Project 2

Title

Blackjack v7.0

An updated recreation of the Blackjack minigame
from Clover Studio's God Hand (2006)
for the Sony PlayStation 2.

Course

CSC/CIS-5

Section

40404

Due Date

February 11, 2024

Author

Noah Garthwaite

(a.k.a. Daisy Garthwaite)

(a.k.a. Dahlia Meat)

(a.k.a. Darlene Petal)

(a.k.a. Dippy)

(here we go again)

Table of Contents

Introduction

or:

"This Seems Familiar"

It's 7:00 PM on February 11th, 2024 as of writing this sentence. I don't have nearly as much time to spend on this write-up when compared to the previous project, so this one will not be nearly as entertaining or funny! The Spring semester begins tomorrow morning and I need to get this over with, so...

A lot of the "features" of this new version were more in the line of mandated feature-creep. The nature of this has obviously changed how (in)accurate this Blackjack program is with regards to the Blackjack minigame in God Hand (2006), but it's not like anybody cares about that (save for me).

Much of the code from Project 1 is, expectedly, re-purposed for the sake of time - but a lot of the game logic didn't need to be modified anyhow. There are some small improvements, such as:

- More consistent pausing between dealer turns
- Small bug fixes regarding double-downs
- No more cheating via reading .dat files
- No more -nan stats if user quit without playing

Surprisingly, I spent little time bug-fixing. I hardly tested my code when refactoring it to meet this project's requirements, but it all fit into place quite easily (likely due to the familiarity I've developed with the concepts introduced and my program itself). All of the code was complete in a manner of twenty hours, which is half as much time spent programming on Project 1. Neat!

Updates

or:

"Every Update We Stray

Further From God Hand's Light"

```
// function prototypes
bool     askMore(bool &);
float    takeBet(float, float [], float []);
void     endGame(float [], float [], short);
void     dckInit(short [], char [][MAXCOLS], short decks = 1);
void     shuffle(short [], char [][MAXCOLS], short);
short    calcHnd(vector<short>, bool &);
void     shwStrs(vector<string>, short);
void     shwStrs(vector<string>, short, bool);
void     selSrt(float [], short);
void     bublSrt(float [], short);
bool     binSrch(float [], short, float);

// execution begins
int main(void) {

    // set random no. seed
    srand(static_cast<unsigned int>(time(0)));

    // variable declarations            // used for storing...
    bool             moreDck,           // if more than one deck is used
                     usrSoft,           // if user has a soft ace
                     usrStnd,           // if user can draw more cards
                     dlrSoft,           // if dealer has a soft ace
                     dlrShow,           // if dealer's cards are shown
                     hndOver,           // if hand should end
                     validIn;           // if user input is valid
    char             crdsDlt,           // total cards dealt this hand
                     usrCrds,           // total cards user has
                     dlrCrds,           // total cards dealer has
                     usrInpt;           // one-character user input
    float            usrCash,           // money in user's wallet
                     betAmnt,           // floats tied to user bets
                     allCash[MAXRNDS],  // money won/lost each hand
                     allBets[MAXRNDS];  // money bet each hand
    short            decksNo,           // number of decks to use
                     roundNo,           // total rounds played
                     dlrTotl,           // total value of dealer's cards
                     usrTotl;           // total value of user's cards
    vector<short>    usrVals,           // ids of user's cards
                     dlrVals;           // ids of dealer's cards
    string           strInpt;           // multi-character user input
    vector<string>   usrStrs,           // user's card strings
                     dlrStrs;           // dealer's card strings
```

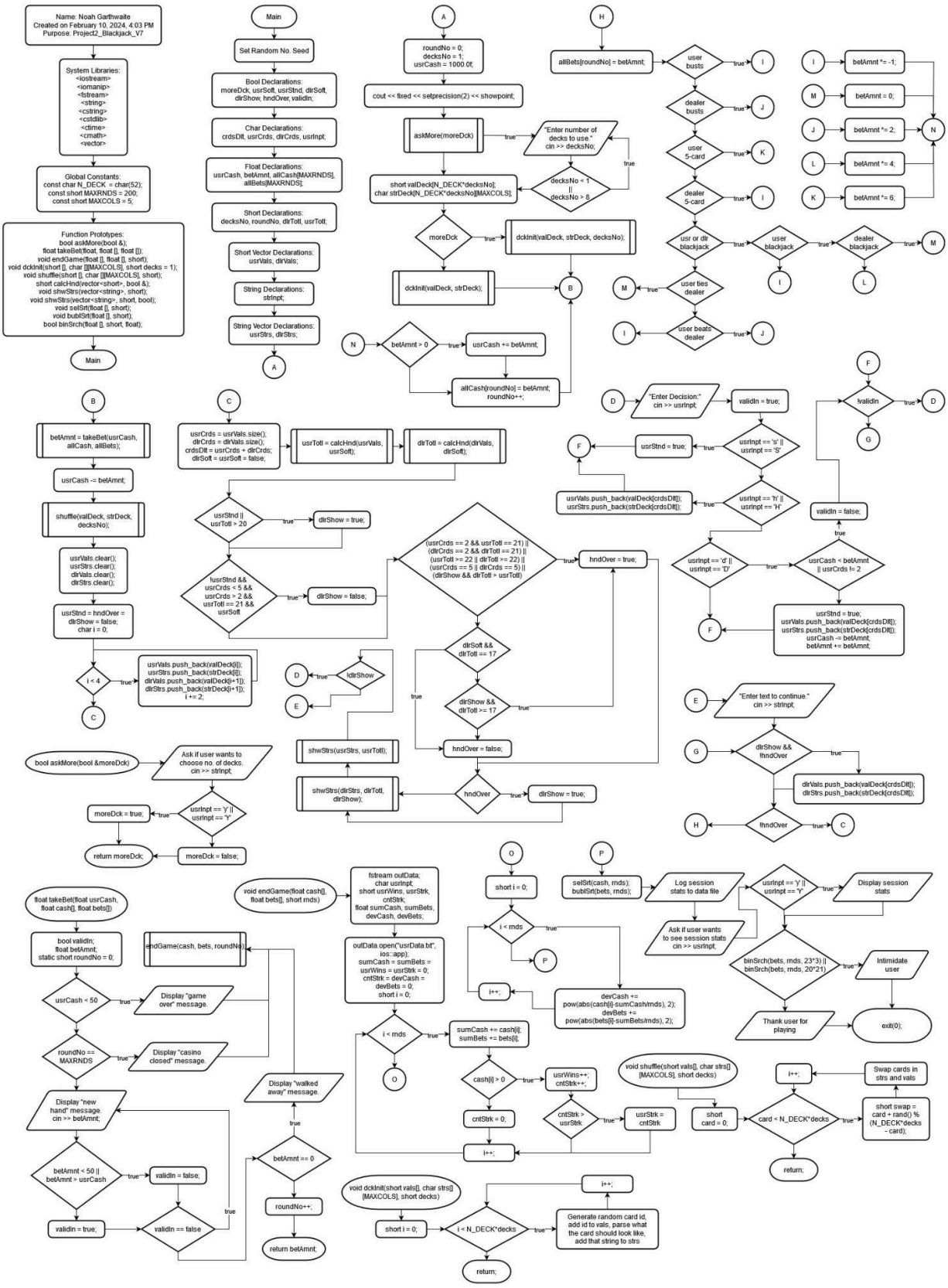
New features to Project 2 include:

- Allowing user to play with up to 8 decks
- Shuffling of full deck(s) each hand
- Dealing of first four cards in proper order
- More statistics displayed at session end
- Appending every session's stats into a .txt file
- Kicking user out of casino after 200 hands

In order to meet the project requirements, this program utilizes:

- 2-D array for storing card strings
- Vectors for storing dealer & user cards
- Bubble & Selection Sort of session stats
- Binary Search to see if user bet an unfunny number
- Pass-by-reference & pass-by-value
- Passing arrays & vectors to/from functions
- Numerous programmer-defined functions
- Overloaded functions in shwStrs()
- Completely unnecessary default argument in dckInit()

Funnily enough, the requirement for parallel arrays was essentially present in Project one through the usage of parallel reading of .dat files as a workaround for not being able to use arrays, so that was incredibly easy to implement. Also, having upwards of 8 decks makes getting Five Card incredibly lucrative. I see why it's not a casino standard. Now let's look at the academically obligatory flowchart!


Name: Noah Garthwaite
Created on February 10, 2024, 4:03 PM
Purpose: Project2_Blackjack_V7
System Libraries:
<iostream>
<iomanip>
<fstream>
<string>
<cstring>
<cstdlib>
<ctime>
<cmath>
<vector>
Global Constants:
const char N_DECK = char(52);
const short MAXRNDS = 200;
const short MAXCOLS = 5;
Function Prototypes:
bool askMore(bool &);
float takeBet(float, float [], float []);
void endGame(float [], float [], short);
void dckInit(short [], char [][MAXCOLS], short decks = 1);
void shuffle(short [], char [][MAXCOLS], short);
short calcHnd(vector<short>, bool &);
void shwStrs(vector<string>, short);
void shwStrs(vector<string>, short, bool);
void selSrt(float [], short);
void bublSrt(float [], short);
bool binSrch(float [], short, float);
Main
Main
Set Random No. Seed
Bool Declarations:
moreDck, usrSoft, usrStnd, dlrSoft,
dlrShow, hndOver, validIn;
Char Declarations:
crdsDlt, usrCrds, dlrCrds, usrInpt;
Float Declarations:
usrCash, betAmnt, allCash[MAXRNDS],
allBets[MAXRNDS];
Short Declarations:
decksNo, roundNo, dlrTotl, usrTotl;
Short Vector Declarations:
usrVals, dlrVals;
String Declarations:
strInpt;
String Vector Declarations:
usrStrs, dlrStrs;
A
A
roundNo = 0;
decksNo = 1;
usrCash = 1000.0f;
cout << fixed << setprecision(2) << showpoint;
askMore(moreDck)
true
"Enter number of decks to use."
cin >> decksNo;
decksNo < 1 || decksNo > 8
true
short valDeck[N_DECK*decksNo];
char strDeck[N_DECK*decksNo][MAXCOLS];
moreDck
true
dckInit(valDeck, strDeck, decksNo);
dckInit(valDeck, strDeck);
B
H
allBets[roundNo] = betAmnt;
user busts
true
I
dealer busts
true
J
user 5-card
true
K
dealer 5-card
true
I
usr or dlr blackjack
true
user blackjack
true
dealer blackjack
true
M
I
L
user ties dealer
M
true
user beats dealer
I
true
J
I
betAmnt *= -1;
M
betAmnt = 0;
J
betAmnt *= 2;
N
L
betAmnt *= 4;
K
betAmnt *= 6;
N
betAmnt > 0
true
usrCash += betAmnt;
allCash[roundNo] = betAmnt;
roundNo++;
B
betAmnt = takeBet(usrCash, allCash, allBets);
usrCash -= betAmnt;
shuffle(valDeck, strDeck, decksNo);
usrVals.clear();
usrStrs.clear();
dlrVals.clear();
dlrStrs.clear();
usrStnd = hndOver = dlrShow = false;
char i = 0;
i < 4
true
usrVals.push_back(valDeck[i]);
usrStrs.push_back(strDeck[i]);
dlrVals.push_back(valDeck[i+1]);
dlrStrs.push_back(strDeck[i+1]);
i += 2;
C
C
usrCrds = usrVals.size();
dlrCrds = dlrVals.size();
crdsDlt = usrCrds + dlrCrds;
dlrSoft = usrSoft = false;
usrTotl = calcHnd(usrVals, usrSoft);
dlrTotl = calcHnd(dlrVals, dlrSoft);
usrStnd || usrTotl > 20
true
dlrShow = true;
!usrStnd && usrCrds < 5 && usrCrds > 2 && usrTotl == 21 && usrSoft
true
dlrShow = false;
(usrCrds == 2 && usrTotl == 21) || (dlrCrds == 2 && dlrTotl == 21) || (usrTotl >= 22 || dlrTotl >= 22) || (usrCrds == 5 || dlrCrds == 5) || (dlrShow && dlrTotl > usrTotl)
true
hndOver = true;
dlrSoft && dlrTotl == 17
true
dlrShow && dlrTotl >= 17
true
hndOver = false;
hndOver
true
dlrShow = true;
D
!dlrShow
E
shwStrs(usrStrs, usrTotl);
shwStrs(dlrStrs, dlrTotl, dlrShow);
D
"Enter Decision:"
cin >> usrInpt;
validIn = true;
usrInpt == 's' || usrInpt == 'S'
true
F
usrStnd = true;
usrInpt == 'h' || usrInpt == 'H'
true
usrVals.push_back(valDeck[crdsDlt]);
usrStrs.push_back(strDeck[crdsDlt]);
usrInpt == 'd' || usrInpt == 'D'
true
usrCash < betAmnt || usrCrds != 2
true
usrStnd = true;
usrVals.push_back(valDeck[crdsDlt]);
usrStrs.push_back(strDeck[crdsDlt]);
usrCash -= betAmnt;
betAmnt += betAmnt;
F
validIn = false;
true
F
!validIn
true
D
G
N
betAmnt > 0
true
usrCash += betAmnt;
allCash[roundNo] = betAmnt;
roundNo++;
bool askMore(bool &moreDck)
Ask if user wants to choose no. of decks.
cin >> strInpt;
usrInpt == 'y' || usrInpt == 'Y'
true
moreDck = true;
moreDck = false;
return moreDck;
E
"Enter text to continue."
cin >> strInpt;
G
dlrShow && !hndOver
true
dlrVals.push_back(valDeck[crdsDlt]);
dlrStrs.push_back(strDeck[crdsDlt]);
H
!hndOver
true
C
float takeBet(float usrCash, float cash[], float bets[])
bool validIn;
float betAmnt;
static short roundNo = 0;
endGame(cash, bets, roundNo);
usrCash < 50
true
Display "game over" message.
roundNo == MAXRNDS
true
Display "casino closed" message.
Display "new hand" message.
cin >> betAmnt;
betAmnt < 50 || betAmnt > usrCash
true
validIn = false;
validIn = true;
validIn == false
true
Display "walked away" message.
true
betAmnt == 0
roundNo++;
return betAmnt;
void endGame(float cash[], float bets[], short rnds)
fstream outData;
char usrInpt;
short usrWins, usrStrk, cntStrk;
float sumCash, sumBets, devCash, devBets;
outData.open("usrData.txt", ios::app);
sumCash = sumBets = usrWins = usrStrk = 0;
cntStrk = devCash = devBets = 0;
short i = 0;
i < rnds
true
sumCash += cash[i];
sumBets += bets[i];
cash[i] > 0
true
usrWins++;
cntStrk++;
cntStrk = 0;
cntStrk > usrStrk
true
usrStrk = cntStrk
i++;
O
O
short i = 0;
i < rnds
true
P
i++;
P
devCash += pow(abs(cash[i]-sumCash/rnds), 2);
devBets += pow(abs(bets[i]-sumBets/rnds), 2);
selSrt(cash, rnds);
bublSrt(bets, rnds);
Log session stats to data file
Ask if user wants to see session stats
cin >> usrInpt;
usrInpt == 'y' || usrInpt == 'Y'
true
Display session stats
binSrch(bets, rnds, 23*3) || binSrch(bets, rnds, 20*21)
true
Intimidate user
Thank user for playing
exit(0);
void shuffle(short vals[], char strs[][MAXCOLS], short decks)
short card = 0;
card < N_DECK*decks
true
short swap = card + rand() % (N_DECK*decks - card);
Swap cards in strs and vals
i++;
return;
void dckInit(short vals[], char strs[][MAXCOLS], short decks)
short i = 0;
i < N_DECK*decks
true
Generate random card id, add id to vals, parse what the card should look like, add that string to strs
i++;
return;

The Rules

or:

"Padding Out My Write-Up"

The goal of a round of Blackjack is to beat The Dealer. Each round of play is a new "hand", as The Player and The Dealer discard all cards from the previous "hand" and are dealt new cards to their respective "hand". In this sense, a "hand" in Blackjack is a dual-purpose word meaning both a round of play and the cards a participant has during a round. The Player's hand is a collection of cards aiming to be better than The Dealer's. However, The Dealer cannot exactly "aim" to do anything, as The Dealer must play in a strict, robotic fashion. When playing Blackjack in a casino with multiple Players and one Dealer, all players individually face The Dealer and are not competing against one another.

Cards in Blackjack are worth the following:

- 2–10: Face value
- Jack, Queen, or King: 10
- Ace: 1 or 11

All card values in a participant's hand are added together. The highest total a participant may have is 21 – anything over is a "bust", an automatic loss.

If a hand contains an Ace, it may be worth 1 or 11 based on context. If a hand would not bust with an Ace valued at 11, it is worth 11 and is considered a "soft" Ace. Otherwise, the Ace is worth 1 and is considered a "hard" Ace. An Ace can change from hard to soft at any point, depending on whichever would benefit the hand most. I'll show off how and why the program determines whether a Hard Ace or Soft Ace is best for a hand in the later Gameplay Demo.

A round of this Blackjack game consists of up to five phases:

1. The Player places a bet for the hand.
2. The Player and The Dealer are dealt two cards each. The Dealer's first card is shown face-up, and the second is kept face-down.
    a. If The Dealer and/or The Player are dealt a 10-value card and an Ace in this first phase; this is a Blackjack, and the hand is over.
3. The Player takes their turn(s). If it is their first turn (and they have enough money), they have the option to "Double Down". This doubles their current bet, gives them one additional card, and begins Phase 4.
    a. If their hand totals over 21, the hand is over.
    b. If they collected five cards without busting, the hand is over.
    c. If they don't want another card, they stand. The Dealer reveals their hand and Phase 4 begins.
    d. If they do want another card, they hit. Phase 3 restarts.
4. The Dealer takes their turn(s).
    a. If their hand totals over 21, the hand is over.
    b. If The Dealer collects five cards without busting, the hand is over.
    c. If their total is better, the hand is over.
    d. If their hand totals 17 or higher, they stand, and the hand is over.
        i. Unless, however, they have a total of 17 and a soft ace, in which case they must hit.
    e. Otherwise, they must hit. Phase 4 restarts.

5. Once the hand is over, the winner is determined, and the player decides if they wish to play another hand (if they have enough money. All cards are collected by The Dealer and deck is reshuffled.
    a. If someone achieved Blackjack, they win.
        i. Unless both did, in which there is a tie.
    b. Otherwise, if someone busted, they lose.
    c. Otherwise, if someone achieved Five Card, they win.
    d. Otherwise, if someone's total is higher, they win.
    e. Otherwise, there is a tie.
    f. If there is a tie, The Player is returned their bet.
    g. If The Player won via Blackjack, they are returned their bet plus triple the bet amount.
    h. If The Player won via Five Card, they are returned their bet plus quintuple the bet amount.
    i. If The Player won via neither Blackjack nor Five Card, they are returned their bet plus the bet amount.s
    j. If The Player loses, their bet is not returned.

These are the rules of Blackjack as present in God Hand (2006). God Hand plays 1-deck Blackjack, but I added the ability to play with up to 8 decks in this version because I couldn't find anywhere else to add a default argument (its usage is completely redundant in the code, but Daddy wants four points). There is no splitting of pairs or insurance in God Hand (2006)'s Blackjack, so there is no splitting of

pairs or insurance in this game. If you don't know what any of that means, good news: You still don't need to!

Pseudocode

or:

"Nothing Will Be As Funny As

Quasicodo So I Won't Bother"

In plain English, the program should behave as so:

```
INCLUDE LIBRARIES
DECLARE GLOBAL CONSTANTS
DECLARE FUNCTIONS
SET RANDOM SEED
DECLARE VARIABLES & CONSTANTS
INITIALIZE VARIABLES
SET DECIMAL PRECISION
BEGIN GAMEPLAY LOOP
     IF USER ASKS TO USE MULTIPLE DECKS:
          PROMPT USER TO ENTER NUMBER OF DECKS
     CREATE CARD IDS AND STRINGS
     DISPLAY NEW-HAND MESSAGE & RULES
     LET USER BET OR END THE GAME
     IF ENDING THE GAME OR USER CAN'T BET:
          OPEN SESSION DATA FILE WRITER
          CALCULATE STATS
          APPEND STATS TO .TXT FILE
          ASK TO DISPLAY STATS
          IF STATS REQUESTED:
               DISPLAY THEM
          CLOSE THE PROGRAM
     REMOVE BET FROM THEIR WALLET
     BEGIN NEW HAND
          SHUFFLE DECK
          DEAL TWO CARDS EACH TO DEALER AND PLAYER
          IF EITHER GOT BLACKJACK:
               END THE HAND
```

```
    SHOW DEALER'S FIRST CARD, HIDE DEALER TOTAL
    SHOW PLAYER'S CARDS AND TOTAL
    BEGIN PLAYER'S TURN
        IF PLAYER BUSTED OR GOT FIVE CARD:
            END THE HAND
        IF PLAYER CAN'T HIT:
            END THEIR TURN
        DISPLAY DECISIONS AND PROMPT PLAYER INPUT
        IF PLAYER CAN'T DOUBLE DOWN:
            DON'T DISPLAY DOUBLE DOWN OPTION
        IF USER HITS:
            DEAL A NEW CARD AND REPEAT TURN
        IF USER STANDS:
            END THEIR TURN
        IF USER DOUBLES DOWN:
            IF THEY CAN DOUBLE DOWN:
                TAKE MONEY NEEDED FROM WALLET
                DEAL A NEW CARD
                IF THEY BUSTED:
                    END THE HAND
                IF THEY DIDN'T BUST:
                    END THEIR TURN
BEGIN DEALER'S TURN
    IF DEALER BUSTED:
        END THE HAND
    IF DEALER GOT FIVE CARD OR BEAT PLAYER:
        END THE HAND
    IF DEALER HAS MORE THAN 17 OR HARD 17:
```

END THE HAND

IF DEALER HAS LESS THAN 17 OR SOFT 17:

DEAL A NEW CARD AND REPEAT TURN

DETERMINE WHO WON

IF PLAYER DIDN'T LOSE:

GIVE PLAYER BET BACK

IF PLAYER WON:

CALCULATE WINNINGS AND AWARD TO PLAYER

LOG WIN/LOSS AMOUNT & BET TOTAL TO ARRAY

REPEAT GAMEPLAY LOOP

The pseudocode is much shorter this time, solely because I no longer have redundant program-ending states at the beginning and end of a hand. This time, it all happens at the beginning, as it should!

Though, it's worth nothing that with all of the refactoring of code present in this iteration of the project, Project 2 contains 60 more lines of code than Project 1 due to the amount of features I was forced to implement.

Why include a Bubble Sort *AND* a Selection Sort? Oh, because I am a greedy point ghoul. Every day for the past four weeks I have received brain-chip-damage upon opening Canvas, wherein a 6/7 Dr. Lehr said he would correct to a 7/7 has tormented me like a remnant K-12 GPA spectre cursing me to revisit distant eras barren of academia-passion, abundant instead with unmedicated report-card trauma. That singular point bothers me more than being thrown in a middle-school trash can ever did. What? Go to therapy. Okay, I will.

Highlights

or:

"Showing Off

My Pretty VSCode Palette"

There's not much to write home about in this new version of the program, but I love my Mom, so I'm going to write to her anyway:

```
decksNo = 1;        // default no. of decks
usrCash = 1000.0f;  // user starts with $1000.00

//configure output to show two decimal places on dollar amounts
cout << fixed << setprecision(2) << showpoint;

if (askMore(moreDck)) {···

// initialize deck arrays now that decksNo is decided
short   valDeck[N_DECK*decksNo];
char    strDeck[N_DECK*decksNo][MAXCOLS];

// fill deck(s) via redundant code to meet project criteria
(moreDck) ? dckInit(valDeck, strDeck, decksNo) :
            dckInit(valDeck, strDeck);
```

Hey, look, dckInit() has a default argument! What is that default argument, pray tell?

```
void    dckInit(short [], char [][MAXCOLS], short decks = 1);
```

Oh, you're saying it's the value "decks" was already assigned because I needed it to declare the size of the deck before filling it? Both the default argument and the ternary operation are completely pointless, you say? You're telling this is my third time griping over my own decision on how to utilize a default argument? Why am I writing solely in questions you aren't actually asking, is that what you're asking me?

```
// parse how to display a card's suit based on card id
switch ((i % N_DECK)/13)
{
    case 0:     strs[i][1] = '\xe2';
                strs[i][2] = '\x99';
                strs[i][3] = '\xa4';
                break;
    case 1:     strs[i][1] = '\xe2';
                strs[i][2] = '\x99';
                strs[i][3] = '\xa7';
                break;
    case 2:     strs[i][1] = '\xe2';
                strs[i][2] = '\x99';
                strs[i][3] = '\xa1';
                break;
    default:    strs[i][1] = '\xe2';
                strs[i][2] = '\x99';
                strs[i][3] = '\xa2';
}
strs[i][4] = '\0';
```

I figured how to string UTF-8 Hex Codes together when experimenting with card suits in Project 1, and it actually came in handy this time around due to the fact that card visuals are stored in a 2D char array instead of a string. In Project 1, all I had to do was this:

```
// parse how to display a card's suit based on card id
switch ((randVal-1) / 13)
{
    case 0:     tempStr = "♤";
                break;
    case 1:     tempStr = "♧";
                break;
    case 2:     tempStr = "♡";
                break;
    default:    tempStr = "♢";
}
```

```
void selSrt(float arr[], short arrSize) {
    int minIndx, temp;
    for (int i = 0; i < arrSize; i++) {···
    return;
}


void bublSrt(float arr[], short arrSize) {
    int     i,
            temp;
    bool    swapped;

    i = 0;
    do {···
    } while (swapped == true);
    return;
}


bool binSrch(float arr[], short arrSize, float toFind) {
    int first,
        middle,
        last;

    first = 0;
    last = arrSize-1;
    while (first <= last) {···
    return false;
}
```

The code for all three of these algorithms were ripped straight from Assignment 6 without changing anything, save for the amount of arguments in binSrch(). Like food, points taste better when they're free.

Don't code what ain't new!

Demonstration

or:

"I Receive Free Points For Literally Gambling"

```
Would you like to choose how many
decks are used in this session? (y/n)
y

Enter number of decks to use.
Input number: 0
Minimum number of decks is 1.

Input number: 9
Maximum number of decks is 8.

Input number: 8


O-----------BLACKJACK----------O
|     DEALER MUST HIT SOFT 17   |
|      BLACKJACK PAYS 3 TO 1    |
|      FIVE CARD PAYS 5 TO 1    |
O-----------*.°.+.°.*----------O

You have: $1000.00
How much would you like to bet?

Minimum bet is 50.
Input a bet of 0 to walk away
with your current cash.

Enter bet: 200
```

```
Dealer:
K♧  ??

Player: 9
4♧  5♧

Pot total: $200.00
You have: $800.00

What would you like to do?
Type S to Stand
Type H to Hit
Type D to Double Down
Enter decision: h

Dealer:
K♧  ??

Player: 20
4♧  5♧  A♧

Pot total: $200.00
You have: $800.00

What would you like to do?
Type S to Stand
Type H to Hit
Enter decision: h
```

```
Dealer:
K♤  ??

Player: 21
4♤  5♤  A♤  A♤

Pot total: $200.00
You have: $800.00

What would you like to do?
Type S to Stand
Type H to Hit
Enter decision: h

Dealer: 20
K♤  K♢

Player: 15
4♤  5♤  A♤  A♤  4♢

Enter any text to continue.
j

FIVE CARDS
YOU WIN!
```

```
O-----------BLACKJACK----------O
|    DEALER MUST HIT SOFT 17   |
|     BLACKJACK PAYS 3 TO 1    |
|     FIVE CARD PAYS 5 TO 1    |
O-----------*.°.+.°.*----------O

You have: $2000.00
How much would you like to bet?

Minimum bet is 50.
Input a bet of 0 to walk away
with your current cash.

Enter bet: 1000

Dealer:
2♧  ??

Player: 9
4♢  5♢

Pot total: $1000.00
You have: $1000.00

What would you like to do?
Type S to Stand
Type H to Hit
Type D to Double Down
Enter decision: h
```

```
Dealer:
2♧  ??

Player: 15
4♢  5♢  6♡

Pot total: $1000.00
You have: $1000.00

What would you like to do?
Type S to Stand
Type H to Hit
Enter decision: h

Dealer:
2♧  ??

Player: 20
4♢  5♢  6♡  5♧

Pot total: $1000.00
You have: $1000.00

What would you like to do?
Type S to Stand
Type H to Hit
Enter decision: s
```

```
Dealer: 12
2♣  Q♡

Player: 20
4♢  5♢  6♡  5♣

Enter any text to continue.
j

DEALER HITS

Dealer: 18
2♣  Q♡  6♣

Player: 20
4♢  5♢  6♡  5♣

Enter any text to continue.
j

DEALER STANDS
YOU WIN!
```

```
O-----------BLACKJACK----------O
|    DEALER MUST HIT SOFT 17   |
|     BLACKJACK PAYS 3 TO 1    |
|     FIVE CARD PAYS 5 TO 1    |
O-----------*.°.+.°.*----------O

You have: $3000.00
How much would you like to bet?

Minimum bet is 50.
Input a bet of 0 to walk away
with your current cash.

Enter bet: 1000

Dealer:
K♧  ??

Player: 9
5♡  4♤

Pot total: $1000.00
You have: $2000.00

What would you like to do?
Type S to Stand
Type H to Hit
Type D to Double Down
Enter decision: h
```

```
Dealer:
K♧  ??

Player: 13
5♡  4♤  4♧

Pot total: $1000.00
You have: $2000.00

What would you like to do?
Type S to Stand
Type H to Hit
Enter decision: h

Dealer:
K♧  ??

Player: 20
5♡  4♤  4♧  7♡

Pot total: $1000.00
You have: $2000.00

What would you like to do?
Type S to Stand
Type H to Hit
Enter decision: s
```

```
Dealer: 14
K♧  4♧

Player: 20
5♡  4♤  4♧  7♡

Enter any text to continue.
j

DEALER HITS

Dealer: 15
K♧  4♧  A♢

Player: 20
5♡  4♤  4♧  7♡

Enter any text to continue.
j

DEALER HITS

Dealer: 18
K♧  4♧  A♢  3♢

Player: 20
5♡  4♤  4♧  7♡

Enter any text to continue.
j
```

```
DEALER STANDS
YOU WIN!

O-----------BLACKJACK----------O
|    DEALER MUST HIT SOFT 17   |
|     BLACKJACK PAYS 3 TO 1    |
|     FIVE CARD PAYS 5 TO 1    |
O-----------*.°.+.°.*----------O

You have: $4000.00
How much would you like to bet?

Minimum bet is 50.
Input a bet of 0 to walk away
with your current cash.

Enter bet: 4000

Dealer: 21
T♧  A♧

Player: 18
8♧  K♧

Enter any text to continue.
k

BLACKJACK
DEALER WINS
```

```
You ran out of money and were
kicked out of the casino.

GAME OVER

See session statistics? (y/n)

Enter decision: y

Hands played.....4
Hands lost.....1
Hands won.....3
Best win streak.....3
Most $ won.....$2000.00
Average $ won.....$300.00
Standard deviation....$2504.00
Most $ bet.....$4000.00
Average $ bet.....$1550.00
Standard deviation....$1451.72

Thank you for playing!

RUN SUCCESSFUL (total time: 45s)
```

Conclusion

or:

"See You Space Codeboy..."

This project, like this class, was a lot of work. I didn't realize until this morning that I currently have CIS-17A listed as my summer class this year. I am going to do just about anything within my power to see if it doesn't have to be that way; but truthfully, there was something cathartic about taking this class so quickly and so intensely. It severely impacted my social life to the point where three different people close to me said, ver batim: "I'm worried about you."

That's concerning! Though, it's given me a taste of what a "code boot camp" might actually be like. It wasn't pleasant or easy by any means, but it was incredibly rewarding. I especially enjoyed helping all my fellow classmates - nearly six people were reaching out regularly to ask for help, and were incredibly appreciative that I would help them understand the material they were working with. It feels as though I've been working with C++ for at least three months - but that's likely just the result of working three times as hard programming than I ever have before.

And, at the end of it, I have an incredibly tight game of Blackjack that was of my own design (excluding bits I added solely to meet the criteria). I'm quite proud of what I've been able to accomplish with not only this project, but with this class as a whole. I was far from a good student before I returned last semester after a five-year gap period. I'm left feeling incredibly encouraged, especially after hearing the most experienced programming professor I've had say to me: "You're one of my brightest students." Here's hoping the years ahead of me are bright, too..

The Code

or:

"The Code"

```
/*
 * File:     main.cpp
 * Author:  Noah Garthwaite
 * Created on February 10, 2024, 4:03 PM
 * Purpose: Project2_Blackjack_V7
 */

// system libraries // used to...
#include <iostream> // get user input and display outputs
#include <iomanip>  // format dollar amount outputs
#include <fstream>  // handle session stats & store cards for current hand
#include <string>   // handle "continue" input & suit ascii
#include <cstring>  // shuffle str values via strcpy
#include <cstdlib>  // randomize cards drawn
#include <ctime>    // set the random number seed
#include <cmath>    // calculate standard deviation stats
#include <vector>   // dynamically add cards to hands

using namespace std;

// global constants
const char  N_DECK  = char(52); // no. of cards in one deck
const short MAXRNDS = 200;       // maximum no. of rounds
const short MAXCOLS = 5;         // size of

// function prototypes
bool   askMore(bool &);
float   takeBet(float, float [], float []);
void   endGame(float [], float [], short);
void   dckInit(short [], char [][MAXCOLS], short decks = 1);
void   shuffle(short [], char [][MAXCOLS], short);
short   calcHnd(vector<short>, bool &);
```

```
void   shwStrs(vector<string>, short);
void   shwStrs(vector<string>, short, bool);
void   selSrt(float [], short);
void   bublSrt(float [], short);
bool   binSrch(float [], short, float);

// execution begins
int main(void) {

      // set random no. seed
      srand(static_cast<unsigned int>(time(0)));

      // variable declarations         // used for storing...
      bool         moreDck,            // if more than one deck is used
                   usrSoft,            // if user has a soft ace
                   usrStnd,            // if user can draw more cards
                   dlrSoft,            // if dealer has a soft ace
                   dlrShow,            // if dealer's cards are shown
                   hndOver,            // if hand should end
                   validIn;            // if user input is valid
      char         crdsDlt,            // total cards dealt this hand
                   usrCrds,            // total cards user has
                   dlrCrds,            // total cards dealer has
                   usrInpt;            // one-character user input
      float        usrCash,            // money in user's wallet
                   betAmnt,            // floats tied to user bets
                   allCash[MAXRNDS],   // money won/lost each hand
                   allBets[MAXRNDS];   // money bet each hand
      short        decksNo,            // number of decks to use
                   roundNo,            // total rounds played
                   dlrTotl,            // total value of dealer's cards
                   usrTotl;            // total value of user's cards
```

```
vector<short>   usrVals,         // ids of user's cards
            dlrVals;             // ids of dealer's cards
string      strInpt;             // multi-character user input
vector<string>  usrStrs,         // user's card strings
            dlrStrs;             // dealer's card strings

roundNo = 0;
decksNo = 1;        // default no. of decks
usrCash = 1000.0f;  // user starts with $1000.00

//configure output to show two decimal places on dollar amounts
cout << fixed << setprecision(2) << showpoint;

if (askMore(moreDck)) {
cout << endl << "Enter number of decks to use." << endl;
do {
      cout << "Input number: ";
      cin >> decksNo;
      if (decksNo < 1)
      cout << "Minimum number of decks is 1.";
      else if (decksNo > 8)
      cout << "Maximum number of decks is 8.";
      cout << endl << endl;
} while (decksNo < 1 || decksNo > 8);
}

// initialize deck arrays now that decksNo is decided
short   valDeck[N_DECK*decksNo];
char   strDeck[N_DECK*decksNo][MAXCOLS];

// fill deck(s) via redundant code to meet project criteria
(moreDck) ? dckInit(valDeck, strDeck, decksNo) :
```

```
	dckInit(valDeck, strDeck);

// gameplay loop starts
while (true) {
// get user bet & check if game should end
betAmnt = takeBet(usrCash, allCash, allBets);

// put user bet into the pot
usrCash -= betAmnt;

// shuffle deck(s)
shuffle(valDeck, strDeck, decksNo);

// empty dealer & user hands
usrVals.clear();
usrStrs.clear();
dlrVals.clear();
dlrStrs.clear();

// reset hand bools
usrStnd = hndOver = dlrShow = false;

// deal starting cards in proper order
for (char i = 0; i < 4; i += 2) {
	usrVals.push_back(valDeck[i]);
	usrStrs.push_back(strDeck[i]);
	dlrVals.push_back(valDeck[i+1]);
	dlrStrs.push_back(strDeck[i+1]);
}

// begin hand loop
do {
```

```
// track how many cards are in hands
usrCrds = usrVals.size();
dlrCrds = dlrVals.size();
crdsDlt = usrCrds + dlrCrds;

// reset soft ace bools
dlrSoft = usrSoft = false;
usrTotl = calcHnd(usrVals, usrSoft);
dlrTotl = calcHnd(dlrVals, dlrSoft);

// check if user can draw more cards
if (usrStnd || usrTotl > 20)
dlrShow = true;
// ... unless user can still go for five cards
if (usrCrds < 5 && usrCrds > 2 && !usrStnd)
if (usrTotl == 21 && usrSoft)
      dlrShow = false;

// current hand should be over if...
hndOver =
// user has blackjack
(usrCrds == 2 && usrTotl == 21)  ? true  :
// dealer has blackjack
(dlrCrds == 2 && dlrTotl == 21)  ? true  :
// user busted
(usrTotl >= 22 || dlrTotl >= 22) ? true  :
// user got five cards
(usrCrds == 5 || dlrCrds == 5)   ? true  :
// dealer beat the user
(dlrShow && dlrTotl > usrTotl)   ? true  :
// check for soft 17
(dlrSoft && dlrTotl == 17)       ? false :
```

```
// dealer must stand
(dlrShow && dlrTotl >= 17)       ? true  : false;

// show dealer cards if hand is over
if (hndOver) dlrShow = true;

// display dealer and user cards
shwStrs(dlrStrs, dlrTotl, dlrShow);
shwStrs(usrStrs, usrTotl);

// prompt user decision if they can make one
if (!dlrShow) {
cout << "Pot total: $" << betAmnt << endl;
cout << "You have: $" << usrCash << endl << endl;
cout << "What would you like to do?" << endl;
cout << "Type S to Stand" << endl;
cout << "Type H to Hit" << endl;
// don't display double-down option if user can't do it
if (usrCash >= betAmnt && usrCrds == 2) {
      cout << "Type D to Double Down" << endl;
}

// validate input
do {
      cout << "Enter decision: ";
      cin >> usrInpt;
      validIn = true;

      switch (usrInpt)
      {
            case 's':   // indicate user is done drawing cards
            case 'S':   usrStnd = true;
```

```
                break;
        case 'h':   // give user another card
        case 'H':   usrVals.push_back(valDeck[crdsDlt]);
                usrStrs.push_back(strDeck[crdsDlt]);
                break;
        case 'd':   // if user can't double down but tries
        case 'D':   if (usrCash < betAmnt || usrCrds != 2) {
                        validIn = false;
                }
                else {
                        // user must stand after double down
                        usrStnd = true;
                        usrVals.push_back(valDeck[crdsDlt]);
                        usrStrs.push_back(strDeck[crdsDlt]);
                        usrCash -= betAmnt;
                        betAmnt += betAmnt;
                }
                break;
        default:    validIn = false;
}
// display why input is invalid
if (!validIn) {
        if (usrInpt == 'd' || usrInpt == 'D') {
        if (usrCrds != 2) {
                cout << "You can only double down" << endl;
                cout << "if you have two cards.";
        }
        else {
                cout << "You need to have at least" << endl;
                cout << "$" << betAmnt << " to double down.";
        }
        }
```

```
                    else cout << "Invalid input.";
                    cout << endl << endl;
              }
       } while (!validIn);
       cout << endl;
       }
       else {
       cout << "Enter any text to continue." << endl;
       cin >> strInpt;
       cout << endl;
       }

       if (dlrShow && !hndOver) {
       cout << "DEALER HITS" << endl << endl;
       dlrVals.push_back(valDeck[crdsDlt]);
       dlrStrs.push_back(strDeck[crdsDlt]);
       }
} while (!hndOver);

// log how much was bet this hand
allBets[roundNo] = betAmnt;

// determine winner
if (usrTotl > 21) {
       cout << "YOU BUST" << endl;
       cout << "DEALER WINS" << endl;
       betAmnt *= -1;  // make betAmnt negative to log a loss
}
else if (dlrTotl > 21) {
       cout << "DEALER BUSTS" << endl;
       cout << "YOU WIN!" << endl;
       betAmnt *= 2;   // standard win pays 1 to 1
```

```
}
else if (usrCrds == 5 || dlrCrds == 5) {
	cout << "FIVE CARDS" << endl;
	if (usrCrds == 5) {
	cout << "YOU WIN!" << endl;
	betAmnt *= 6;   // five cards pays 5 to 1
	}
	else {
	cout << "DEALER WINS" << endl;
	betAmnt *= -1;
	}
}
else if (crdsDlt == 4 && (dlrTotl == 21 || usrTotl == 21)) {
	cout << "BLACKJACK" << endl;
	if (usrTotl == 21 && dlrTotl == 21) {
	cout << "TIE" << endl;
	usrCash += betAmnt; // return pot to the user
	betAmnt = 0; // set betAmnt to zero to log a tie
	}
	else if (usrTotl == 21) {
	cout << "YOU WIN!" << endl;
	betAmnt *= 4;   // blackjack pays 3 to 1
	}
	else {
	cout << "DEALER WINS" << endl;
	betAmnt *= -1;
	}
}
else if (usrTotl == dlrTotl) {
	cout << "DEALER STANDS" << endl;
	cout << "TIE" << endl;
	usrCash += betAmnt;
```

```
            betAmnt = 0;
        }
        else {
            cout << "DEALER STANDS" << endl;
            if (usrTotl > dlrTotl) {
            cout << "YOU WIN!" << endl;
            betAmnt *= 2;
            }
            else {
            cout << "DEALER WINS" << endl;
            betAmnt *= -1;
            }
        }
        cout << endl;

        // reward user if they won
        usrCash += (betAmnt > 0) ? betAmnt : 0;

        // log money won/lost this round
        allCash[roundNo] = betAmnt;

        // increment round counter
        roundNo++;
        }
}


bool askMore(bool &moreDck) {
        char usrInpt;

        cout << "Would you like to choose how many" << endl;
        cout << "decks are used in this session? (y/n)" << endl;
```

```
	cin >> usrInpt;

	moreDck = (usrInpt == 'y' || usrInpt == 'Y') ? true : false;

	return moreDck;
}


float takeBet(float usrCash, float cash[], float bets[]) {
	bool	validIn;
	float	betAmnt;
	static short roundNo = 0;

	// if user can't make minimum bet, game is over
	if (usrCash < 50) {
	if (usrCash == 0) {
		cout << "You ran out of money and were" << endl;
		cout << "kicked out of the casino." << endl;
	}
	else {
		cout << "You did not have enough money" << endl;
		cout << "to continue playing and walked" << endl;
		cout << "home in shame." << endl;
	}
	cout << endl;
	cout << "GAME OVER" << endl << endl;
	endGame(cash, bets, roundNo);
	}

	if (roundNo == MAXRNDS) {
	cout << "You played until the casino closed" << endl;
	cout << "and they kicked you out." << endl << endl;
```

```
endGame(cash, bets, roundNo);
}
else {
cout << "O----------BLACKJACK----------O" << endl;
cout << "|   DEALER MUST HIT SOFT 17   |" << endl;
cout << "|   BLACKJACK PAYS 3 TO 1     |" << endl;
cout << "|   FIVE CARD PAYS 5 TO 1     |" << endl;
cout << "O----------*⁺°.+.°⁺*----------O" << endl;

cout << endl << "You have: $" << usrCash << endl;
cout << "How much would you like to bet?" << endl << endl;
cout << "Minimum bet is 50." << endl;
cout << "Input a bet of 0 to walk away" << endl;
cout << "with your current cash." << endl << endl;

// let user decide to bet or walk away
do {
       cout << "Enter bet: ";
       cin >> betAmnt;
       cout << endl;

       // validate user input
       if (betAmnt < 50.0f && betAmnt > 0) {
       validIn = false;
       cout << "Bet must be at least 50." << endl;
       }
       else if (betAmnt > usrCash) {
       validIn = false;
       cout << "You can't bet more than you have, wise guy.";
       cout << endl;
       }
       else if (betAmnt < 0.0f) {
```

```
            validIn = false;
            cout << "You can't bet a negative amount. Nice try.";
            cout << endl;
            }
            else validIn = true;
      } while (validIn == false);
      }

      if (betAmnt == 0) {
      cout << "You walked away with $" << usrCash;
      // no need to celebrate if the user didn't win anything
      (usrCash > 1000.0f) ? (cout << "!") : (cout << ".");
      cout << endl;
      endGame(cash, bets, roundNo);
      }

      roundNo++;

      return betAmnt;
}


void endGame(float cash[], float bets[], short rnds) {
      fstream outData;
      char   usrInpt;
      short   usrWins,
            usrStrk,
            cntStrk;
      float   sumCash,
            sumBets,
            devCash,
            devBets;
```

```
outData.open("usrData.txt", ios::app);
sumCash = sumBets = usrWins = usrStrk = 0;
cntStrk = devCash = devBets = 0;

// calculate bet/cash sums and win streak
for (short i = 0; i < rnds; i++) {
sumCash += cash[i];
sumBets += bets[i];
if (cash[i] > 0) {
       usrWins++;
       usrStrk = (++cntStrk > usrStrk) ? cntStrk : usrStrk;
}
else cntStrk = 0;
}

// calculate standard deviation sums
for (short i = 0; i < rnds; i++) {
devCash += pow(abs(cash[i] - sumCash/rnds), 2);
devBets += pow(abs(bets[i] - sumBets/rnds), 2);
}

// sort arrays for easy
selSrt(cash, rnds);
bublSrt(bets, rnds);

// log session statistics in .txt file
outData << "----------------------------------------" << endl;
outData << "Hands played..........";
outData << rnds << endl;
outData << "Hands lost............";
outData << rnds - usrWins << endl;
```

```
outData << "Hands won.............";
outData << usrWins << endl;
outData << "Best win streak........";
outData << usrStrk << endl;
outData << "Most $ won...........$";
outData << cash[rnds-1] << endl;
outData << "Average $ won.........$";
outData << ((rnds > 0) ? sumCash/rnds : 0) << endl;
outData << "Standard deviation....$";
outData << ((rnds > 0) ? sqrt(devCash/rnds) : 0) << endl;
outData << "Most $ bet...........$";
outData << bets[rnds-1] << endl;
outData << "Average $ bet.........$";
outData << ((rnds > 0) ? sumBets/rnds : 0) << endl;
outData << "Standard deviation....$";
outData << ((rnds > 0) ? sqrt(devBets/rnds) : 0) << endl;
outData << "---------------------------------------" << endl;

// ask if user wants to see
cout << "See session statistics? (y/n)" << endl << endl;
cout << "Enter decision: ";
cin >> usrInpt;
cout << endl;

// display if so
if (usrInpt == 'y' || usrInpt == 'Y') {
cout << "Hands played...........";
cout << rnds << endl;
cout << "Hands lost.............";
cout << rnds - usrWins << endl;
cout << "Hands won.............";
cout << usrWins << endl;
```

```
	cout << "Best win streak........";
	cout << usrStrk << endl;
	cout << "Most $ won............$";
	cout << cash[rnds-1] << endl;
	cout << "Average $ won.........$";
	cout << ((rnds > 0) ? sumCash/rnds : 0) << endl;
	cout << "Standard deviation....$";
	cout << ((rnds > 0) ? sqrt(devCash/rnds) : 0) << endl;
	cout << "Most $ bet............$";
	cout << bets[rnds-1] << endl;
	cout << "Average $ bet.........$";
	cout << ((rnds > 0) ? sumBets/rnds : 0) << endl;
	cout << "Standard deviation....$";
	cout << ((rnds > 0) ? sqrt(devBets/rnds) : 0) << endl;
	cout << endl;
	}

	// i used binary search please give me an A :)
	if (binSrch(bets, rnds, 23*3) || binSrch(bets, rnds, 20*21))
	cout << "I know what you did." << endl;
	else
	cout << "Thank you for playing!" << endl;

	exit(0);
}


void dckInit(short vals[], char strs[][MAXCOLS], short decks) {
	for (short i = 0; i < N_DECK*decks; i++) {
	// log card id
	vals[i] = i+1;
```

```
// parse how to display a card's face value based on card id
switch (i % 13)
{
	case 0:		strs[i][0] = 'A';
			break;
	case 1:		strs[i][0] = '2';
			break;
	case 2:		strs[i][0] = '3';
			break;
	case 3:		strs[i][0] = '4';
			break;
	case 4:		strs[i][0] = '5';
			break;
	case 5:		strs[i][0] = '6';
			break;
	case 6:		strs[i][0] = '7';
			break;
	case 7:		strs[i][0] = '8';
			break;
	case 8:		strs[i][0] = '9';
			break;
	case 9:		strs[i][0] = 'T';
			break;
	case 10:	strs[i][0] = 'J';
			break;
	case 11:	strs[i][0] = 'Q';
			break;
	default:	strs[i][0] = 'K';
}

// parse how to display a card's suit based on card id
switch ((i % N_DECK)/13)
```

```
	{
		case 0:		strs[i][1] = '\xe2';
				strs[i][2] = '\x99';
				strs[i][3] = '\xa4';
				break;
		case 1:		strs[i][1] = '\xe2';
				strs[i][2] = '\x99';
				strs[i][3] = '\xa7';
				break;
		case 2:		strs[i][1] = '\xe2';
				strs[i][2] = '\x99';
				strs[i][3] = '\xa1';
				break;
		default:	strs[i][1] = '\xe2';
				strs[i][2] = '\x99';
				strs[i][3] = '\xa2';
	}
	strs[i][4] = '\0';
	}
	return;
}


void shuffle(short vals[], char strs[][MAXCOLS], short decks) {
	for (short card = 0; card < N_DECK*decks; card++) {
	char   tempStr[MAXCOLS];
	short   tempVal;

	// pick random card to swap with
	short swap = card + rand()%(N_DECK*decks - card);

	// swap strings
```

```
        strcpy(tempStr, strs[card]);
        strcpy(strs[card], strs[swap]);
        strcpy(strs[swap], tempStr);

        // swap ids
        tempVal = vals[card];
        vals[card] = vals[swap];
        vals[swap] = tempVal;
        }
        return;
}


short calcHnd(vector<short> vals, bool &soft) {
        short val;
        short total = 0;
        for (char i = char(0); i < vals.size(); i++) {
        // parse what card face is
        val = (vals[i]-1)%13+1;

        // set value to 10 if face card
        if (val > 10) val = 10;
        total += val;

        // determine how aces are calculated
        if (val == 1 && total < 12 && !soft) {
                total += 10;
                soft = true;
        }
        else if (soft && total > 21) {
                total -= 10;
                soft = false;
```

```
        }
        }
        return total;
}


void shwStrs(vector<string> strs, short total, bool show) {
        cout << "Dealer: ";
        if (show) cout << total;
        cout << endl;

        // display dealer cards
        for (char i = char(0); i < strs.size(); i++) {
        // hide second dealer card if needed
        if (!show && i == 1) cout << "??";
        else cout << strs[i];
        cout << " ";
        }
        cout << endl << endl;
        return;
}


void shwStrs(vector<string> strs, short total) {
        cout << "Player: " << total << endl;

        // display user cards
        for (char i = char(0); i < strs.size(); i++)
        cout << strs[i] << " ";
        cout << endl << endl;
        return;
}
```

```
void selSrt(float arr[], short arrSize) {
	int minIndx, temp;
	for (int i = 0; i < arrSize; i++) {
	int minIndx = temp = i;

	for (int j = i+1; j < arrSize; j++)
		if (arr[j] < arr[minIndx])
		minIndx = j;

	if (minIndx != i) {
		temp = arr[i];
		arr[i] = arr[minIndx];
		arr[minIndx] = temp;
	}
	}
	return;
}


void bublSrt(float arr[], short arrSize) {
	int	i,
		temp;
	bool	swapped;

	i = 0;
	do {
	swapped = false;
	for (int j = 0; j < arrSize-i-1; j++) {
		if (arr[j] > arr[j+1]) {
		temp		= arr[j];
```

```
            arr[j]       = arr[j+1];
            arr[j+1]     = temp;
            swapped      = true;
            }
      }
      i++;
      } while (swapped == true);
      return;
}


bool binSrch(float arr[], short arrSize, float toFind) {
      int first,
      middle,
      last;

      first = 0;
      last = arrSize-1;
      while (first <= last) {
      middle = (first + last)/2;
      if (arr[middle] == toFind)
            return true;
      else if (arr[middle] > toFind)
            last = middle-1;
      else if (arr[middle] < toFind)
            first = middle+1;
      }
      return false;
}
```