 0
                                            //valid = true, match = false (exit do while loop)
// Stooge that was first played DIAMONDS
                    // if player played DIAMONDS
                                    //valid = true, match = true (exit do while loop)
    //begin loop to check for matching suit – run 13 times
                            //if current card < min
                                            //set min to current card value
                    //if there is a valid match that wasn't played
                            //while valid card not chosen
                                            //prompt for valid choice
                                                    //cin new choice
                                            //if new choice is matching suit
                                            //valid = true, match = true (exit do while loop)
                            //else if no valid match accept any choice not 0
                                    //valid = true, match = false (exit do while loop)
// Stooge that was first played SPADES
                    // if player played SPADES
                                    //valid = true, match = true (exit do while loop)
    //begin loop to check for matching suit – run 13 times
                            //if current card < min
                                            //set min to current card value
                    //if there is a valid match that wasn't played
                            //while valid card not chosen
                                            //prompt for valid choice
                                            //cin new choice
                                            //if new choice is matching suit
                                            //valid = true, match = true (exit do while loop)
                            //else if no valid match accept any choice not 0
                                    //valid = true, match = false (exit do while loop)
// Stooge that was first played HEARTS
                    // if player played HEARTS
                                    //valid = true, match = true (exit do while loop)
    //begin loop to check for matching suit – run 13 times
                            //if current card < min
                                            //set min to current card value
                    //if there is a valid match that wasn't played
                            //while valid card not chosen
                                            //prompt for valid choice
                                            //cin new choice
                                            //if new choice is matching suit
                                            //valid = true, match = true (exit do while loop)
                            //else if no valid match accept any choice not 0
                                    //valid = true, match = false (exit do while loop)
                    // end while loop ((p[0].choice < 1 || p[0].choice > 13) &&
                    p[0].hand.cards[p[0].choice-1] != 0 && !valid)
        //played(p[0,pfv[0)
            ***BEGIN PLAYED**
            //message to indicate who played
            //set switch to check for choice
                //case 1
                    //cout pfv.pshow[0]
                    //break
                //case 2
                    //cout pfv.pshow[1]
                    //break
                //repeat until case 12 (increment case by 1 each time)
                    //cout pfv.pshow[11] (increment element by 1 each time)
                    //break
                //default
                    //cout pfv.pshow[12]
                    //break
            ***END PLAYED***
        //set(p[0])
            ***BEGIN SET**
            //set choice to the value of the element 1 less than choice
            ***END SET***
// Larry's Turn
    //playCard(p,1)
                ***BEGIN PLAYCARD**
                //bool chose = false
                //if player is first
                    //if stooge has 2 clubs
                        //choice = 1, match = true, chosen = true
                    //else no 2 of clubs
                        //begin loop to check all 13 cards – run 13 times or
                        until a card is chosen (start at lowest)
                            //if current card != 0
                                //set choice to i+1, chosen  = true,
                                match = true, exit the loop
                //else
                    //begin loop s to check all 4 players
                        //if p[s] == FIRST
                            //declare variable min
                            // if p[s].choice < 14 then min = 0
          // else if p[s].choice < 27 then min = 14
          // else if p[s].choice < 40 then min = 27
          // else min = 40
          //int max = min + 13
          //begin loop to check all 13 cards – run 13 times or until a card is chosen (start
        at lowest)
            //if current card >= min & <= max & !=0
                //set choice to i+1, chosen  = true, match = true, exit the loop
          //if not chosen
            //begin loop to check all 13 cards – run 13 times or until a card is chosen
            (start at highest)
                //if current card == Queen Spades (37)
                    //set choice to j+1, chosen  = true, match = false, exit  loop
              //if current card == Ace Spades (39)
                    //set choice to j+1, chosen  = true, match = false, exit  loop
              //if current card == King Spades (38)
                    //set choice to j+1, chosen  = true, match = false, exit  loop
```

```
                        //if still not chosen play highest card
                            //begin loop at 12, run until 0 or a card is chosen
                                    //if current card != 0
                                            //set choice to j+1, chosen  = true, match = false, exit the
                    loop
                                    ***END PLAYCARD***
                    //played(p[1],pfv[1])   (see player for pseudocode)
                    //set(p[1])                             (see player for pseudocode)
    // Curly's Turn
            //playCard(p,2)                 (see Larry for pseudocode)
                    //played(p[2],pfv[2])   (see player for pseudocode)
                    //set(p[2])                             (see player for pseudocode)
    // Moe's Turn
            //playCard(p,3)                 (see Larry for pseudocode)
                    //played(p[3],pfv[3])   (see player for pseudocode)
                    //set(p[3])                             (see player for pseudocode)
                    ***END PLAY***
//score the trick(p)
    ***BEGIN TRICK***
    //set local counters score & max to 0
    //declare winner variable
    //begin loop to check all 4 players – run 4 times
            //if current player matches and their choice > max
                    //set max to current player choice
                    //set winner to current player
            //if player played a heart
                    //score += 1
            //if player played Queen of Spades
                    //score += 13
    //output trick value
    //output the trick winner
    //set the order – winner = FIRST
    //if winner == 0
            Player = FIRST, Larry = SECOND, Curly = THIRD, Moe = FOURTH
    //if winner == 1
            Larry = FIRST, Curly = SECOND, Moe = THIRD, Player = FOURTH
    //if winner == 2
            Curly = FIRST, Moe = SECOND, Player = THIRD, Larry = FOURTH
    //if winner == 3
            Moe = FIRST, Player = SECOND, Larry = THIRD, Curly = FOURTH
    //add score to winner’s tScore
    //begin loop to reset match – run 4 times
            //set current player’s match status to false
    //endl
    ***END TRICK***
//Begin loop to unset cards so they can’t be replayed – run 4 times
    //unset(p[n],pfv[n])
        ***BEGIN UNSET***
            //check all 13 options – whatever card choice matches, set the value to 0 so it can’t be
            replayed
            //do the same for corresponding pshow values (set to empty string)
        ***END UNSET***
    // cout trick break & endl
    // Check for Shooting the Moon
    //if any player gets 26 points in trick change their score to 0 and all other players to
    //cout end hand message
```

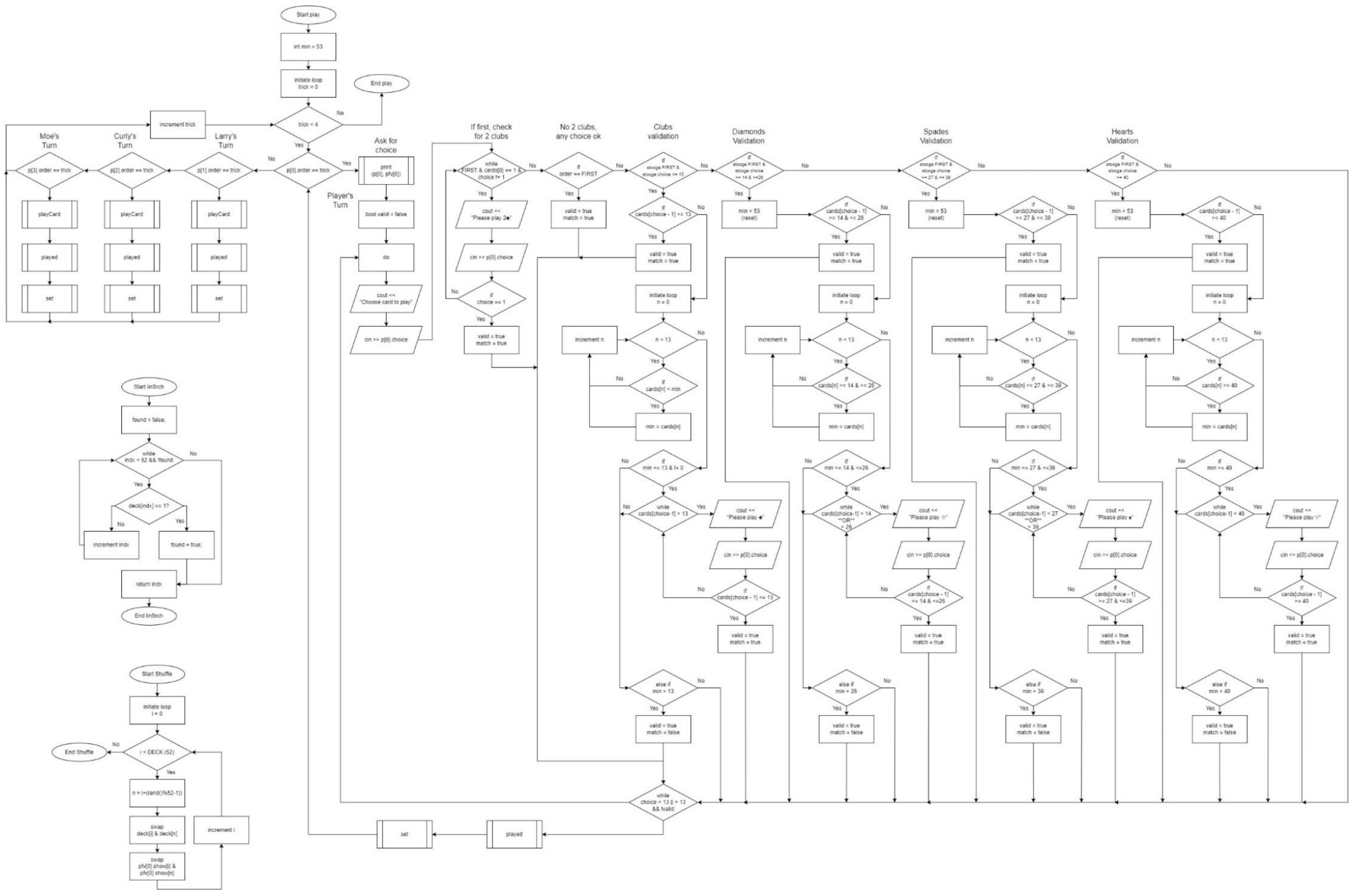
```
 //begin loop to add tscores to score – run 4 times
    //add current player tScore to their game score
    //output current game score for each player
//}while(p[0].score < 50 && p[1].score < 50 && p[2].score < 50 && p[3].score < 50);
    //endl
    //Find the winner – int winner, int min = 150
    //begin loop to check all 4 players
    //if current player score < min
    //set min to current player’s score
    //set winner to current loop number
    //if (winner == 0)
    //cout "You won the game!"
    //else a stooge won
        //cout “You lost”
//open the saved file – in | out | binary – to store the best scores
//create the structures – 4 for file 4 for current game (8 total)
//set file seek to beginning
    //output the old best scores
    //begin loop to get scores – run 4 times
        //read next structure
        //output name and score
//begin loop copy all 4 current game scores to winner structure
    //strcpy(w[i].name,p[i-4].name);
    //w[i].score = p[i-4].score;
//sort all 8 structures with mSort(w)
        ***BEGIN MSORT**
        //begin loop j – run 7 times
            //begin loop I – run 8 times
                //if w[j].score > w[i].score
                    //create temp structure – set to w[j]
                    //w[j] = w[i]
                    //w[i] = w[j]
    ***END MSORT***
    //output new best scores message
    //begin loop to output – run 4 times (we ignore the other 4 – they suck)
        //cout current name & score
//write the 4 best to file
//set write to beginning
//initiate loop – run 4 times
    //write current structure to the file
    //double check our write
    //set read to beginning again
    //create another structure to hold the test values
    //output the old best scores message
    //begin loop to read in saved data – run 4 times
        //read in current structure
        //cout current name and score
    //close the file
    //delete the structure name arrays
    //delete the player structure array
//Exit Program
```

main

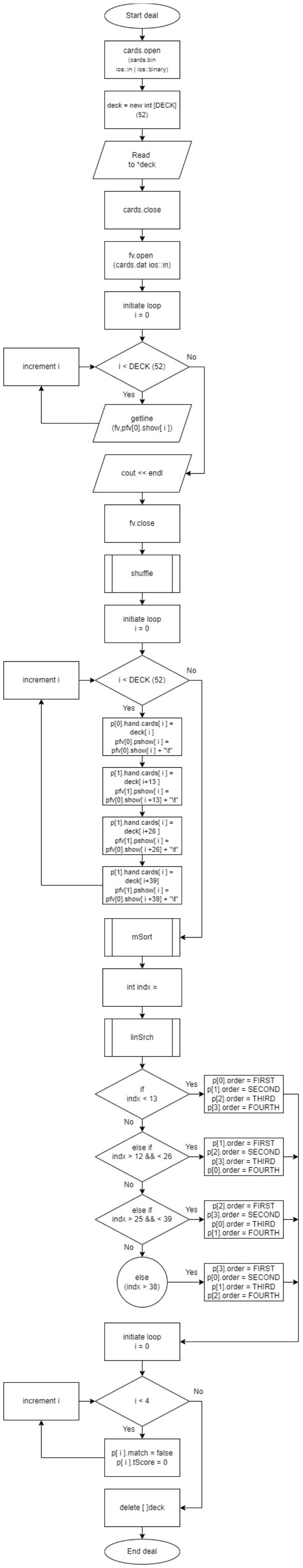


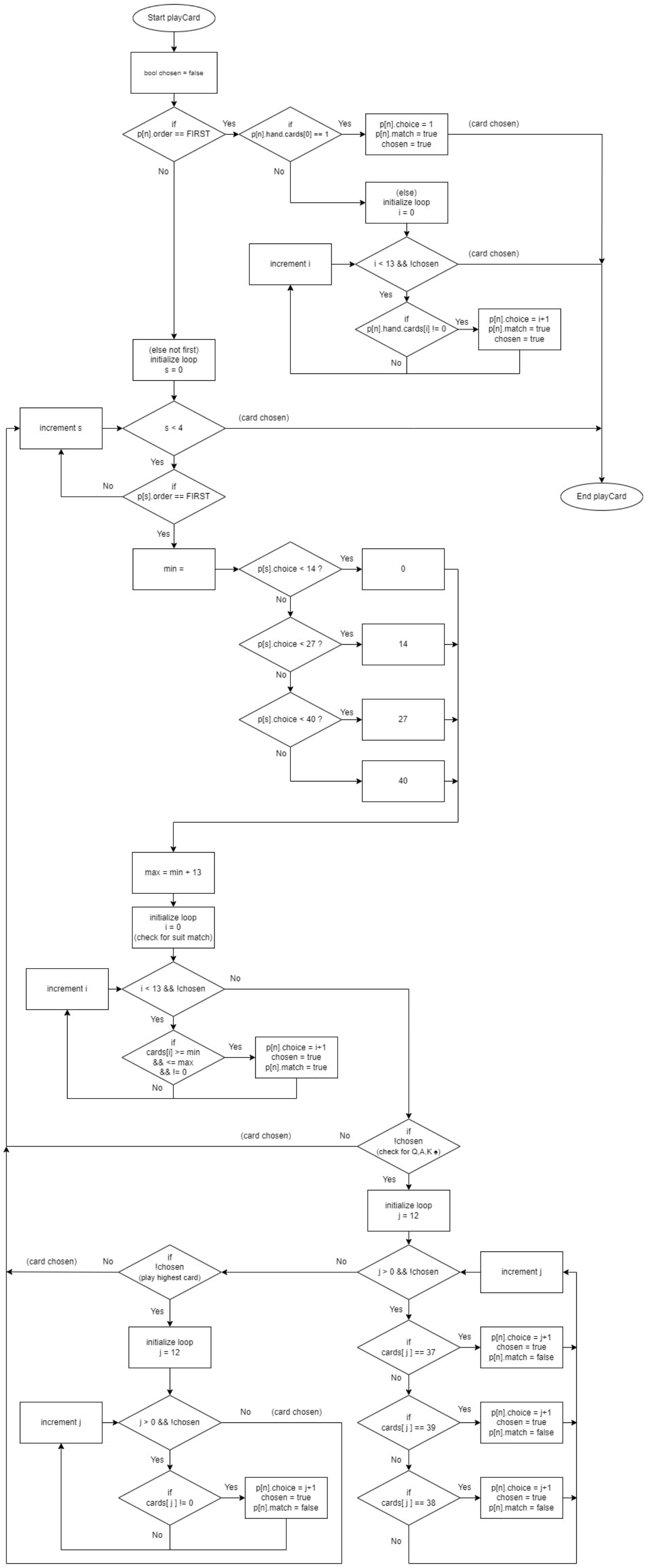
Start main
srand
initialize variables
allocate dynamic arrays for each stooge p[ ].name
strcpy each Stooge name into their name array
set all p.scores to 0
cout intro message
create string input
getline(cin,input)
allocate dynamic memory for p[0].name input.length()+1
cout welcome p[0].name
do { while(scores < 50)
No
Yes
deal
initiate for loop i = 0
increment i
i < 13
No
Yes
play
trick
initiate for loop n = 0
increment n
n < 13
No
Yes
unSet(p[n], pfv[n])
cout endl
if p[0].tScore == 26
Yes
p[0].tScore = 0 p[1-3].tScore = 26
cout Player shot the moon
No
if p[1].tScore == 26
Yes
p[1].tScore = 0 p[2,3,0].tScore = 26
cout Larry shot the moon
No
if p[2].tScore == 26
Yes
p[2].tScore = 0 p[3,0,1].tScore = 26
cout Curly shot the moon
No
(else) cout What a fun hand
initiate for loop i = 0
increment i
i < 4
Yes
p[i].score += p[i].tScore
No
cout current game score
cout endl
int winner int min = 150
initiate for loop i = 0
increment i
i < 4
No
Yes
if p[i].score < min
No
Yes
min = p[i].score winner = i
if winner == 0
Yes
No
cout You won! endl
cout You lost! endl
open saved.bin in | out | binary
declare Win w[8]
move read to start of file
cout Previous Scores
initiate for loop i = 0
increment i
i < 4
No
Yes
read in next winner
cout winner name & score
initiate for loop i = 0
increment i
i < 4
No
Yes
strcpy(w[i].name, p[i].name)
w[i].score = p[i].score
mSort
cout Current Scores
initiate for loop i = 0
i < 4
increment i
Yes
No
cout winner name & score
move write to beginning of file
initiate for loop i = 0
i < 4
increment i
Yes
No
write current winner to file
move write to beginning of file
Wins n[4]
cout Saved Best Scores
initiate for loop i = 0
i < 4
increment i
Yes
No
read winner to n[i]
cout n[i] name & score
save.close
delete[]p[0].name delete[]p[1].name delete[]p[2].name delete[]p[3].name
delete []p
End main

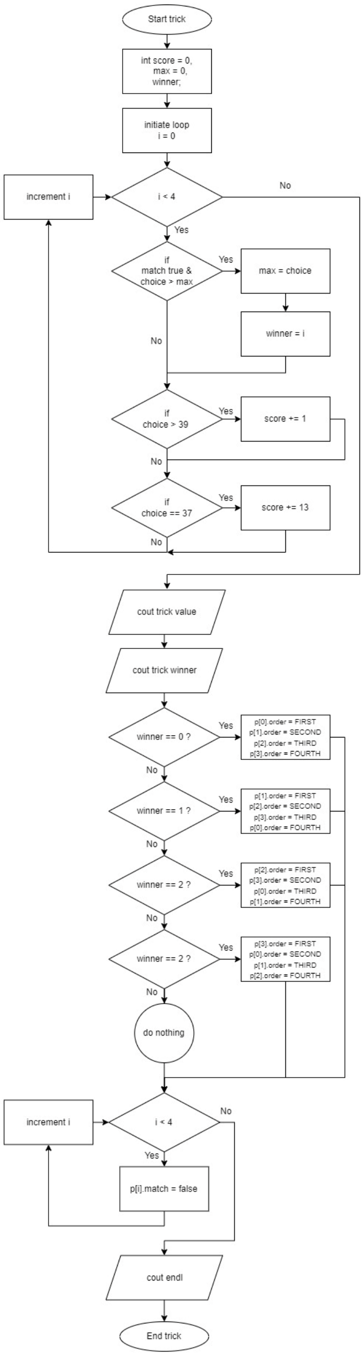
play , linSrch, shuffle



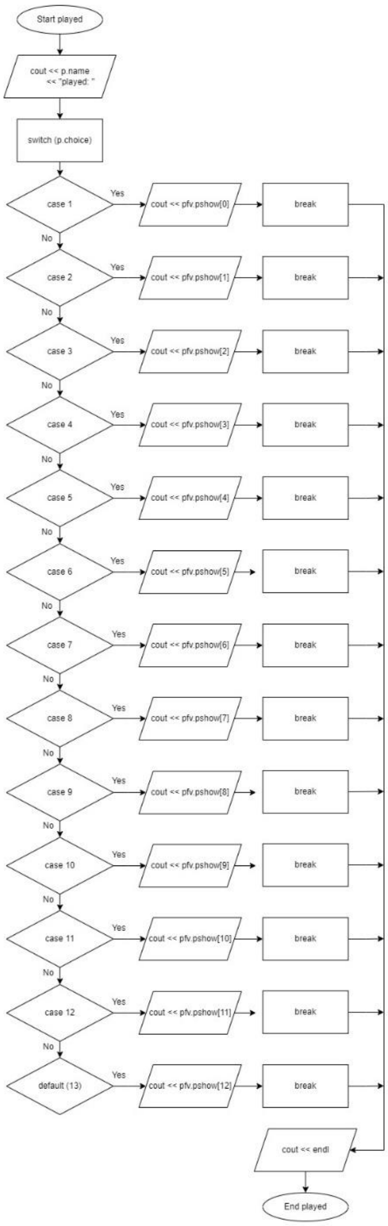
deal, playCard, trick

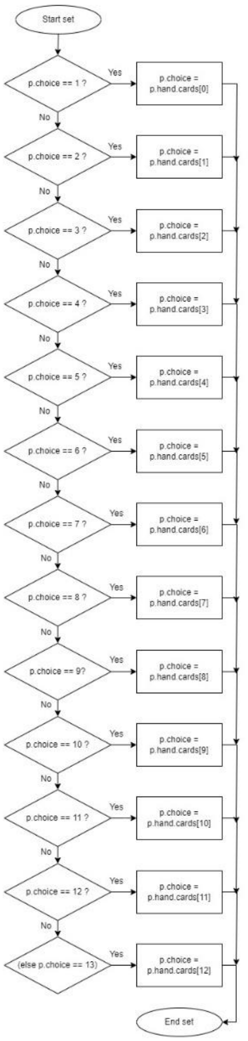

Start deal
cards.open (cards.bin ios::in | ios::binary)
deck = new int [DECK] (52)
Read to *deck
cards.close
fv.open (cards.dat ios::in)
initiate loop i = 0
increment i
i < DECK (52)
No
Yes
getline (fv,pfv[0].show[i])
cout << endl
fv.close
shuffle
initiate loop i = 0
increment i
i < DECK (52)
No
Yes
p[0].hand.cards[i] = deck[i] pfv[0].pshow[i] = pfv[0].show[i] + "\t"
p[1].hand.cards[i] = deck[i+13] pfv[1].pshow[i] = pfv[0].show[i +13] + "\t"
p[1].hand.cards[i] = deck[i+26] pfv[1].pshow[i] = pfv[0].show[i +26] + "\t"
p[1].hand.cards[i] = deck[i+39] pfv[1].pshow[i] = pfv[0].show[i +39] + "\t"
mSort
int indx =
linSrch
if indx < 13
Yes
p[0].order = FIRST p[1].order = SECOND p[2].order = THIRD p[3].order = FOURTH
No
else if indx > 12 && < 26
Yes
p[1].order = FIRST p[2].order = SECOND p[3].order = THIRD p[0].order = FOURTH
No
else if indx > 25 && < 39
Yes
p[2].order = FIRST p[3].order = SECOND p[0].order = THIRD p[1].order = FOURTH
No
else (indx > 38)
Yes
p[3].order = FIRST p[0].order = SECOND p[1].order = THIRD p[2].order = FOURTH
initiate loop i = 0
increment i
i < 4
No
Yes
p[i].match = false p[i].tScore = 0
delete [ ]deck
End deal

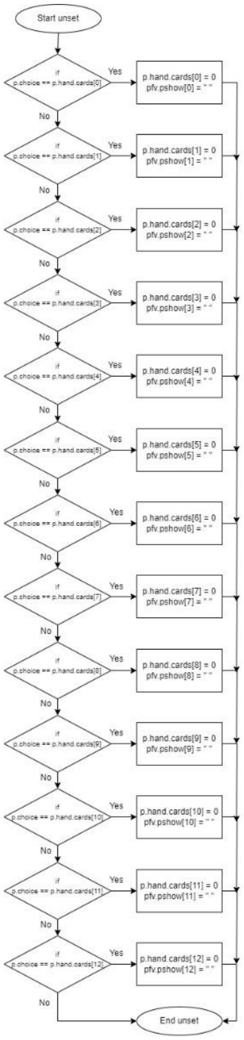

Start playCard
bool chosen = false
if p[n].order == FIRST
Yes
if p[n].hand.cards[0] == 1
Yes
p[n].choice = 1 p[n].match = true chosen = true
(card chosen)
No
(else) initialize loop i = 0
increment i
i < 13 && !chosen
(card chosen)
Yes
if p[n].hand.cards[i] != 0
Yes
p[n].choice = i+1 p[n].match = true chosen = true
No
No
(else not first) initialize loop s = 0
increment s
s < 4
(card chosen)
End playCard
Yes
if p[s].order == FIRST
No
Yes
min =
p[s].choice < 14 ?
Yes
0
No
p[s].choice < 27 ?
Yes
14
No
p[s].choice < 40 ?
Yes
27
No
40
max = min + 13
initialize loop i = 0 (check for suit match)
increment i
i < 13 && !chosen
No
Yes
if cards[i] >= min && <= max && != 0
Yes
p[n].choice = i+1 chosen = true p[n].match = true
No
(card chosen)
No
if !chosen (check for Q,A,K ♠)
Yes
initialize loop j = 12
(card chosen)
No
if !chosen (play highest card)
No
j > 0 && !chosen
increment j
Yes
if cards[j] == 37
Yes
p[n].choice = j+1 chosen = true p[n].match = false
No
if cards[j] == 39
Yes
p[n].choice = j+1 chosen = true p[n].match = false
No
if cards[j] == 38
Yes
p[n].choice = j+1 chosen = true p[n].match = false
No
Yes
initialize loop j = 12
increment j
j > 0 && !chosen
No
(card chosen)
Yes
if cards[j] != 0
Yes
p[n].choice = j+1 chosen = true p[n].match = false
No

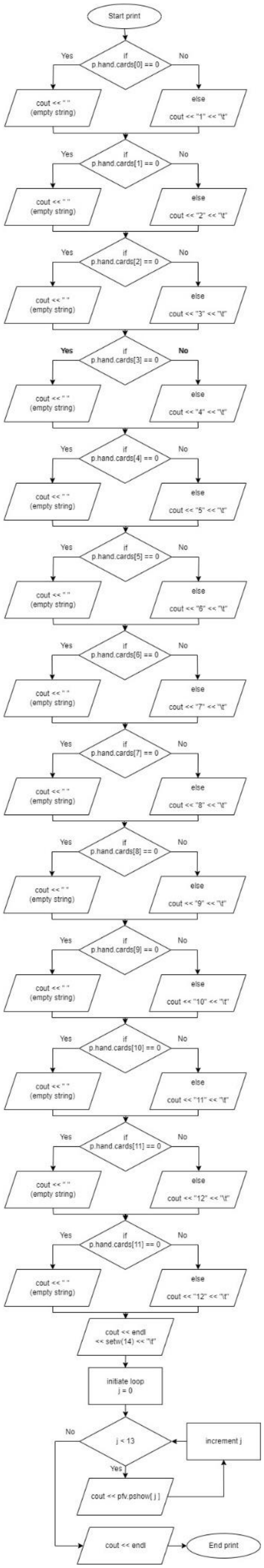

Start trick
int score = 0, max = 0, winner;
initiate loop i = 0
increment i
i < 4
No
Yes
if match true & choice > max
Yes
max = choice
winner = i
No
if choice > 39
Yes
score += 1
No
if choice == 37
Yes
score += 13
No
cout trick value
cout trick winner
winner == 0 ?
Yes
p[0].order = FIRST p[1].order = SECOND p[2].order = THIRD p[3].order = FOURTH
No
winner == 1 ?
Yes
p[1].order = FIRST p[2].order = SECOND p[3].order = THIRD p[0].order = FOURTH
No
winner == 2 ?
Yes
p[2].order = FIRST p[3].order = SECOND p[0].order = THIRD p[1].order = FOURTH
No
winner == 2 ?
Yes
p[3].order = FIRST p[0].order = SECOND p[1].order = THIRD p[2].order = FOURTH
No
do nothing
increment i
i < 4
No
Yes
p[i].match = false
cout endl
End trick

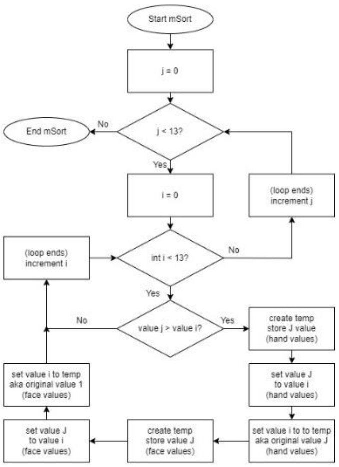
Played, set, unset, print, mSort (p,pfv version only)




Start set
p.choice == 1 ?
Yes
p.choice = p.hand.cards[0]
No
p.choice == 2 ?
Yes
p.choice = p.hand.cards[1]
No
p.choice == 3 ?
Yes
p.choice = p.hand.cards[2]
No
p.choice == 4 ?
Yes
p.choice = p.hand.cards[3]
No
p.choice == 5 ?
Yes
p.choice = p.hand.cards[4]
No
p.choice == 6 ?
Yes
p.choice = p.hand.cards[5]
No
p.choice == 7 ?
Yes
p.choice = p.hand.cards[6]
No
p.choice == 8 ?
Yes
p.choice = p.hand.cards[7]
No
p.choice == 9?
Yes
p.choice = p.hand.cards[8]
No
p.choice == 10 ?
Yes
p.choice = p.hand.cards[9]
No
p.choice == 11 ?
Yes
p.choice = p.hand.cards[10]
No
p.choice == 12 ?
Yes
p.choice = p.hand.cards[11]
No
(else p.choice == 13)
Yes
p.choice = p.hand.cards[12]
End set


Start unset
if p.choice == p.hand.cards[0]
Yes
p.hand.cards[0] = 0 pfv.pshow[0] = " "
No
if p.choice == p.hand.cards[1]
Yes
p.hand.cards[1] = 0 pfv.pshow[1] = " "
No
if p.choice == p.hand.cards[2]
Yes
p.hand.cards[2] = 0 pfv.pshow[2] = " "
No
if p.choice == p.hand.cards[3]
Yes
p.hand.cards[3] = 0 pfv.pshow[3] = " "
No
if p.choice == p.hand.cards[4]
Yes
p.hand.cards[4] = 0 pfv.pshow[4] = " "
No
if p.choice == p.hand.cards[5]
Yes
p.hand.cards[5] = 0 pfv.pshow[5] = " "
No
if p.choice == p.hand.cards[6]
Yes
p.hand.cards[6] = 0 pfv.pshow[6] = " "
No
if p.choice == p.hand.cards[7]
Yes
p.hand.cards[7] = 0 pfv.pshow[7] = " "
No
if p.choice == p.hand.cards[8]
Yes
p.hand.cards[8] = 0 pfv.pshow[8] = " "
No
if p.choice == p.hand.cards[9]
Yes
p.hand.cards[9] = 0 pfv.pshow[9] = " "
No
if p.choice == p.hand.cards[10]
Yes
p.hand.cards[10] = 0 pfv.pshow[10] = " "
No
if p.choice == p.hand.cards[11]
Yes
p.hand.cards[11] = 0 pfv.pshow[11] = " "
No
if p.choice == p.hand.cards[12]
Yes
p.hand.cards[12] = 0 pfv.pshow[12] = " "
No
End unset




Start mSort
j = 0
j < 13?
No
End mSort
Yes
i = 0
int i < 13?
No
(loop ends) increment j
Yes
value j > value i?
Yes
create temp store J value (hand values)
set value J to value i (hand values)
set value i to to temp aka original value J (hand values)
create temp store value J (face values)
set value J to value i (face values)
set value i to temp aka original value 1 (face values)
No
(loop ends) increment i

```
/*
 * File:   main.cpp
 * Author: Stephanie Peacock
 * Created on October  24, 2023, 6:45 PM
 * Purpose: Play a game of Hearts vs the Three Stooges.
 */

//System Libraries
#include <iostream>  // Input / Output Library  : cin, cout
#include <cstdlib>   // Random Function Library : srand()
#include <ctime>     // Time Library            : time()
#include <fstream>   // File Library            : open(), close()
#include <cstring>   // Cstring Library         : strcpy(),
#include <iomanip>   // Formatting library      : setw()

using namespace std;

//User Libraries
#include "hearts.h"  // Hearts specific library
//Global Constants - Math/Physics/Chemistry/Conversions Only

//Function Prototypes - included in hearts.h

//Execution Begins Here
int main(int argc, char** argv) {
    //Setting Random Number Seed
    srand(static_cast<unsigned int>(time(0)));
    //Declare all variables here

    //Initialize variables/input values
    fstream save,           //saved.bin - stream for save file
            cards,          //cards.bin - initial card values file
            fv;             //cards.dat - for storing face values
    int *deck;              //pointer to store the card values
    Player *p;              //4 player structures, for each player
    Show pfv[4];            //4 string structures, for each player

    //Create the structures
    p = new Player[4];
    //Make the NPC name arrays and fill them
    p[1].name = new char[6];p[2].name = new char[6];p[3].name = new char[4];
    strcpy(p[1].name,"Larry"); strcpy(p[2].name,"Curly"); strcpy(p[3].name,"Moe");
    //set all game scores to 0
    p[0].score = p[1].score = p[2].score = p[3].score = 0;

    //INTRO
    cout << "Hey buddy, we need a fourth player!" << endl
         << "Let's play some Hearts!" << endl
         << "Would you like to resume a saved game?" << endl
         << "Enter y to load saved game or n to start a new game." << endl;
//    cin >> saved;
//    if(saved == 'y' || saved == 'Y'){

//    }else{
    cout << "My name is Larry, this is my good friend Curly," << endl
         << "and that's his brother, Moe." << endl << endl
         << "So what's your name?" << endl;
    string input;
    getline(cin,input);
    p[0].name = new char[input.length()+1];
    strcpy(p[0].name,input.c_str());
    cout << "Alright " << p[0].name << ", let's play Hearts! I'll deal." << endl;
//    }
    //make sure we ignore the newline char
    //cin.ignore();


    //start the game loop
    do{
        //Deal the cards
        deal(cards,fv,deck,p,DECK,pfv);

        //Hand loop
        for(int i = 0; i < 13; i++){
            //Play 13 tricks
            play(p,pfv);
            //Score the trick
            trick(p);
            //Unset the cards so they can't be replayed
            for(int n = 0; n < 4; n++){
                unset(p[n],pfv[n]);
            }
            cout << "**************************************************" << endl;
        }
        // Check for Shooting the Moon
        p[0].tScore == 26 ? p[0].tScore = 0, p[1].tScore = 26, p[2].tScore = 26, p[3].tScore = 26,cout << "Hey, Wiseguy! You shot the moon! The Stooges takes all points." << endl << endl :
        p[1].tScore == 26 ? p[1].tScore = 0, p[2].tScore = 26, p[3].tScore = 26, p[0].tScore = 26, cout << "Larry shot the moon! The rest of you schmucks can take the points."  << endl << endl :
        p[2].tScore == 26 ? p[2].tScore = 0, p[3].tScore = 26, p[0].tScore = 26, p[1].tScore = 26, cout << "Curly shot the moon! The rest of you schmucks can take the points."  << endl << endl :
        p[3].tScore == 26 ? p[3].tScore = 0, p[0].tScore = 26, p[1].tScore = 26, p[2].tScore = 26, cout << "Moe shot the moon! The rest of you schmucks can take the points."  << endl << endl :
        cout << endl << "What a fun hand!" << endl << endl;
        //add tscores to score
        for(int n = 0; n < 4; n++){
            p[n].score += p[n].tScore;
        }
        cout << "The current game score is:" << endl<< p[0].name << " : " << p[0].score << endl;
        cout << "Larry : " << p[1].score << endl;
        cout << "Curly : " << p[2].score << endl;
        cout << "Moe   : " << p[3].score << endl;
    }while(p[0].score < 50 && p[1].score < 50 && p[2].score < 50 && p[3].score < 50);
    cout << endl;
    //Find the winner
    int winner;
    int min = 150;
    for(int i = 0; i < 4; i++){
        if(p[i].score < min){
            min = p[i].score;
            winner = i;
        }
    }
    if (winner == 0) { cout << "You won the game!" << endl; }
    else { cout << "You lost the game. Better luck next time!" << endl; }
    //open the file
    save.open("saved.bin", ios::out | ios::in | ios::binary);   //read in saved file
    //create the structures - 1 extra for current winner
    Win w[8];
    //read in the file - make sure we start at the beginning
    save.seekg(0);
    //output the old best scores
    cout << "Previous Best Scores: " << endl;
    for(int i = 0; i < 4; i++){
        //get the next winner
        save.read(reinterpret_cast<char*>(&w[i]), 1 * sizeof(Win));
        cout << i+1 << ": " << setw(20) << left << w[i].name << "\t" << w[i].score << endl;
    }
    //copy all 4 current game scores to winner structure
    for(int i = 4; i < 8; i++){
        strcpy(w[i].name,p[i-4].name);
        w[i].score = p[i-4].score;
    }

    //sort all 5
    mSort(w);
    //output new best scores
    cout << "Current Best Scores: " << endl;
    for(int i = 0; i < 4; i++){
        cout << i+1 << ": " << setw(20) << left << w[i].name << "\t" << w[i].score << endl;
    }
    //write the 4 best to file
    save.seekp(0);
    for(int i = 0; i < 4; i++){
        save.write(reinterpret_cast<char*>(&w[i]), sizeof(Win));
    }
    //double check our write
    save.seekg(0);
    //output the old best scores
    Win n[4];
    cout << "Saved Best Scores: " << endl;
    for(int i = 0; i < 4; i++){
        save.read(reinterpret_cast<char*>(&n[i]), sizeof(Win));
        cout << i+1 << ": " << setw(20) << left << n[i].name << "\t" << n[i].score << endl;
    }
    //close the file
    save.close();



    delete []p[0].name;
    delete []p[1].name;
    delete []p[2].name;
    delete []p[3].name;
    delete []p;
    //Exit Program
    return 0;
}
//Deal to each player - deck array & all players
void deal(fstream &cards, fstream &fv, int *deck, Player *p, const int DECK, Show *pfv){
    //read in fresh cards
    cards.open("cards.bin", ios::in | ios::binary);
    deck = new int[DECK];
    cards.read(reinterpret_cast<char*>(deck), DECK * sizeof(int));
    //close the file
    cards.close();
    //GET THE FACE VALUES OF THE DECK
    fv.open("cards.dat", ios::in);
    //just the player 1 values will be used
    for(int i = 0; i < DECK; i++){
       getline(fv,pfv[0].show[i]);
    }
    cout << endl;
    //close the file
    fv.close();
    //shuffle the deck
    shuffle(deck,DECK,pfv); //shuffle the cards
    //pass out the cards
    for(int i = 0; i < 13; i++) {   //deal out the cards & face values
        p[0].hand.cards[i] =  deck[i];      pfv[0].pshow[i] = pfv[0].show[i] + "\t";
        p[1].hand.cards[i] =  deck[(i+13)]; pfv[1].pshow[i] = pfv[0].show[i+13] + "\t";
        p[2].hand.cards[i] =  deck[(i+26)]; pfv[2].pshow[i] = pfv[0].show[i+26] + "\t";
        p[3].hand.cards[i] =  deck[(i+39)]; pfv[3].pshow[i] = pfv[0].show[i+39] + "\t";
    }
    //sort all the hands
    mSort(p,pfv);
    //find the player order
    int indx = linSrch(deck,DECK);
    if     (indx < 13)              { p[0].order = FIRST; p[1].order = SECOND; p[2].order = THIRD; p[3].order = FOURTH; }
    else if(indx > 12 && indx < 26) { p[1].order = FIRST; p[2].order = SECOND; p[3].order = THIRD; p[0].order =
FOURTH; }
    else if(indx > 25 && indx < 39) { p[2].order = FIRST; p[3].order = SECOND; p[0].order = THIRD; p[1].order =
FOURTH; }
    else                            { p[3].order = FIRST; p[0].order = SECOND; p[1].order = THIRD; p[2].order = FOURTH; }
    //reset the match status & tScores
    for(int i = 0; i < 4; i++){
        p[i].match = false;
        p[i].tScore = 0;
    }
    //Delete the deck array since it's not needed until redeal
    delete[] deck;
}

//Shuffle the deck array (nested inside deal)
void shuffle(int *deck, const int DECK, Show *pfv){
    for (int i = 0; i < DECK; i++) {
        int n = i + (rand()%(52 - i));
        swap(deck[i], deck[n]);
        swap(pfv[0].show[i],pfv[0].show[n]);
    }
}

//Find 2 of clubs  (nested inside deal) - deck array
int  linSrch(int *deck, const int DECK){
    int indx = 0;
    // start off with false
    bool found = false;
    // run until the val is found or we run through all the numbers
    while (indx < DECK && !found){
        // if the val is found it returns true and stops the loop, if false it adds to the count
        deck[indx] == 1 ?  found = true : indx++;
    }
    // sends back what index 2 of clubs (1) was found at
    return indx;
}

//Sort each player's hand (nested inside deal)
void mSort(Player *p, Show *pfv){
    //sort all four players
    for(int n = 0; n < 4; n++){
        for(int j = 0; j < 13; j++){
            for(int i = j + 1; i < 13; i++){
                if(p[n].hand.cards[j] > p[n].hand.cards[i]){
                    //sort the player int values
                    int temp = p[n].hand.cards[j];
                    p[n].hand.cards[j] = p[n].hand.cards[i];
                    p[n].hand.cards[i] = temp;
                    //now sort the corresponding show values
                    string ts = pfv[n].pshow[j];
                    pfv[n].pshow[j] = pfv[n].pshow[i];
                    pfv[n].pshow[i] = ts;
                }
            }
        }
    }
}

//Print out face values of player's hand
void print(Player &p, Show &pfv){
    // output Player's remaining cards - if value is 0 an empty string is output
    cout << endl << "Play Card #:\t";
    if (p.hand.cards[0] == 0) { cout << ""; } else { cout << "1" << "\t"; }
    if (p.hand.cards[1] == 0) { cout << ""; } else { cout << "2" << "\t"; }
    if (p.hand.cards[2] == 0) { cout << ""; } else { cout << "3" << "\t"; }
    if (p.hand.cards[3] == 0) { cout << ""; } else { cout << "4" << "\t"; }
    if (p.hand.cards[4] == 0) { cout << ""; } else { cout << "5" << "\t"; }
    if (p.hand.cards[5] == 0) { cout << ""; } else { cout << "6" << "\t"; }
    if (p.hand.cards[6] == 0) { cout << ""; } else { cout << "7" << "\t"; }
    if (p.hand.cards[7] == 0) { cout << ""; } else { cout << "8" << "\t"; }
    if (p.hand.cards[8] == 0) { cout << ""; } else { cout << "9" << "\t"; }
    if (p.hand.cards[9] == 0) { cout << ""; } else { cout << "10"<< "\t"; }
    if (p.hand.cards[10]== 0) { cout << ""; } else { cout << "11"<< "\t"; }
    if (p.hand.cards[11]== 0) { cout << ""; } else { cout << "12"<< "\t"; }
    if (p.hand.cards[12]== 0) { cout << ""; } else { cout << "13"<< "\t"; }
    // now output the remaining face values
    cout << endl << setw(14) << "\t";
    for (int j = 0; j < 13; j++) {
        cout << pfv.pshow[j];
    }
    cout << endl;
}

//Play the trick
void play(Player *p, Show *pfv){
    //set min higher than possible - used to find valid range
    int min = 53;
    //loop 4 times so each player goes
    for(int trick = 0; trick < 4; trick++){
        //Player's Turn
        if (p[0].order == trick) {
```

```
// Print player's cards
print(p[0], pfv[0]);
//set valid to false
bool valid = false;
//run until valid choice is found
do {
    cout << "Choose a card in your hand you wish to play: ";
    cin >> p[0].choice;
    // Add 2clubs validation if player is first
    while (p[0].order == FIRST && p[0].hand.cards[0] == 1 && p[0].choice != 1) {
        //player has 2 clubs and didnt play it
        cout << "Please play 2\u2663: ";
        cin >> p[0].choice;
        //player plays 2 clubs
        if(p[0].choice == 1){
            //choice is valid
            valid = true;
            //first player always matches
            p[0].match = true;
        }
    }
    //no 2 clubs, player is first
    if(p[0].order == FIRST && p[0].hand.cards[p[0].choice-1] != 0){
        //first player card always valid
        valid = true;
        //first player always matches
        p[0].match = true;
    }

    // Add suit validation if not first player
    // Check for Clubs
    if((p[1].order == FIRST && p[1].choice <= 13) ||
       (p[2].order == FIRST && p[2].choice <= 13) ||
       (p[3].order == FIRST && p[3].choice <= 13)) {
        //choice is valid match
        if (p[0].hand.cards[p[0].choice-1] <= 13 && p[0].hand.cards[p[0].choice-1] != 0){
            valid = true;
            p[0].match = true;
        }
        //loop through all the cards to get the min
        for (int n = 0; n < 13; n++){
            if(p[0].hand.cards[n] < min){
                min = p[0].hand.cards[n];
            }
        }
        // else if there is a valid match that wasn't played
        if(min <= 13 && min != 0){
            while(p[0].hand.cards[p[0].choice-1] > 13 && p[0].hand.cards[p[0].choice-1] != 0){
                //prompt for new choice
                cout << "Please play \u2663: ";
                        cin >> p[0].choice;
                //recheck if choice is valid match
                if (p[0].hand.cards[p[0].choice-1] <= 13 && p[0].hand.cards[p[0].choice-1] != 0){
                    valid = true;
                    p[0].match = true;
                }
            }
        }
    else if (min > 13 && p[0].hand.cards[p[0].choice-1] != 0){
        //no matching cards, so any card will do
        valid = true;
        //but we didn't match
        p[0].match = false;
    }
}
//now check for Diamonds
else if((p[1].order == FIRST && (p[1].choice >= 14 && p[1].choice <= 26)) ||
        (p[2].order == FIRST && (p[2].choice >= 14 && p[2].choice <= 26)) ||
        (p[3].order == FIRST && (p[3].choice >= 14 && p[3].choice <= 26))) {
    //reset min to original value
    min = 53;
    //choice is valid match
    if (p[0].hand.cards[p[0].choice-1] >= 14 && p[0].hand.cards[p[0].choice-1] <= 26){
        valid = true;
        p[0].match = true;
    }
    //loop through all the cards to get the min & max
    for (int n = 0; n < 13; n++){
        if(p[0].hand.cards[n] >= 14 && p[0].hand.cards[n] <= 26){
            min = p[0].hand.cards[n];
        }
    }
    // else if there is a valid match that wasn't played
    if(min >= 14 && min <= 26){
        while(p[0].hand.cards[p[0].choice-1] < 14 || p[0].hand.cards[p[0].choice-1] > 26){
            //prompt for new choice
            cout << "Please play \u2662: ";
                    cin >> p[0].choice;
            //recheck if choice is valid match
            if (p[0].hand.cards[p[0].choice-1] >= 14 && p[0].hand.cards[p[0].choice-1] <= 26){
                valid = true;
                p[0].match = true;
            }
        }
    }
    else if (min > 26){
        //no matching cards, so any card will do
        valid = true;
        //but we didn't match
        p[0].match = false;
    }
}

//now check for Spades
else if((p[1].order == FIRST && (p[1].choice >= 27 && p[1].choice <= 39)) ||
        (p[2].order == FIRST && (p[2].choice >= 27 && p[2].choice <= 39)) ||
        (p[3].order == FIRST && (p[3].choice >= 27 && p[3].choice <= 39))) {
    //reset min again
    min = 53;
    //choice is valid match
    if (p[0].hand.cards[p[0].choice-1] >= 27 && p[0].hand.cards[p[0].choice-1] <= 39){
        valid = true;
        p[0].match = true;
    }
    //loop through all the cards to get the min & max
    for (int n = 0; n < 13; n++){
        if(p[0].hand.cards[n] >= 27 && p[0].hand.cards[n] <= 39){
            min = p[0].hand.cards[n];
        }
    }
    // else if there is a valid match that wasn't played
    if(min <= 39){
        while(p[0].hand.cards[p[0].choice-1] < 27 || p[0].hand.cards[p[0].choice-1] > 39){
            //prompt for new choice
            cout << "Please play \u2660: ";
                    cin >> p[0].choice;
            //recheck if choice is valid match
            if (p[0].hand.cards[p[0].choice-1] >= 27 && p[0].hand.cards[p[0].choice-1] <= 39){
                valid = true;
                p[0].match = true;
            }
        }
    }
    else if (min > 39){
        //no matching cards, so any card will do
        valid = true;
        //but we didn't match
        p[0].match = false;
    }
}
//last check for Hearts
else if((p[1].order == FIRST && p[1].choice >= 40) ||
        (p[2].order == FIRST && p[2].choice >= 40) ||
        (p[3].order == FIRST && p[3].choice >= 40)) {
    //reset min (it's actually max now)
    min = 1;
    //choice is valid match
    if (p[0].hand.cards[p[0].choice-1] >= 40){
        valid = true;
        p[0].match = true;
    }
    //loop through all the cards to get the min & max
    for (int n = 0; n < 13; n++){
        if(p[0].hand.cards[n] >= 40){
            min = p[0].hand.cards[n];
        }
    }
    // else if there is a valid match that wasn't played
    if(min > 39){
        while(p[0].hand.cards[p[0].choice-1] < 40){
            //prompt for new choice
            cout << "Please play \u2661: ";
                    cin >> p[0].choice;
            //recheck if choice is valid match
            if (p[0].hand.cards[p[0].choice-1] >= 40){
                valid = true;
                p[0].match = true;
            }
        }
    }
    else if (min < 40){
        //no matching cards, so any card will do
        valid = true;
        //but we didn't match
```

```
                        p[0].match = false;
                    }
                }
            } while ((p[0].choice < 1 || p[0].choice > 13) && p[0].hand.cards[p[0].choice-1] != 0 && !valid);
        // Player played
        played(p[0],pfv[0]);
        //Player set
        set(p[0]);
        }
        // Larry's Turn
        else if (p[1].order == trick) {
        //    cout << "Larry's cards";
            //print(p[1],pfv[1]);
            playCard(p,1);
            // Output the card played
            played(p[1], pfv[1]);
            // Set the played card for scoring
            set(p[1]);
        }
        // Curly's Turn
        else if (p[2].order == trick) {
        //    cout << "Curly's cards";
            //print(p[2],pfv[2]);
            playCard(p,2);
            // Output the card played
            played(p[2], pfv[2]);
            // Set the played card for scoring
            set(p[2]);
        }
        // Moe's Turn
        else if (p[3].order == trick) {
        //    cout << "Curly's cards";
            //print(p[3],pfv[3]);
            playCard(p,3);
            // Output the card played
            played(p[3], pfv[3]);
            // Set the played card for scoring
            set(p[3]);
        }
    }
}

//Stooge picks a card
void playCard(Player *p, int n){
    bool chosen = false;
    // If npc is first to play in the trick - check for 2 clubs first
    if(p[n].order == FIRST) {
        if ( p[n].hand.cards[0] == 1) {p[n].choice = 1; p[n].match = true; chosen = true;}
        // If no 2 clubs, play lowest card
        else {
            for(int i = 0; i < 13 && !chosen; i++){
                if(p[n].hand.cards[i] != 0) { p[n].choice = i+1; chosen = true; p[n].match = true; }
            }
        }
    }
    else {
        //Check against all 4 players
        for(int s = 0; s < 4; s++){
            if(p[s].order == FIRST) {
                int min = p[s].choice < 14 ? 0 : p[s].choice < 27 ? 14 : p[s].choice < 40 ? 27 : 40;
                int max = min + 13;
            // p[n].match the suit if possible - playing lowest card possible
                for (int i = 0; i < 13 && !chosen; i++) {
                    if (p[n].hand.cards[i] >= min && p[n].hand.cards[i] <= max && p[n].hand.cards[i] != 0) {
                        p[n].choice = (i+1); chosen = true; p[n].match = true; }
                }
            // Play Q spades, A spades, or K spades first if can't p[n].match suit
                if(!chosen){
                    for (int j = 12; j > 0 && !chosen; j--) {
                        if (p[n].hand.cards[j] == 37) {p[n].choice = (j+1); chosen = true; p[n].match = false;}
                        if (p[n].hand.cards[j] == 39) {p[n].choice = (j+1); chosen = true; p[n].match = false;}
                        if (p[n].hand.cards[j] == 38) {p[n].choice = (j+1); chosen = true; p[n].match = false;}
                    }
                }
            // Lastly, play the highest card available
                if(!chosen){
                    for (int j = 12; j > 0 && !chosen; j--) {
                        if (p[n].hand.cards[j] != 0) {p[n].choice = (j+1); chosen = true; p[n].match = false; }
                    }
                }
            }
        }
    }
}

//Print out the cards played (nested inside playCard)
void played(Player &p, Show &pfv){
    cout << p.name << " played: ";
    switch (p.choice) {
        case 1: cout << pfv.pshow[0]; break;
        case 2: cout << pfv.pshow[1]; break;
        case 3: cout << pfv.pshow[2]; break;
        case 4: cout << pfv.pshow[3]; break;
        case 5: cout << pfv.pshow[4]; break;
        case 6: cout << pfv.pshow[5]; break;
        case 7: cout << pfv.pshow[6]; break;
        case 8: cout << pfv.pshow[7]; break;
        case 9: cout << pfv.pshow[8]; break;
        case 10:cout << pfv.pshow[9]; break;
        case 11:cout << pfv.pshow[10];break;
        case 12:cout << pfv.pshow[11];break;
        default:cout << pfv.pshow[12];break;
    }
    cout << endl;
}

//Set the player's choice to card value (nested inside playCard)
void set(Player &p){
// Set p.choice to card value for scoring
    p.choice == 1  ? p.choice = p.hand.cards[0] : p.choice == 2  ? p.choice = p.hand.cards[1] :
    p.choice == 3  ? p.choice = p.hand.cards[2] : p.choice == 4  ? p.choice = p.hand.cards[3] :
    p.choice == 5  ? p.choice = p.hand.cards[4] : p.choice == 6  ? p.choice = p.hand.cards[5] :
    p.choice == 7  ? p.choice = p.hand.cards[6] : p.choice == 8  ? p.choice = p.hand.cards[7] :
    p.choice == 9  ? p.choice = p.hand.cards[8] : p.choice == 10 ? p.choice = p.hand.cards[9] :
    p.choice == 11 ? p.choice = p.hand.cards[10]: p.choice == 12 ? p.choice = p.hand.cards[11]:
    p.choice = p.hand.cards[12];
```

```
}

//Score the trick
void trick( Player *p){
    //local counters
    int score = 0,  //holds the trick points
        max = 0,    //hold the highest card value
        winner;     //holds the winner's location
    //loop through all four players
    for(int i = 0; i < 4; i++){
        //if the player matched the suit and their choice is highest
        if(p[i].match == true && p[i].choice > max){
            //set their choice to the max
            max = p[i].choice;
            //set them as winner
            winner = i;
        }
        //now see if there was a heart played
        if(p[i].choice > 39){
            //add 1 to the trick value
            score += 1;
        }
        //last check if the Q Spades was played
        if(p[i].choice == 37){
            score += 13;
        }
    }
    //output the trick value
    cout << endl << "Trick is worth " << score << " points. ";
    //output the winner
    cout << p[winner].name << " takes the trick." << endl << endl;
    //set the new order - player was winner?
    winner == 0 ? p[0].order = FIRST,p[1].order = SECOND,p[2].order = THIRD,p[3].order = FOURTH :
    //larry was winner?
    winner == 1 ? p[1].order = FIRST,p[2].order = SECOND,p[3].order = THIRD,p[0].order = FOURTH :
    //curly was winner?
    winner == 2 ? p[2].order = FIRST,p[3].order = SECOND,p[0].order = THIRD,p[1].order = FOURTH :
    //else moe was the winner
    winner == 3 ? p[3].order = FIRST,p[0].order = SECOND,p[1].order = THIRD,p[2].order = FOURTH : 0;
    //assign the points to the winner
    p[winner].tScore += score;
    //check all 4 players
    for(int i = 0; i < 4; i++){
        //reset match for all
        p[i].match = false;
        //for testing - output all hand values
        cout << p[i].name <<"'s hand is " << p[i].tScore << "\t";
    }
    cout << endl;
}

//Set choice back & remove played card from hand (nested inside trick)
void unset(Player &p, Show &pfv){
    // set chosen card to 0 & empty face values so they dont show
    if (p.choice == p.hand.cards[0]) { p.hand.cards[0]  = 0; pfv.pshow[0]  = "";}
    if (p.choice == p.hand.cards[1]) { p.hand.cards[1]  = 0; pfv.pshow[1]  = "";}
    if (p.choice == p.hand.cards[2]) { p.hand.cards[2]  = 0; pfv.pshow[2]  = "";}
    if (p.choice == p.hand.cards[3]) { p.hand.cards[3]  = 0; pfv.pshow[3]  = "";}
    if (p.choice == p.hand.cards[4]) { p.hand.cards[4]  = 0; pfv.pshow[4]  = "";}
    if (p.choice == p.hand.cards[5]) { p.hand.cards[5]  = 0; pfv.pshow[5]  = "";}
    if (p.choice == p.hand.cards[6]) { p.hand.cards[6]  = 0; pfv.pshow[6]  = "";}
    if (p.choice == p.hand.cards[7]) { p.hand.cards[7]  = 0; pfv.pshow[7]  = "";}
    if (p.choice == p.hand.cards[8]) { p.hand.cards[8]  = 0; pfv.pshow[8]  = "";}
    if (p.choice == p.hand.cards[9]) { p.hand.cards[9]  = 0; pfv.pshow[9]  = "";}
    if (p.choice == p.hand.cards[10]){ p.hand.cards[10] = 0; pfv.pshow[10] = "";}
    if (p.choice == p.hand.cards[11]){ p.hand.cards[11] = 0; pfv.pshow[11] = "";}
    if (p.choice == p.hand.cards[12]){ p.hand.cards[12] = 0; pfv.pshow[12] = "";}
}

//Sort the best scores
void mSort(Win *w){
    //sort all four players
    for(int j = 0; j < 8-1; j++){
        for(int i = j + 1; i < 8; i++){
            if(w[j].score > w[i].score){
                //sort the player int values
                Win temp = w[j];
                w[j] = w[i];
                w[i] = temp;
            }
        }
    }

}
```