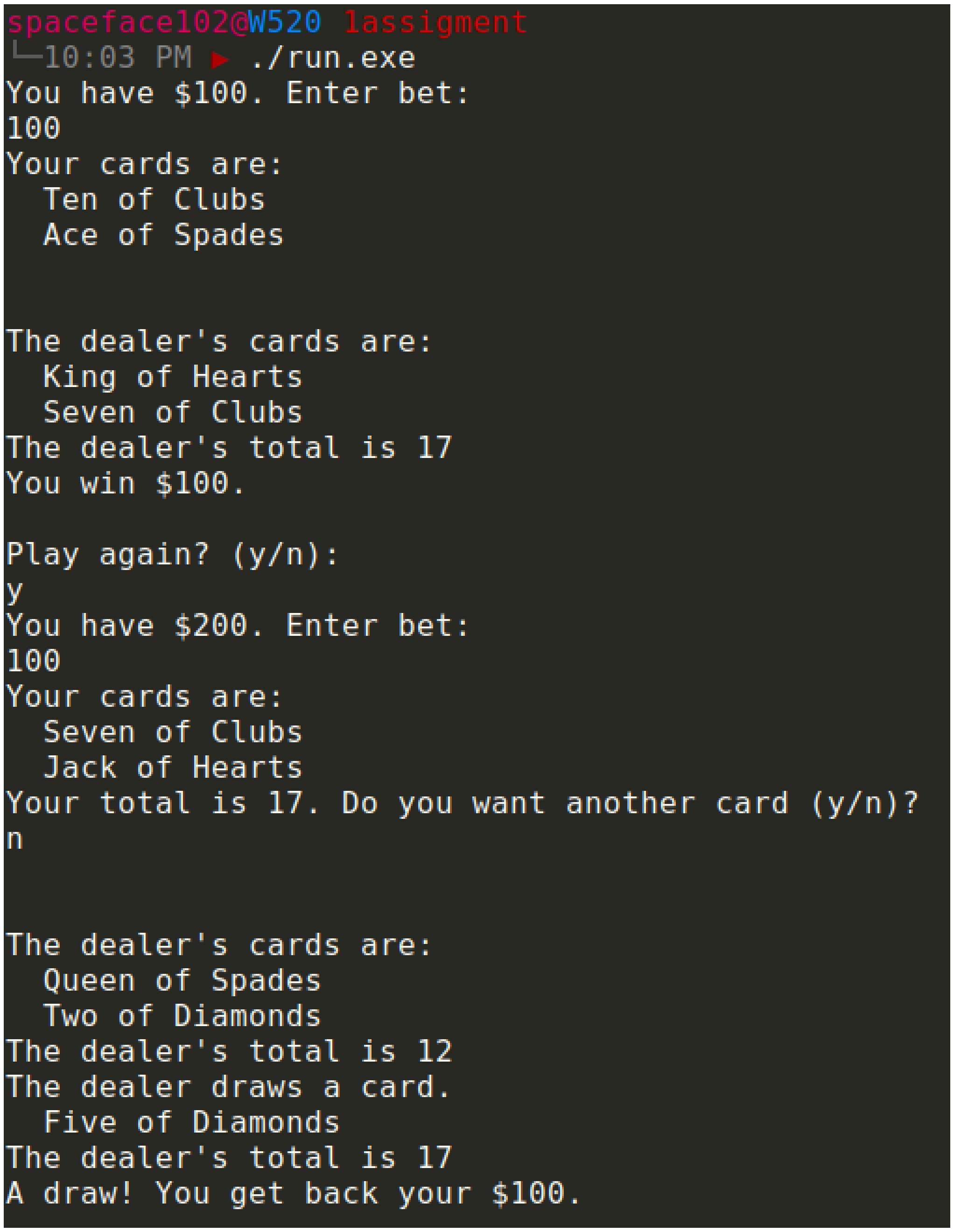
Result of code:

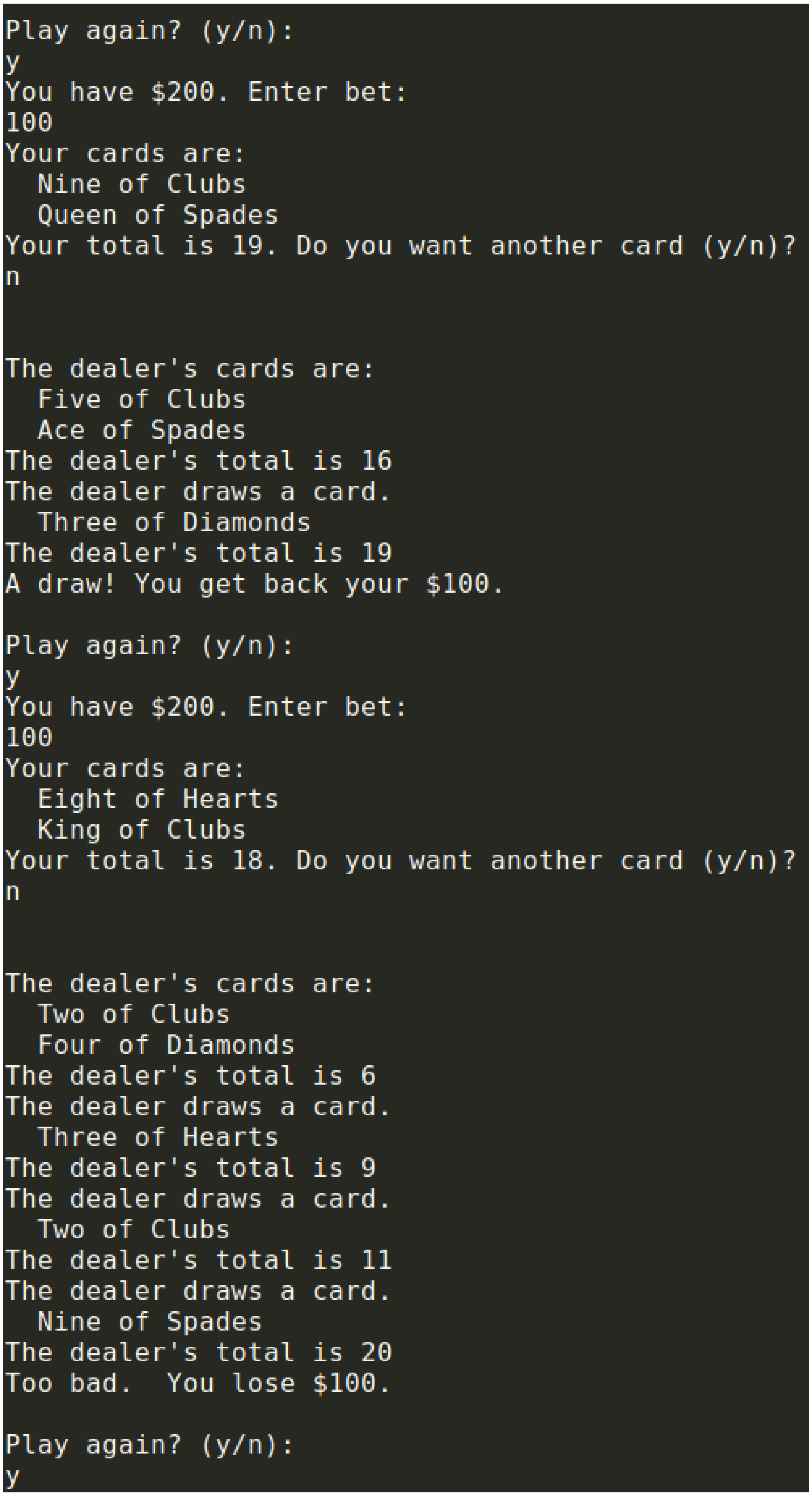


```
spaceface102@W520 lassigment
└─10:03 PM ▶ ./run.exe
You have $100. Enter bet:
100
Your cards are:
  Ten of Clubs
  Ace of Spades


The dealer's cards are:
  King of Hearts
  Seven of Clubs
The dealer's total is 17
You win $100.

Play again? (y/n):
y
You have $200. Enter bet:
100
Your cards are:
  Seven of Clubs
  Jack of Hearts
Your total is 17. Do you want another card (y/n)?
n


The dealer's cards are:
  Queen of Spades
  Two of Diamonds
The dealer's total is 12
The dealer draws a card.
  Five of Diamonds
The dealer's total is 17
A draw! You get back your $100.
```

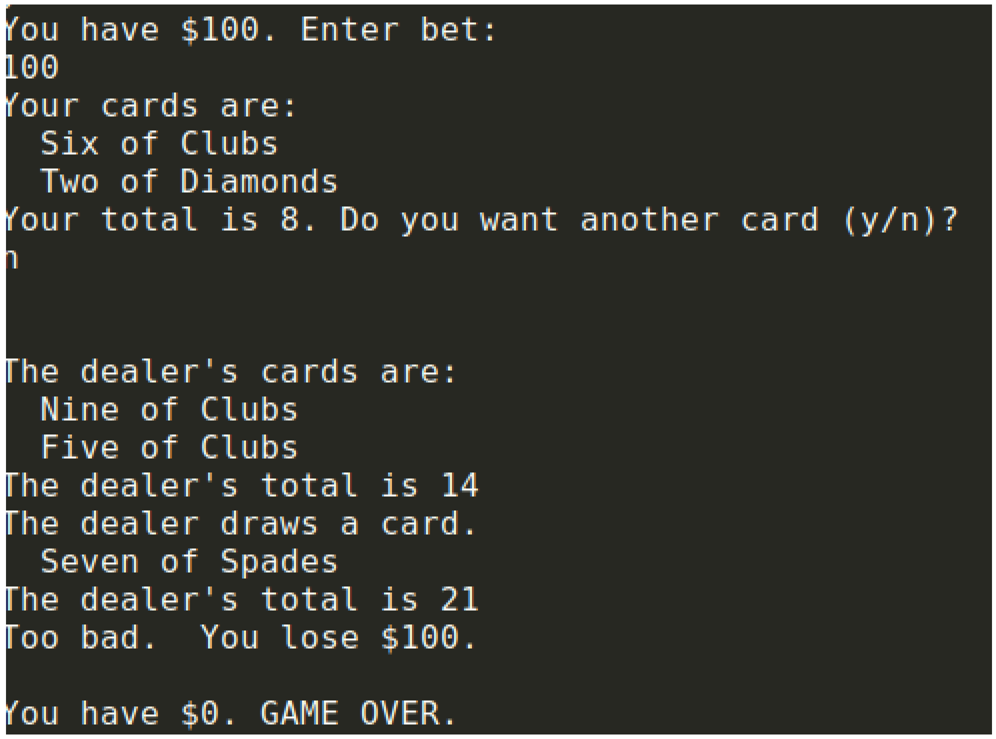

```
Play again? (y/n):
y
You have $200. Enter bet:
100
Your cards are:
  Nine of Clubs
  Queen of Spades
Your total is 19. Do you want another card (y/n)?
n


The dealer's cards are:
  Five of Clubs
  Ace of Spades
The dealer's total is 16
The dealer draws a card.
  Three of Diamonds
The dealer's total is 19
A draw! You get back your $100.

Play again? (y/n):
y
You have $200. Enter bet:
100
Your cards are:
  Eight of Hearts
  King of Clubs
Your total is 18. Do you want another card (y/n)?
n


The dealer's cards are:
  Two of Clubs
  Four of Diamonds
The dealer's total is 6
The dealer draws a card.
  Three of Hearts
The dealer's total is 9
The dealer draws a card.
  Two of Clubs
The dealer's total is 11
The dealer draws a card.
  Nine of Spades
The dealer's total is 20
Too bad.  You lose $100.

Play again? (y/n):
y
```

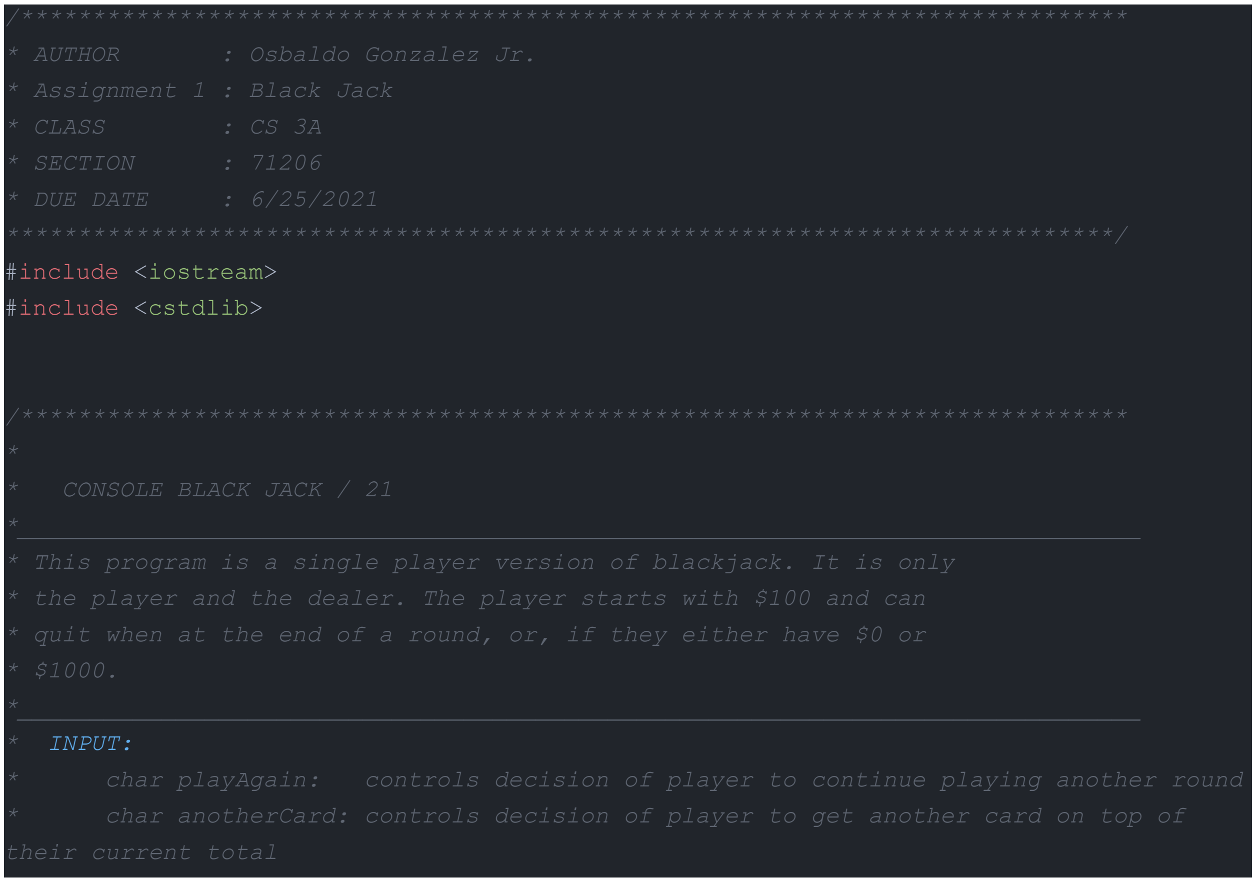


```
You have $100. Enter bet:
100
Your cards are:
  Six of Clubs
  Two of Diamonds
Your total is 8. Do you want another card (y/n)?
n


The dealer's cards are:
  Nine of Clubs
  Five of Clubs
The dealer's total is 14
The dealer draws a card.
  Seven of Spades
The dealer's total is 21
Too bad.  You lose $100.

You have $0. GAME OVER.
```

Code:



```
/*****************************************************************************
* AUTHOR       : Osbaldo Gonzalez Jr.
* Assignment 1 : Black Jack
* CLASS        : CS 3A
* SECTION      : 71206
* DUE DATE     : 6/25/2021
*****************************************************************************/
#include <iostream>
#include <cstdlib>



/*****************************************************************************
*
*   CONSOLE BLACK JACK / 21
*
* This program is a single player version of blackjack. It is only
* the player and the dealer. The player starts with $100 and can
* quit when at the end of a round, or, if they either have $0 or
* $1000.
*
*  INPUT:
*      char playAgain:   controls decision of player to continue playing another round
*      char anotherCard: controls decision of player to get another card on top of
their current total
```

```
*  OUTPUT:
*      struct Player player:    stores all pertinent data to user, such as balance,
bet, card/card value and more
*****************************************************************************/



//CONSTANTS && ENUMS
const int BLACKJACK_MAX = 21;
const int DEALERDRAW_LIMIT = 16;    //continue to draw cards if total is less than or
equal to 16
const int MAXPLAYER_BALANCE = 1000; //If player's balance reaches 1000, game ends

enum CardSuite {SPADES = 0, CLUBS = 1, DIAMONDS = 2, HEARTS = 3};
enum CardRank {ACE = 1, JACK = 11, QUEEN = 12, KING = 13};
enum CardValue {ACELOW_VAL = 1, JACK_VAL = 10, QUEEN_VAL = 10, KING_VAL = 10,
ACEHIGH_VAL = 11};
//CardValue and CardRank only account for cards who's values are not immediately
obvious
//EOC&&E


//STRUCT DECLERATIONS && DEFINITIONS
struct ACard
{
   int suite;      //suite of card: heart, spades, diamonds, or clubs.
   int rank;       //determines the rank, a rank of 1 is an ace, 13 is a king, 12 is a
queen
   int cardValue;  //value based on the rank, between 1 and 11 inclusive
   const char *suiteStr;
   const char *rankStr;
   bool isAce;     //Extra; Ace is special since can have both the value of 1 and 11
};

struct Player
{
   ACard cards[6];     //Can have at most 6 cards and be under 21, Ace, 2, 3, 4, 5, 6
= 21.
   int currCardIndex;  //Used to track index in ACard cards[] array.
   int cardTotal;      //Tracks total value of cards so far
   int balance;        //How much money the player has
   int bet;            //Tracks the bet from the player
   int gameEnd;        //Status variable, tracks wether player can continue playing
   int busted;         //Status variable, tracks wether the current round is done
};
//EOD&&D
```

```
//FUNCTION DECLERATIONS/PROTYPES
/*****************************************************************************
* DrawCard
*  Takes no arguments and returns a struct of type ACard. All information
*  related to a card such as, suite, rank, cardValue, suiteStr, rankStr,
*  and isAce is set here.
*
* GetPlayerACard
*  Ties in DrawCard() func with a passed in refernce to a player. Will
*  Update player's collection of cards for the round
*
* GetPlayerBet
*  Simply gets a bet from the player. If the player tries to set a negative
*  bet or a bet higher than their balance, it will ask them for a bet once more
*
* InitPlayerStruct
*  Takes in a player struct by reference, and all the necessary information
*  for the start of the game, aka the starting balance.
*
* PrintPlayerCardBetween
*  Takes in a const reference to a player, and since all the cards are
*  represented in array, will choose a range between that array to print in
*  a standard formated way
*
* PrintPlayerLastCard
*  Again takes in a const reference to a player, but no other argument
*  since it is only printing the last/newest card
*
* StartNewRound
*  Since only a single player game, only takes two player references, that
*  of the console player, and that of the dealer. User is asked for a bet
*  here, and new cards are set. Values from previous rounds are also reset
*  in order not to mix up current round and previous round developments
*
* ResetPlayerForNewRound
*  Takes in a reference to player where their currCardIndex, cardTotal, bet,
*  and busted attributes are reset.
*
* PlayerWin
*  Displays player win. Updates player's balance to add bet money
*
* PlayerLose
*  Displays player loose. Updates player's balance to remove bet money
*
* PlayerDraw
*  Displays player draw. Does not touch player's balance
*****************************************************************************/
```

```cpp
ACard DrawCard(void); //No arguments passed
void GetPlayerACard(Player &p /* IN - All of player attributes defined in struct
Player */);
void GetPlayerBet(Player &p   /* IN - All of player attributes defined in struct
Player */);
void InitPlayerStruct(Player &p /* IN - All of player attributes defined in struct
Player*/,
                     int startingBalance /*IN - Used to set player's starting
balance*/);
void PrintPlayerCardsBetween(const Player &p,/* IN - const of All of player attributes
defined in struct Player */
                            int startingAt, /* IN - Defines which card to start
printing, inclusive, from the cards array from player struct*/
                            int upTo        /* IN - Defines which card to stop at, non
inclusive.*/);
void PrintPlayerLastCard(const Player &p /* IN - const of All of player attributes
defined in struct Player */);
void StartNewRound(Player &player, /* IN - All of player attributes defined in struct
Player */
                  Player &dealer /* IN - All of player attributes defined in struct
Player */);
void ResetPlayerForNewRound(Player &player /* IN - All of player attributes defined in
struct Player */);
void PlayerWin(Player &p /* IN - All of player attributes defined in struct Player
*/);
void PlayerLose(Player &p /* IN - All of player attributes defined in struct Player
*/);
void PlayerDraw(const Player &p /* IN - const of All of player attributes defined in
struct Player */);
//EOD/P


//MAIN
int main(void)
{
   //SETUP - Code needed before we start playing, or even inputing
   Player dealer;      //PROC && TRACK - Tracks dealer's cards, and running card total
value
   Player player;      //PROC && TRACK - Same as dealer, but also uses balace && bet
&& gameEnd atributes
   char playAgain;     //TRACK         - Track if the user wants to continue playing
   char anotherCard;   //PROC && TRACK - Asks player if they want another card.

   srand(333);         //According to the assignment, set srand seed to 333

   //Init variables
   InitPlayerStruct(player, 100); //player starts with 100 dollars
   InitPlayerStruct(dealer, 100); //even though dealer does not bet, required field,
really not used though
   playAgain = 'y';              //default value of playAgain
   anotherCard = 'y';            //default value of anotherCard

   //Header and class info
   #ifdef NOT_HYPERGRADE //done to quickly compile a version with the header
   std::cout <<
"*****************************************************************************\n";
   std::cout << " * AUTHOR       : Osbaldo Gonzalez Jr.\n";
   std::cout << " * Assignment 1 : Black Jack\n";
   std::cout << " * CLASS        : CS 3A\n";
   std::cout << " * SECTION      : 71206\n";
   std::cout << " * DUE DATE     : 6/25/2021\n";
   std::cout <<
"*****************************************************************************\n";
   #endif

   //INPUT && OUTPUT && PROCESSING - with the StartNewRound function, the user will be
asked to input a bet
   //Throughout the main loop, we will also see if the player lost, won, or had a draw
aka OUTPUT
   while (playAgain == 'y' && !player.gameEnd)
   {
       //INPUT - ask player for bet, and set two new cards for player and dealer
       StartNewRound(player, dealer); //also reset player and dealer atributes from
previous round

       std::cout << "Your cards are:\n";
       PrintPlayerCardsBetween(player, 0, 2); //print the first two cards

       anotherCard = 'y';      //done to ensure loop will start, no matter past value
       while (player.cardTotal < BLACKJACK_MAX && anotherCard == 'y')
       {
           std::cout << "Your total is " << player.cardTotal
           << ". Do you want another card (y/n)?\n";
           std::cin >> anotherCard;

           if (anotherCard == 'y')
           {
               GetPlayerACard(player);
               std::cout << "You draw a:\n";
               PrintPlayerLastCard(player);
           }

           if (player.cardTotal > BLACKJACK_MAX)
           {
               player.busted = true; //will end the round "early"
               std::cout << "Your total is " << player.cardTotal
               << ". You busted!\n";
           }
       }
       std::cout << "\n\n";    //spacer


       //PROC - User does not have say in dealer card choices, just do this quickly
       std::cout << "The dealer's cards are:\n";
       PrintPlayerCardsBetween(dealer, 0, 2);
       while (!player.busted && dealer.cardTotal <= DEALERDRAW_LIMIT)
       {
           std::cout << "The dealer's total is " << dealer.cardTotal << "\n";
           GetPlayerACard(dealer);
           std::cout << "The dealer draws a card.\n";
           PrintPlayerLastCard(dealer);

           if (dealer.cardTotal > BLACKJACK_MAX)
           {
               dealer.busted = true;
               std::cout << "The dealer's total is "
               << dealer.cardTotal << "\n";
           }
       }
       std::cout << "The dealer's total is " << dealer.cardTotal << "\n";

       //PROC && OUTPUT - Figure out game outcome based on player and dealer
attributes
       if (player.busted)      //first since player gets cards first
           PlayerLose(player); //also ensures that if both the dealer and player bust,
player still loses
       else if (dealer.busted) //assured player.busted is false
           PlayerWin(player);
       else if (dealer.cardTotal > player.cardTotal) //assured that both values are <=
BLACKJACK_MAX
           PlayerLose(player);
       else if (dealer.cardTotal < player.cardTotal)
           PlayerWin(player);
       else if (dealer.cardTotal == player.cardTotal)
           PlayerDraw(player);

       std::cout << "\n"; //spacer



       //END OF ROUND OUPUT
       //Always check!
       if (player.balance <= 0 || player.balance > MAXPLAYER_BALANCE)
           player.gameEnd = true;  //end the game, will exit out of main loop
       else
       {
           std::cout << "Play again? (y/n):\n";
           std::cin >> playAgain;
       }
   }

   //FINAL OUTPUT: Display player end balance
   std::cout << "You have $" << player.balance << ". GAME OVER.\n";

   return 0;
}
//EOMain

void InitPlayerStruct(Player &p, int startingBalance)
{
   //Note, player does not get cards here, player
   //gets a a card at StartNewRound() func or in the
   //main loop in main()
   p.currCardIndex = 0;
   p.cardTotal = 0;
   p.balance = startingBalance;
   p.gameEnd = (startingBalance <= 0); //gameEnd set to true if starting balance is 0
or less
   p.busted = false;                   //busted is dependent on cardTotal value being
greater than 21
}
//EOF

void ResetPlayerForNewRound(Player &player)
{
   /* NOTE: player.cards array is not being reset, all the previous values
    * card strucutres are there. The thing done to "reset" that array is to simply
    * reset the currCardIndex back to 0. Like this, the program doesn't have
    * to unnecessarily clear out the .cards array for each player, but it is wise to
    * note that "valid" values might still be hidden there*/
   player.bet = 0;
   player.busted = false;
   player.cardTotal = 0;
   player.currCardIndex = 0;
   //player's gameEnd state remains, as well as their balance attribute
}

void StartNewRound(Player &player, Player &dealer)
{
   ResetPlayerForNewRound(player); //there are values such as busted state, cardTotal,
and currCardIndex that need to be reset
   ResetPlayerForNewRound(dealer);

   GetPlayerBet(player);

   //Every Player gets two cards at the beginning
   GetPlayerACard(player);
   GetPlayerACard(player);
   /* Note if you are expecting a specific result,
    * with say a specific srand() seed, the order
    * In which the cards are assigned to either
    * the player or dealer is important. */
   GetPlayerACard(dealer);
   GetPlayerACard(dealer);
}
//EOF


void GetPlayerBet(Player &p)
{
   do{
       std::cout << "You have $" << p.balance << ". Enter bet:\n";
       std::cin >> p.bet;
   } while(p.bet > p.balance || p.bet < 0);
}
//EOF

void PrintPlayerCardsBetween(const Player &p, int startingAt, int upTo)
{
   /* i < p.currCardIndex ensures that we don't access an index
    * that doesn't yet have a card set to it. This also ensures
    * we don't get out of bounds on the index. */
   for (int i = startingAt; i < p.currCardIndex && i < upTo; i++)
       std::cout << "   " <<  p.cards[i].rankStr
       << " of " << p.cards[i].suiteStr << "\n";
}
//EOF

void PrintPlayerLastCard(const Player &p)
{
   /* Note, currCardIndex always holds the index of the next unrealized card
    * and therefore, - 1 is used to get to the last realized card*/
   std::cout << "   " << p.cards[p.currCardIndex - 1].rankStr
   << " of " << p.cards[p.currCardIndex - 1].suiteStr << "\n";
}
//EOF
```

```
void GetPlayerACard(Player &p)
{
   p.cards[p.currCardIndex] = DrawCard();
   if (p.cards[p.currCardIndex].isAce)
   {   /* if card drawn is an Ace, and the player's total
        * running card value is less than or equal to 10
        * then set the value of the ace card to 11, else 1
        * Note: the automatic value of the Ace card is 1,
        * look at DrawCard() func to see that.*/
       p.cards[p.currCardIndex].cardValue = (p.cardTotal <= 10) ? ACEHIGH_VAL :
ACELOW_VAL;
   }

   p.cardTotal += p.cards[p.currCardIndex].cardValue;
   p.currCardIndex += 1; //get ready for the next card.
}
//EOF

ACard DrawCard(void)
{
   /* On the lifetime of the strings:
    * https://stackoverflow.com/questions/9970295/life-time-of-a-string-literal-in-c
    */
   /* extra note, static variables have a lifetime till the end of program,
    * therfore every time DrawCard is called, it is the exact same arrays.
    * this is done to reduce the number of initialization that is done at
    * every function call.*/

   static const char *SUITE_TABLE[4] =
   {"Spades", "Clubs", "Diamonds", "Hearts"};

   static const char *RANK_TABLE[14] =
   {"", "Ace", "Two", "Three", "Four", //at index 0 it is empty string since no rank =
0
   "Five", "Six", "Seven", "Eight",
   "Nine", "Ten", "Jack", "Queen", "King"};

   static const int VALUE_TABLE[14] =
   {0, ACELOW_VAL, 2, 3, 4, 5, 6, 7, 8, 9,
   10, JACK_VAL, QUEEN_VAL, KING_VAL};


   ACard temp; //PROC & RETRUN - Used for temp processing, will return value

   temp.rank = (rand()%13) + 1;        //There are 13 cards, 1 == ACE ... 13 == KING
   temp.rankStr = RANK_TABLE[temp.rank];
   temp.cardValue = VALUE_TABLE[temp.rank];
   temp.isAce = (temp.rank == ACE);     //ACE is an enum constant value globably
defined
   temp.suite = rand()%4;                //There are only four suites, each denoted a
value between 0 and 3
   temp.suiteStr = SUITE_TABLE[temp.suite];

   return temp;
}
//EOF

void PlayerWin(Player &p)
{
   std::cout << "You win $" << p.bet << ".\n";
   p.balance += p.bet;
}
//EOF

void PlayerLose(Player &p)
{
   std::cout << "Too bad.  You lose $"
   << p.bet << ".\n";
   p.balance -= p.bet;
}
//EOF

void PlayerDraw(const Player &p)
{
   std::cout << "A draw! You get back your $"
   << p.bet << ".\n";
   /* no bet to be added to p.balance since
    * bets are only accounted until result is cemented
    * that is also why I can keep the promise that
    * the function will not modify the player reference*/
}
//EOF
```