Blackjack - The Game

# 

[**Introduction 2**](#_gesoe5v5cuyo)

[**Rules of Blackjack 2**](#_eqy361vyus2o)

[**Instructions for Playing Blackjack Program 3**](#_aot7c974rds0)

[**This Program’s features 4**](#_pgps5dgoks5z)

[**Overall Program Flow 4**](#_99ias0jaqggk)

[**Flowchart 5**](#_pagqff2g4oqc)

[**Diagram 6**](#_a9qywoqzdgiw)

[**Functions in the Program 8**](#_iranoeedplvt)

[**1. getCard(): 8**](#_i7coetyxyrct)

[**2. getSuit(): 9**](#_436vxmmme9cl)

[**3. updFile(): 9**](#_4qpeuuwbkmwb)

[**4. updHand(): 9**](#_ydl3a8fqfox)

[**5. recWin(): 9**](#_hh47v1ah3wm5)

[**6. valInput(): 9**](#_g12pelreh339)

[**7. valBet(): 10**](#_wrtfuxhzh9o8)

[**8. showLead(): 10**](#_qh2d2hlieffe)

[**9. exitMsg(): 10**](#_erm5uxkenflm)

[**10. bubbleSort(): 10**](#_u3hl2foed3zv)

[**11. selectionSort(): 11**](#_yi02jelsqek4)

[**12. linearSearch(): 11**](#_eyyy1wxh51re)

[**Game Demonstration 12**](#_4fgv2by005bl)

[**Appendix-1: The Code 16**](#_s7skj4adrf24)

[**Appendix 2: GitHub Repository Information 26**](#_tk9s5y8h2xej)

[**Appendix 3: Cross Reference of Concepts 27**](#_p3mu1dy16ni5)

# 

# **Introduction**

Blackjack, or 21, is a popular card game in casinos worldwide. Blackjack was first invented in the 1700s as a royal court game played by French monarchs. Usually, Blackjack is played by around 2-7 players and is meant to be a multiplayer game.It is simple to understand yet offers a thrilling mix of strategy and luck. The objective is to have a hand value closer to 21 than the dealer without exceeding 21. This write-up covers the rules of Blackjack, instructions for playing the game, and details about the project implementation.



Credits for picture - <https://github.com/arinda1/blackjack-qlearning-rl>

# **Rules of Blackjack**

1. Card Values:
   * Number cards (2-10) are worth their face value.
   * Face cards (Jack, Queen, King) are worth 10 points each.
   * Aces can be worth 1 or 11 points, depending on which value benefits the hand the most. In this Program, we are setting it to 11 points.
2. Gameplay:
   * The game is played with one or more standard 52-card decks.
   * Each player is dealt two cards, and the dealer is dealt two cards (one face up and one face down).
   * Players can see their cards and the dealer's face-up card.
3. Player Options:
   * Hit: Draw a card from the deck. Players can continue to hit as many times as they like unless they exceed 21, which is called a "bust."
   * Stand: Keep the current hand and end the turn.
4. Dealer's Turn:
   * After all players have completed their turns, the dealer reveals the face-down card.
   * The dealer must hit until the hand value is at least 17.
   * If the dealer busts (exceeds 21), all remaining players win.
5. Winning the Game:
   * If the player's hand value is closer to 21 than the dealer's hand value without busting, the player wins.
   * The dealer wins if the dealer's hand value is closer to 21 than the player's hand value without busting.
   * A tie is a tie if the player and the dealer have the same hand value.

# **Instructions for Playing Blackjack Program**

1. Starting the Game:
   * Enter player names and bets for each player.
   * Deal two cards to each player and the dealer. Display the player's cards and the dealer's face-up card.
2. Player's Turn:
   * Players decide whether to "hit" or "stand" based on their hand value and the dealer's face-up card.
   * If a player chooses to hit, draw a card and add it to the player's hand value.
   * If a player busts, they lose immediately.
   * If a player chooses to stand, their turn ends and the next player takes their turn.
3. Dealer's Turn:
   * After all players have completed their turns, the dealer reveals the face-down card.
   * The dealer must hit until the hand value is at least 17.
   * If the dealer busts, all remaining players win.
4. Determining the Winner:
   * Compare each player's hand value with the dealer's hand value.
   * If the player's hand value is closer to 21 without busting, the player wins.
   * The dealer wins if the dealer's hand value is closer to 21 without busting.
   * If the player and the dealer have the same hand value, it is a tie (push).

# 

# **This Program’s features**

* We are handling multiple players (1 to 8).
* Random cards dealing with appropriate values and suits.
* Player decisions to hit or stand.
* Dealer’s automatic decision to hit or stand based on predefined rules.
* Determination of winners and updating of their balances and total winnings.
* Recording game results in a file.
* Sorting and displaying player scores and winnings.
* Searching for player details by name.

# **Overall Program Flow**

**Initialization**:

* + Seed the random number generator.
  + Prompt the user for the number of players and their names.
  + Each player places a bet.

**Card Dealing**:

* + Deal two cards to each player and the dealer.
  + Display the initial hands of all players and the dealer.

**Player Turns**:

* + Each player decides to hit or stand.
  + Update player hands based on their decisions.

**Dealer Turn**:

* + Dealer hits until their hand value reaches at least 17.

**Determine Winners**:

* + Compare the player's hands to the dealer's hand.
  + Update balances and winnings.
  + Record results in a file.

**Display Results**:

* + Show leaderboard.
  + Sort and display player totals and winnings.
  + Search for specific player details.

**End Game**:

* + Display final results and exit the program.

# **Flowchart**

Initialize random seed

Open output file for appending

Prompt user for number of players (1-8)

Validate the number of players

For each player:

Prompt for player name

Prompt for bet amount

Validate bet amount

Deal two cards to each player and the dealer

Display initial hands

For each player:

While player chooses to hit:

Deal a new card

Update and display hand

If player busts, break

Dealer hits until total is at least 17

Display dealer's hand

Determine winners:

Compare each player's hand to dealer's hand

Update balances and winnings

Record results in file

Display leaderboard

Sort and display player totals (bubble sort)

Sort and display player winnings (selection sort)

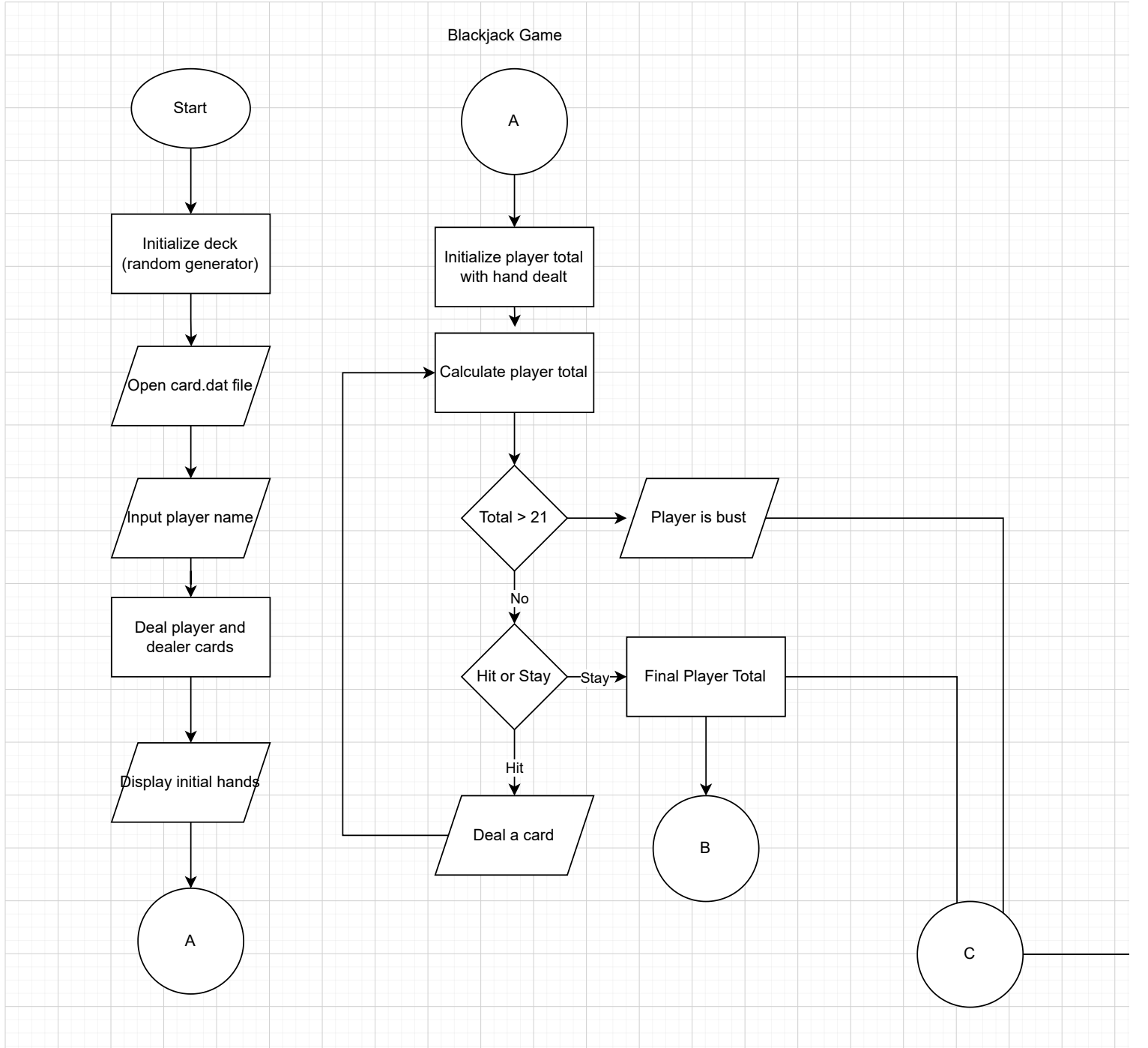
Search for a player by name and display their details

Close the output file

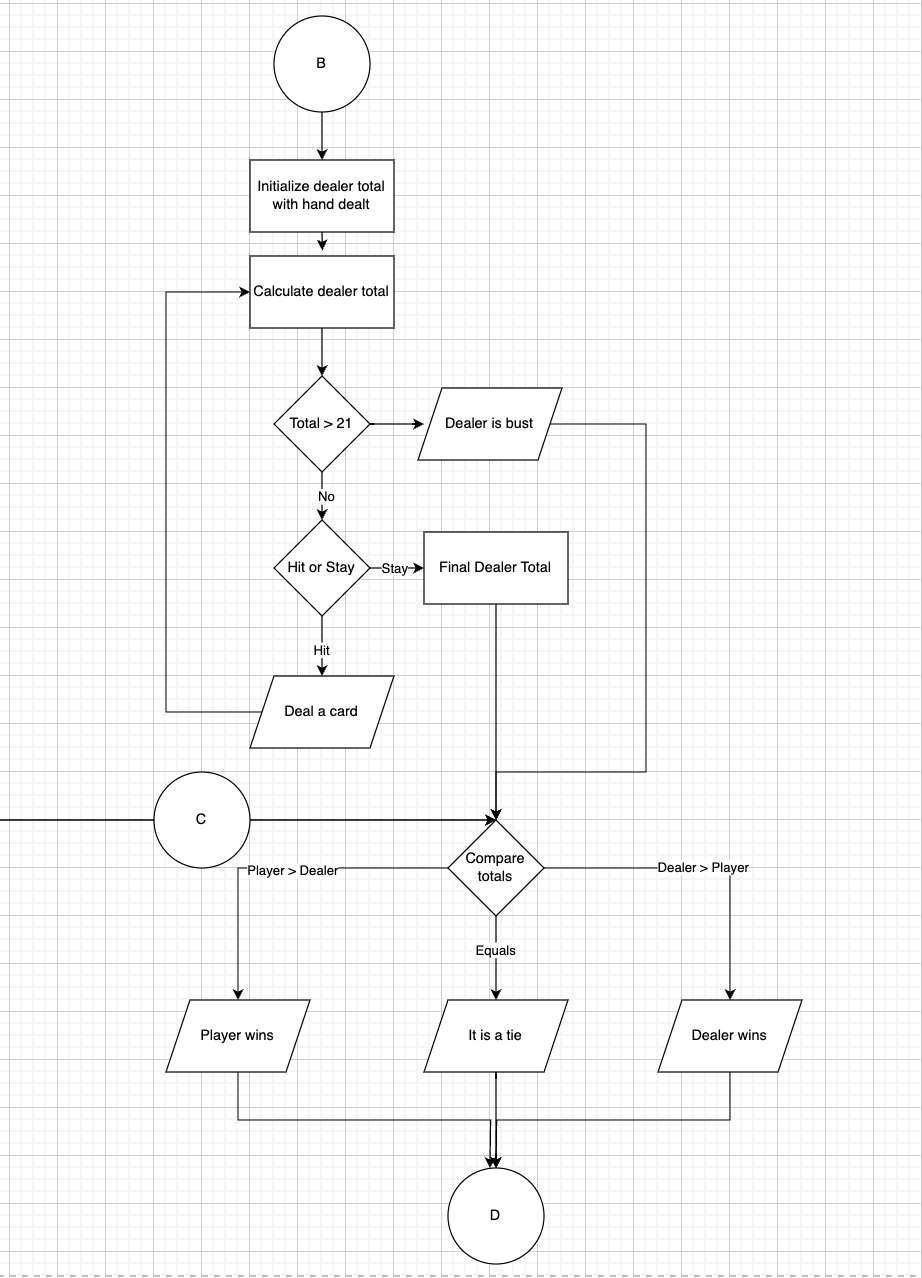
Exit program

## **Diagram**

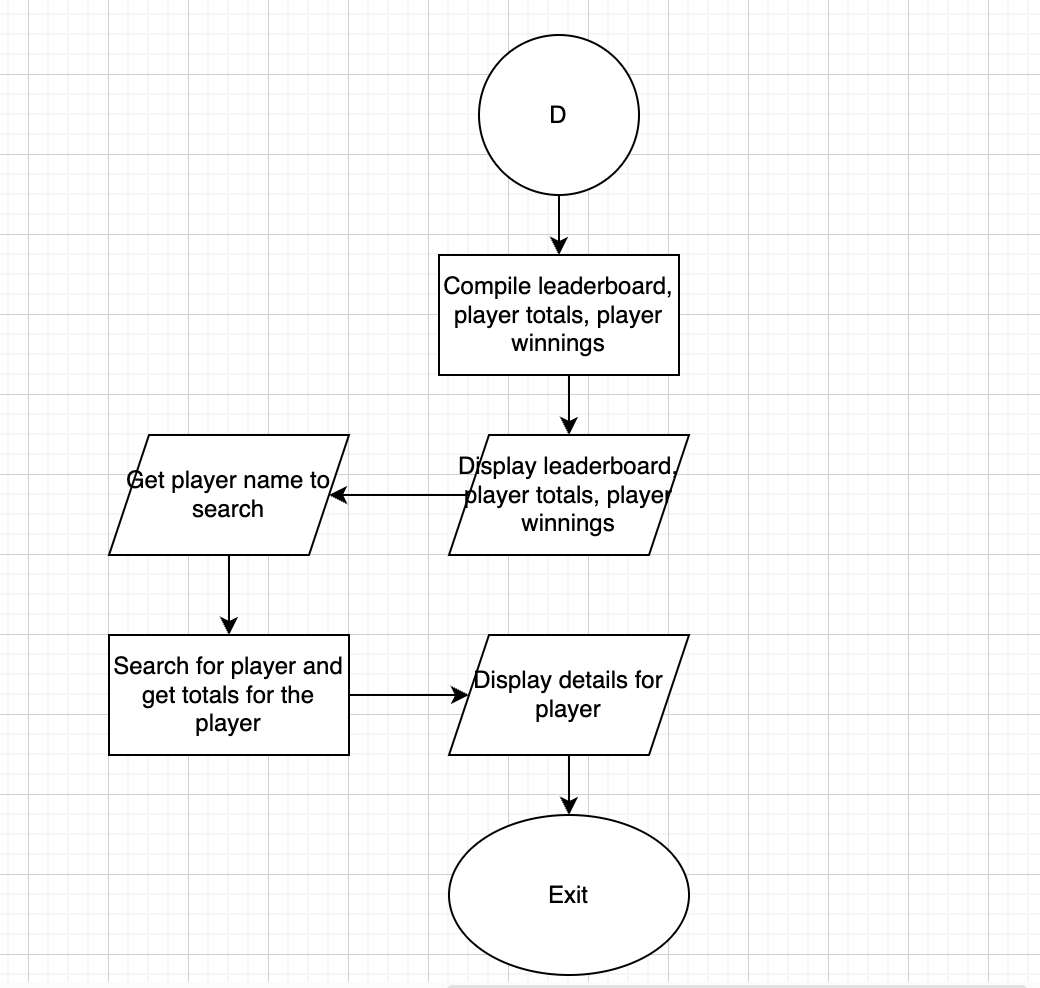
1. Initial sections of the game



1. Middle sections of the game



1. End of the game



# **Functions in the Program**

## **getCard()**:

**Flow**: Generates a random card value between 1 and 13. Face cards are converted to 10, and Aces are treated as 11.

Generate a random number between 1 and 13

If number > 10, return 10

If number == 1, return 11

Otherwise, return the number

## **getSuit()**:

**Flow**: Generates a random suit from the four possible suits (Hearts, Diamonds, Clubs, Spades).

Generate a random number between 0 and 3

If 0, return "Hearts"

If 1, return "Diamonds"

If 2, return "Clubs"

If 3, return "Spades"

## **updFile()**:

**Flow**: Updates the file with card details, recording the player’s card.

If file is open

Write player name, card value, and suit to the file

## **updHand()**:

**Flow**: Updates the hand array and total for the given player with the new card details.

If indices are within bounds

Add card value to player's hand total

Append card value and suit to the player's hand suit string

## **recWin()**:

**Flow**: Records the winner’s details in the file.

If file is open

Write player or dealer's hand and total to the file

Write the winner's name to the file

## **valInput()**:

**Flow**: Validates the user input for hitting or standing.

While input is invalid

Clear input buffer

Prompt for valid input

## **valBet()**:

**Flow**: Validates the bet amount input, ensuring it is a positive number.

While bet is invalid

Clear input buffer

Prompt for valid bet amount

## **showLead()**:

**Flow**: Displays the leaderboard showing player winnings.

For each player

If player name is not empty

Display player's name and winnings

## **exitMsg()**:

**Flow**: Displays an exit message and terminates the program.

Display the message  
Exit the program

## **bubbleSort()**:

**Flow**: Sorts the players’ scores in descending order using the bubble sort algorithm.

Do

Set swapped to false

For each pair of adjacent elements

If first element is less than the second

Swap the elements

Set swapped to true

While swapped is true

## **selectionSort()**:

**Flow**: Sorts the players’ winnings in descending order using the selection sort algorithm.

For each element in the array

Assume the current element is the largest

For each of the remaining elements

If element is larger than the assumed largest

Update the largest element index

Swap the current element with the largest element found

## **linearSearch()**:

**Flow**: Searches for a player’s name in the list and returns the index if found.

For each name in the list

If name matches the target

Return the index

Return -1 if not found

# 

# **Game Demonstration**

| Players register and place bets |  |
| --- | --- |
| All players are dealt 2 cards and the dealer is also dealt 2 cards.  Each player is asked to hit or stand |  |
| If a player stands, their total is locked.  If a player hits, another card is dealt and the total is checked. Player is busted if total exceeds 21 |  |
| When all players stand or their total exceeds 21, the game is completed, dealers final total is shown and winner is declared. |  |
| Leaderboard and player totals are shown |  |
| Player winnings in a sorted order is shown |  |
| Search for a player |  |

# 

## **Appendix-1: The Code**

| /\*  \* File: main.cpp  \* Author: Akshay Balaji  \* Created: 7/21/2024  \* Purpose: Blackjack game  \*/  // System Libraries  #include <iostream>  #include <cstdlib>  #include <ctime>  #include <string>  #include <fstream>  #include <vector>  #include <limits>  using namespace std;  // User Libraries  // Global Constants, no Global Variables are allowed  // Math/Physics/Conversions/Higher Dimensions - i.e. PI, e, etc...  const int MAX\_SCORE = 21;  const int DEALER\_LIMIT = 17;  const int MIN\_PLAYERS = 1;  const int MAX\_PLAYERS = 8;  const int MAX\_CARDS = 10;  // Structure to hold player information  struct Player {  string name;  int total;  int wnngs;  };  // Function Prototypes  int getCard(); // Get a random card value  string getSuit(); // Get a random card suit  void updFile(ofstream&, const string&, int, const string& = "Unknown"); // Update file with card details  void updHand(int[][MAX\_CARDS], int[], int, int, int, vector<string>&, const string&); // Update hand with card details  void recWin(ofstream&, const vector<string>&, int[][MAX\_CARDS], int[], int, const string&); // Record the winner in the file  void valInput(char&); // Validate user input for hit or stand  void valBet(int&); // Validate bet input  void showLead(const vector<Player>&, int); // Display the leaderboard  void exitMsg(const string& msg); // Exit function with a message  void bubbleSort(vector<Player>&); // Bubble sort for player scores  void selectionSort(vector<Player>&); // Selection sort for player winnings  int linearSearch(const vector<string>&, const string&); // Linear search for player names  int getCard(int multiplier); // Overloaded function to get a random card value with a multiplier  // Execution Begins Here!  int main() {  // Declare Variables  int nPlyrs;  vector<string> pNames; // Declare vector without initializing size  int pBets[MAX\_PLAYERS];  int pBlns[MAX\_PLAYERS] = {0}; // Player balances initialized to 0  int tWngs[MAX\_PLAYERS + 1] = {0}; // Total wnngs (last index for dealer)  vector<string> winnrs(MAX\_PLAYERS + 1); // Names of winners  int pTotals[MAX\_PLAYERS + 1] = {0}; // +1 for the dealer  int dTotal;  int dCard1, dCard2;  string dSuit1, dSuit2;    int hands[MAX\_PLAYERS + 1][MAX\_CARDS] = {0}; // +1 for the dealer  vector<string> hSuits; // Vector for hand suits based on number of players  srand(static\_cast<unsigned int>(time(0)));  ofstream outFile("card.dat", ios::app);  if (!outFile.is\_open()) {  exitMsg("Error opening file!");  }  while (true) {  cout << "Enter number of players (1-8): ";  cin >> nPlyrs;    // Check if the input is an integer  if (cin.fail()) {  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  cout << "Invalid input. Please enter a number between 1 and 8." << endl;  } else if (nPlyrs < MIN\_PLAYERS || nPlyrs > MAX\_PLAYERS) {  cout << "Invalid number of players. Enter a number between 1 and 8." << endl;  } else {  break; // Valid input  }  }    pNames.resize(nPlyrs); // Resize vector to the number of players  vector<Player> players(nPlyrs); // Vector of Player structures  hSuits.resize(nPlyrs + 1); // Resize handSuits for the number of players and one for the dealer  for (int i = 0; i < nPlyrs; ++i) {  cout << "Enter player " << i + 1 << "'s name: ";  cin.ignore();  getline(cin, pNames[i]);  players[i].name = pNames[i];  cout << "Enter bet amount for " << pNames[i] << ": ";  cin >> pBets[i];  valBet(pBets[i]);  }    for (int i = 0; i < nPlyrs; ++i) {  // Deal two cards to each player  for (int j = 0; j < 2; ++j) {  int crd = getCard();  string sut = getSuit();    updHand(hands, pTotals, i, j, crd, hSuits, sut);  }  players[i].total = pTotals[i];  cout << pNames[i] << "'s Hand: " << hSuits[i] << endl;  }  // Initial card dealing for the dealer  dCard1 = getCard();  dSuit1 = getSuit();  updHand(hands, pTotals, nPlyrs, 0, dCard1, hSuits, dSuit1);  dCard2 = getCard();  dSuit2 = getSuit();  updHand(hands, pTotals, nPlyrs, 1, dCard2, hSuits, dSuit2);  dTotal = dCard1 + dCard2;  cout << "Dealer's Hand: " << hSuits[nPlyrs] << endl;  // Player's Turn: Each player can hit or stand  for (int i = 0; i < nPlyrs; ++i) {  char choice;  int cardIndex = 2;  do {  cout << pNames[i] << "'s total: " << pTotals[i] << endl;  cout << "Do you want to hit (h/H) or stand (s/S)? ";  cin >> choice;  valInput(choice);  if (choice == 'h' || choice == 'H') {  int newCrd = getCard();  string newSut = getSuit();  if (cardIndex < MAX\_CARDS) {  updHand(hands, pTotals, i, cardIndex, newCrd, hSuits, newSut);  cardIndex++;  cout << "You got " << newCrd << " of " << newSut << endl;  } else {  cout << "Maximum cards reached for " << pNames[i] << endl;  break;  }  if (pTotals[i] > MAX\_SCORE) {  cout << pNames[i] << " busts with total: " << pTotals[i] << endl;  break;  }  }  } while (choice == 'h' || choice == 'H');  players[i].total = pTotals[i];  }  // Dealer's Turn: Dealer hits until reaching DEALER\_LIMIT  int dealerCardIndex = 2;  while (dTotal < DEALER\_LIMIT && dealerCardIndex < MAX\_CARDS) {  int newCrd = getCard();  string newSut = getSuit();  dTotal += newCrd;  cout << "Dealer got " << newCrd << " of " << newSut << endl;  updHand(hands, pTotals, nPlyrs, dealerCardIndex, newCrd, hSuits, newSut);  dealerCardIndex++;  }  cout << "Dealer's total: " << dTotal << endl;  string bPlyr = "";  int bTotal = 0;  // Determine winners  for (int i = 0; i < nPlyrs; ++i) {  if (pTotals[i] > MAX\_SCORE) {  cout << pNames[i] << " busts and cannot win!" << endl;  tWngs[nPlyrs] += pBets[i];  winnrs[nPlyrs] = "Dealer";  } else if (dTotal > MAX\_SCORE || pTotals[i] > dTotal) {  cout << pNames[i] << " wins!" << endl;  recWin(outFile, pNames, hands, pTotals, i, pNames[i]);  tWngs[i] += pBets[i] \* (nPlyrs + 1);  pBlns[i] += pBets[i] \* nPlyrs;  winnrs[i] = pNames[i];  players[i].wnngs = pBets[i] \* (nPlyrs + 1);  if (pTotals[i] > bTotal) {  bTotal = pTotals[i];  bPlyr = pNames[i];  }  } else if (dTotal > pTotals[i]) {  cout << "Dealer wins against " << pNames[i] << "!" << endl;  recWin(outFile, pNames, hands, pTotals, i, "Dealer");  tWngs[nPlyrs] += pBets[i];  winnrs[nPlyrs] = "Dealer";  } else {  cout << "It's a tie for " << pNames[i] << "!" << endl;  recWin(outFile, pNames, hands, pTotals, i, "Tie");  winnrs[i] = pNames[i];  }  if (bTotal > 0)  cout << "Best Player: " << bPlyr << " Score: " << bTotal << endl;  }  // Record dealer's final hand  recWin(outFile, pNames, hands, pTotals, nPlyrs, "Dealer");  // Close the card.dat file  outFile.close();  // Show Leaderboard  showLead(players, nPlyrs);  // Sort player totals using bubble sort  bubbleSort(players);  // Display sorted player totals  cout << "\nSorted Player Totals (Bubble Sort):" << endl;  for (int i = 0; i < nPlyrs; ++i) {  cout << players[i].name << ": " << players[i].total << endl;  }  // Sort player wnngs using selection sort  selectionSort(players);  // Display sorted wnngs  cout << "\nSorted Player Winnings (Selection Sort):" << endl;  for (int i = 0; i < nPlyrs; ++i) {  cout << players[i].name << ": " << players[i].wnngs << " points" << endl;  }  // Display player balances  for (int i = 0; i < nPlyrs; ++i) {  cout << pNames[i] << "'s balance: " << pBlns[i] << endl;  }  // Search for a player by name  string srchNm;  cout << "\nEnter player name to search: ";  cin.ignore();  getline(cin, srchNm);  int srchIdx = linearSearch(pNames, srchNm);  if (srchIdx != -1) {  cout << srchNm << " found with total: " << pTotals[srchIdx] << endl;  } else {  cout << srchNm << " not found." << endl;  }  return 0;  }  // Static variable to count number of cards dealt  static int cardCount = 0;  // Get a random card value  int getCard() {  cardCount++;  // get a random card  int card = rand() % 13 + 1;  // all face cards get value 10  if (card > 10) return 10;  // Ace gets value 11  if (card == 1) return 11;  // return card value otherwise  return card;  }  // Overloaded function to get a random card value with a multiplier  int getCard(int multiplier) {  cardCount++;  int card = (rand() % 13 + 1) \* multiplier;  if (card > 10 \* multiplier) return 10 \* multiplier;  if (card == 1 \* multiplier) return 11 \* multiplier;  return card;  }  // Get a random card suit  string getSuit() {  // get a random suite  int suit = rand() % 4;  switch (suit) {  case 0: return "Hearts";  case 1: return "Diamonds";  case 2: return "Clubs";  case 3: return "Spades";  }  return "";  }  // Update file with card details  void updFile(ofstream &file, const string &plyr, int val, const string &suit) {  file << plyr << ": " << val << " of " << suit << endl;  }  // Record the winner in the file  void recWin(ofstream &file, const vector<string> &pNames, int hands[][MAX\_CARDS], int pTotals[], int idx, const string &wnr) {  if (file.is\_open()) {  // Write the accumulated hand for the player or dealer  if (idx < pNames.size()) {  file << pNames[idx] << ": ";  } else {  file << "Dealer: ";  }  for (int i = 0; i < MAX\_CARDS; ++i) {  if (hands[idx][i] == 0) break;  file << hands[idx][i] << " ";  }  file << "\nTotal: " << pTotals[idx] << endl;  file << "Winner: " << wnr << endl;  } else {  ofstream outFile("card.dat", ios::app);  if (idx < pNames.size()) {  outFile << pNames[idx] << ": ";  } else {  outFile << "Dealer: ";  }  for (int i = 0; i < MAX\_CARDS; ++i) {  if (hands[idx][i] == 0) break;  outFile << hands[idx][i] << " ";  }  outFile << "\nTotal: " << pTotals[idx] << endl;  outFile << "Winner: " << wnr << endl;  outFile.close();  }  }  // Validate user input for hit or stand  void valInput(char &choice) {  while (cin.fail() || (choice != 'h' && choice != 'H' && choice != 's' && choice != 'S')) {  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  cout << "Invalid choice. Please enter 'h/H' to hit or 's/S' to stand: ";  cin >> choice;  }  }  // Update hands in array  void updHand(int hands[][MAX\_CARDS], int pTotals[], int idx, int cardIdx, int crd, vector<string> &hSuits, const string &suit) {  if (idx >= MAX\_PLAYERS + 1 || cardIdx >= MAX\_CARDS) {  cout << "Index out of bounds. Skipping hand update." << endl;  return;  }  hands[idx][cardIdx] = crd;  pTotals[idx] += crd;  if (hSuits[idx].empty()) {  hSuits[idx] = to\_string(crd) + " of " + suit;  } else {  hSuits[idx] += ", " + to\_string(crd) + " of " + suit;  }  }  // Validate bet input  void valBet(int &bet) {  while (cin.fail() || bet <= 0) {  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  cout << "Invalid bet amount. Please enter a positive number: ";  cin >> bet;  }  }  // Display the leaderboard  void showLead(const vector<Player> &players, int nPlyrs) {  cout << "\nLeaderboard:" << endl;  for (int i = 0; i < nPlyrs; ++i) {  if (!players[i].name.empty()) {  cout << players[i].name << ": " << players[i].wnngs << " points" << endl;  }  }  }  // Exit function with a message  void exitMsg(const string &msg) {  cout << msg << endl;  exit(1);  }  // Bubble sort for player scores  void bubbleSort(vector<Player> &players) {  bool swapped;  do {  swapped = false;  for (size\_t i = 0; i < players.size() - 1; ++i) {  if (players[i].total < players[i + 1].total) {  swap(players[i], players[i + 1]);  swapped = true;  }  }  } while (swapped);  }  // Selection sort for player wnngs  void selectionSort(vector<Player> &players) {  for (size\_t i = 0; i < players.size() - 1; ++i) {  size\_t maxIdx = i;  for (size\_t j = i + 1; j < players.size(); ++j) {  if (players[j].wnngs > players[maxIdx].wnngs) {  maxIdx = j;  }  }  swap(players[i], players[maxIdx]);  }  }  // Linear search for player names  int linearSearch(const vector<string> &names, const string &target) {  for (size\_t i = 0; i < names.size(); ++i) {  if (names[i] == target) {  return i;  }  }  return -1; // Not found  } |
| --- |

## 

## **Appendix 2: GitHub Repository Information**

* Repository:<https://github.com/abalaji05/cis-5>
* Folder: Blackjack
* Code Line Count: 408 lines (excluding comments and blank lines)

## 

## **Appendix 3: Cross Reference of Concepts**

| **Chapter** | **Section** | **Topic** | **Line Numbers** | **Notes** |
| --- | --- | --- | --- | --- |
| 6 | 3 | Function Prototypes | 40, 41, 42, 43, 44 | Always use prototypes |
| 6 | 5 | Pass by Value | 254, 258, 263, 271, 278 | Functions with value passing |
| 6 | 8 | return | 254, 258, 263, 271, 278 | Returning a value from a function |
| 6 | 9 | returning boolean | 244, 245, 246, 247, 248 | Function validation |
| 6 | 10 | Global Variables | N/A | Not used, following project rules |
| 6 | 11 | static variables | 554, 555 | Used to count the number of cards dealt |
| 6 | 12 | defaulted arguments | 128, 129, 130, 131, 132 | Default arguments in updFile |
| 6 | 13 | pass by reference | 255, 262, 271, 298, 325 | Passing vectors and arrays by reference |
| 6 | 14 | overloading | 574, 581 | Overloaded getCard function |
| 6 | 15 | exit() function | 98 | Used to exit on file open failure |
| 7 | 1 to 6 | Single Dimensioned Arrays | 111, 112, 113, 114, 115 | Arrays for player hands and totals |
| 7 | 7 | Parallel Arrays | 82, 83, 84, 85, 86 | Parallel arrays for player data |
| 7 | 8 | Single Dimensioned as Function Args | 126, 135, 137, 172, 177 | Arrays passed to functions |
| 7 | 9 | 2 Dimensioned Arrays | 111, 112, 113, 114, 115 | 2D array for card hands |
| 7 | 12 | STL Vectors | 79, 80, 81, 82, 83 | Vectors for dynamic player data |
| 7 |  | Passing Arrays to and from Functions | 126, 135, 137, 172, 177 | Arrays passed to functions |
| 7 |  | Passing Vectors to and from Functions | 255, 262, 271, 298, 325 | Passing vectors by reference |
| 8 | 3 | Bubble Sort | 481, 482, 483, 484, 485 | Bubble sort for player totals |
| 8 | 3 | Selection Sort | 503, 504, 505, 506, 507 | Selection sort for player winnings |
| 8 | 1 | Linear or Binary Search | 535, 536, 537, 538, 539 | Linear search for player names |