**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 the loop

//if current card == Ace Spades (39)

//set choice to j+1, chosen = true, match = false, exit the loop

//if current card == King Spades (38)

//set choice to j+1, chosen = true, match = false, exit the 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