

# Blackjack - The Game

<b>Introduction.....</b>	<b>2</b>
<b>Rules of Blackjack.....</b>	<b>2</b>
<b>Instructions for Playing Blackjack Program.....</b>	<b>3</b>
<b>This Program's features.....</b>	<b>4</b>
<b>Overall Program Flow.....</b>	<b>4</b>
<b>Flowchart.....</b>	<b>5</b>
<b>Diagram.....</b>	<b>6</b>
<b>Functions in the Program.....</b>	<b>8</b>
1. <b>getCard():.....</b>	<b>8</b>
2. <b>getSuit():.....</b>	<b>9</b>
3. <b>updFile():.....</b>	<b>9</b>
4. <b>updHand():.....</b>	<b>9</b>
5. <b>recWin():.....</b>	<b>9</b>
6. <b>valInput():.....</b>	<b>9</b>
7. <b>valBet():.....</b>	<b>10</b>
8. <b>showLead():.....</b>	<b>10</b>
9. <b>exitMsg():.....</b>	<b>10</b>
10. <b>bubbleSort():.....</b>	<b>10</b>
11. <b>selectionSort():.....</b>	<b>11</b>
12. <b>linearSearch():.....</b>	<b>11</b>
<b>Game Demonstration.....</b>	<b>12</b>
<b>Appendix-1: The Code.....</b>	<b>16</b>
<b>Appendix 2: GitHub Repository Information.....</b>	<b>26</b>
<b>Appendix 3: Cross Reference of Concepts.....</b>	<b>27</b>

## 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)

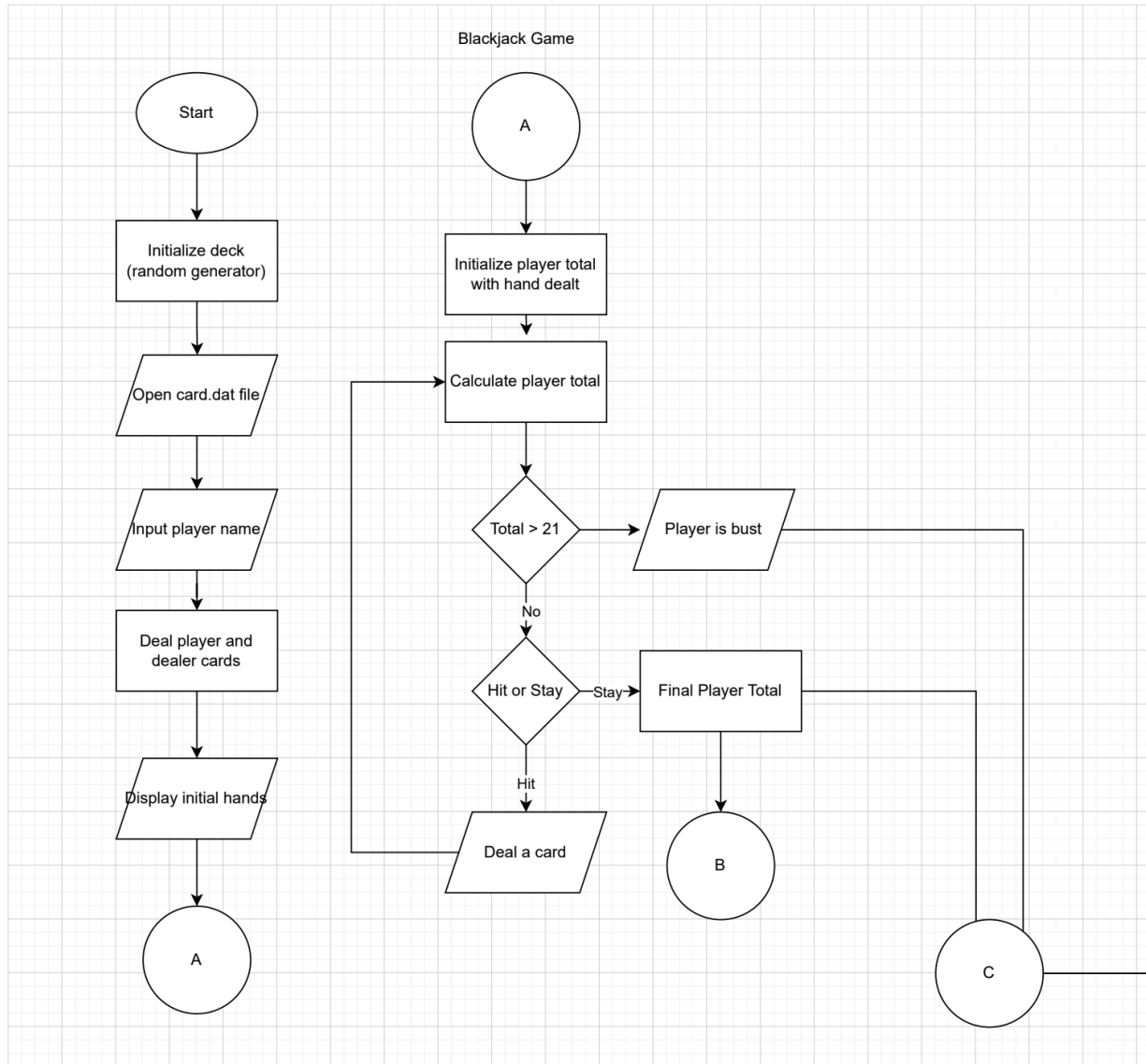
Search for a player by name and display their details

Close the output file

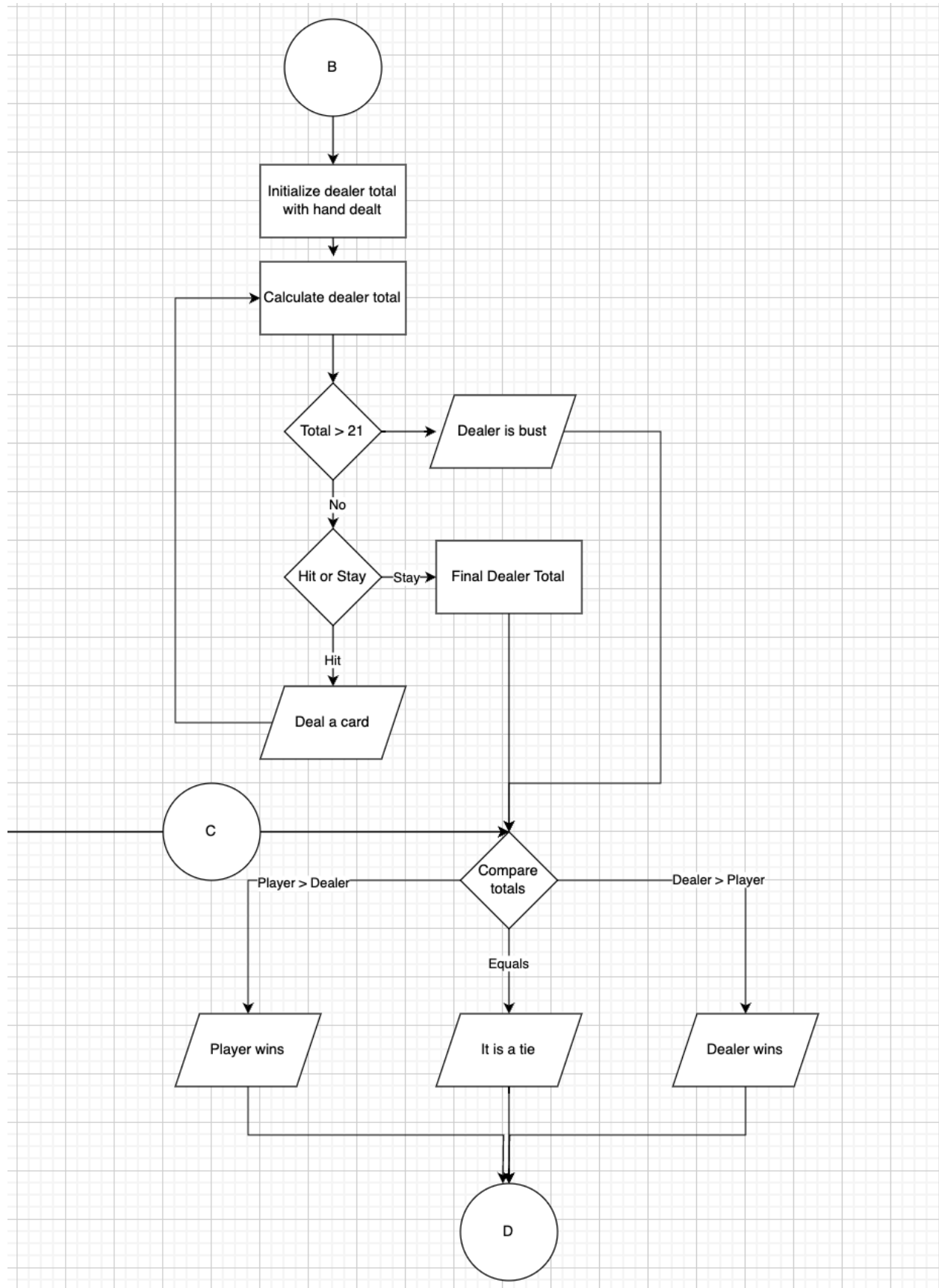
Exit program

## Diagram

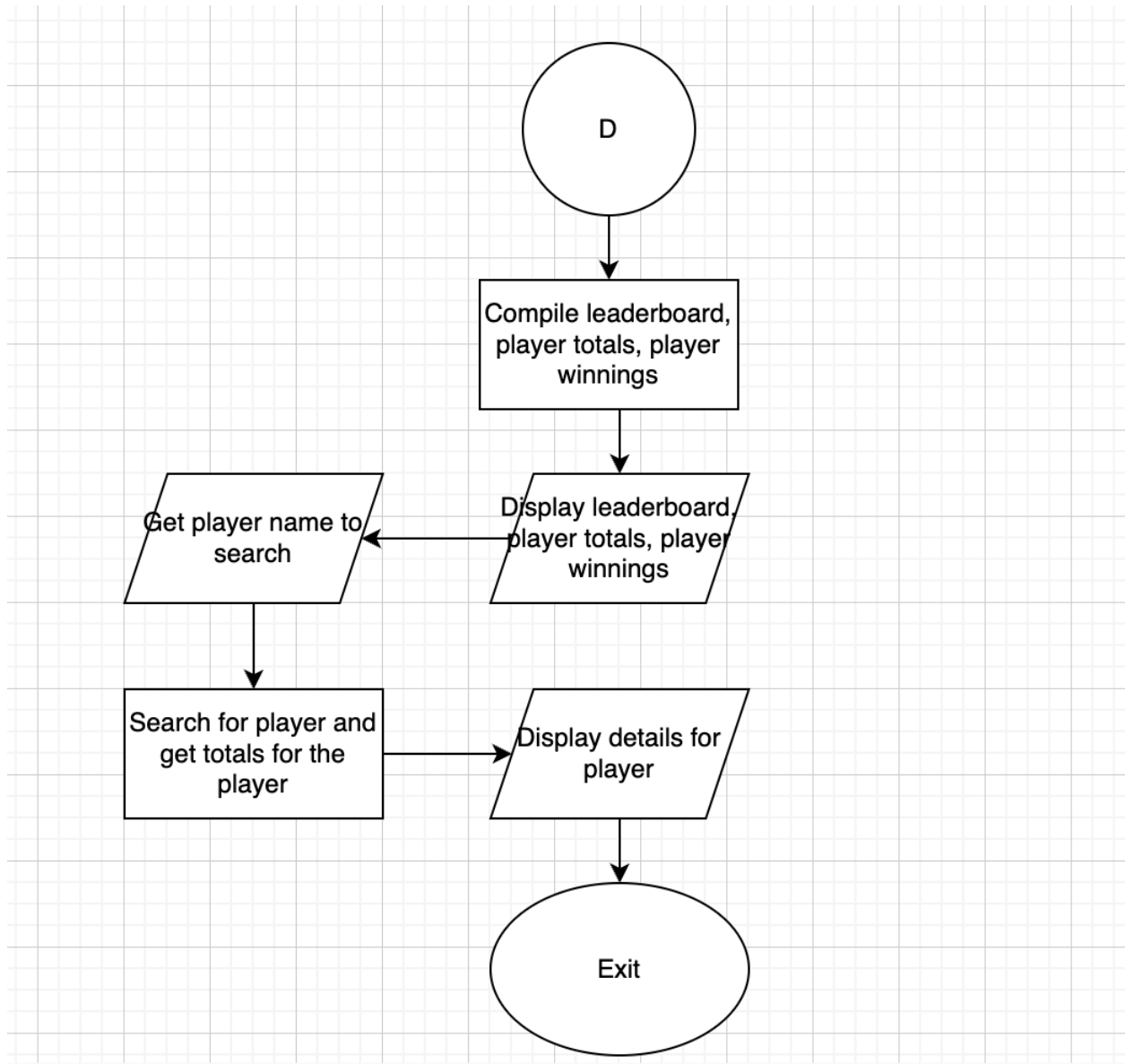
### 1. Initial sections of the game



### 2. Middle sections of the game



### 3. End of the game



## Functions in the Program

### 1. **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

## 2. **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"

## 3. **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

## 4. **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

## 5. **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

## 6. **valInput():**

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

- While input is invalid
  - Clear input buffer
  - Prompt for valid input

## 7. **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

## 8. **showLead():**

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

- For each player
  - If player name is not empty
    - Display player's name and winnings

## 9. **exitMsg():**

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

- Display the message
- Exit the program

## 10. **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

## 11. **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
```

## 12. **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

```

Blackjack - Apache NetBeans IDE 14
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help Search (Ctrl+I)
Debug 198.4/641.0MB
Projects Services Output
Blackjack
  Header Files
  Resource Files
  Source Files
  Test Files
  Important Files
main() - Navigator
  <No View Available>
Blackjack (Build, Run) x Blackjack (Run) x
Enter number of players (1-8): 3
Enter player 1's name: Akshay
Enter bet amount for Akshay: 10
Enter player 2's name: John
Enter bet amount for John: 10
Enter player 3's name: Scott
Enter bet amount for Scott: 10
Blackjack (Run) x INS

```

All players are dealt 2 cards and the dealer is also dealt 2 cards.

Each player is asked to hit or stand

```

Blackjack - Apache NetBeans IDE 14
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help Search (Ctrl+I)
Debug 324.4/641.0MB
Projects Services Output
Blackjack
  Header Files
  Resource Files
  Source Files
  Test Files
  Important Files
main() - Navigator
  <No View Available>
Blackjack (Build, Run) x Blackjack (Run) x
Enter number of players (1-8): 3
Enter player 1's name: Akshay
Enter bet amount for Akshay: 10
Enter player 2's name: John
Enter bet amount for John: 10
Enter player 3's name: Scott
Enter bet amount for Scott: 10
Akshay's Hand : 10 of Clubs, 10 of Spades
John's Hand : 3 of Hearts, 10 of Hearts
Scott's Hand : 10 of Diamonds, 7 of Spades
Dealer's Hand: 8 of Hearts, 10 of Diamonds
Akshay's total: 20
Do you want to hit (h/H) or stand (s/S)? 
Blackjack (Run) x INS

```

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

```

Blackjack - Apache NetBeans IDE 14
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help Search (Ctrl+I)
Debug 415.9/641.0MB
Projects Services Output
Blackjack (Build, Run) x Blackjack (Run) x
> Enter number of players (1-8): 3
> Enter player 1's name: Akshay
> Enter bet amount for Akshay: 10
> Enter player 2's name: John
> Enter bet amount for John: 10
> Enter player 3's name: Scott
> Enter bet amount for Scott: 10
Akshay's Hand : 10 of Clubs, 10 of Spades
John's Hand : 3 of Hearts, 10 of Hearts
Scott's Hand : 10 of Diamonds, 7 of Spades
Dealer's Hand: 8 of Hearts, 10 of Diamonds
Akshay's total: 20
Do you want to hit (h/H) or stand (s/S)? s
John's total: 13
Do you want to hit (h/H) or stand (s/S)? h
You got 9 of Hearts
John busts with total: 22
Scott's total: 17
Do you want to hit (h/H) or stand (s/S)? █
Blackjack (Run) x INS

```

When all players stand or their total exceeds 21, the game is completed, dealers final total is shown and winner is declared.

```

Blackjack - Apache NetBeans IDE 14
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Debug 387.9/641.0MB
Projects Services Output Set Project Configuration
Blackjack (Build, Run) x Blackjack (Run) x
> Sean's total: 18
> Do you want to hit (h/H) or stand (s/S)? s
> Roger's total: 16
> Do you want to hit (h/H) or stand (s/S)? s
> Aaron's total: 18
> Do you want to hit (h/H) or stand (s/S)? s
Dealer got 6 of Clubs
Dealer's total: 20
Dealer wins against Akshay!
Dealer wins against John!
It's a tie for Scott!
Ben wins!
Best Player : Ben Score :21
Dealer wins against Bill!
Best Player : Ben Score :21
Dealer wins against Sean!
Best Player : Ben Score :21
Dealer wins against Roger!
Best Player : Ben Score :21

```

Leaderboard and player totals are shown

```

Blackjack - Apache NetBeans IDE 14
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Debug 270.6/641.0MB
Projects Services Output
Blackjack
  > Header Files
  > Resource Files
  > Source Files
  > Test Files
  > Important Files
main() - Navigator
Blackjack (Build, Run) x Blackjack (Run) x
Leaderboard:
Akshay: 0 points
John: 0 points
Scott: 0 points
Ben: 90 points
Bill: 0 points
Sean: 0 points
Roger: 0 points
Aaron: 0 points

Sorted Player Totals (Bubble Sort):
Ben's total: 21
Scott's total: 20
Akshay's total: 19
Sean's total: 18
Aaron's total: 18
Roger's total: 16
Bill's total: 14

```

Player winnings in a sorted order is shown

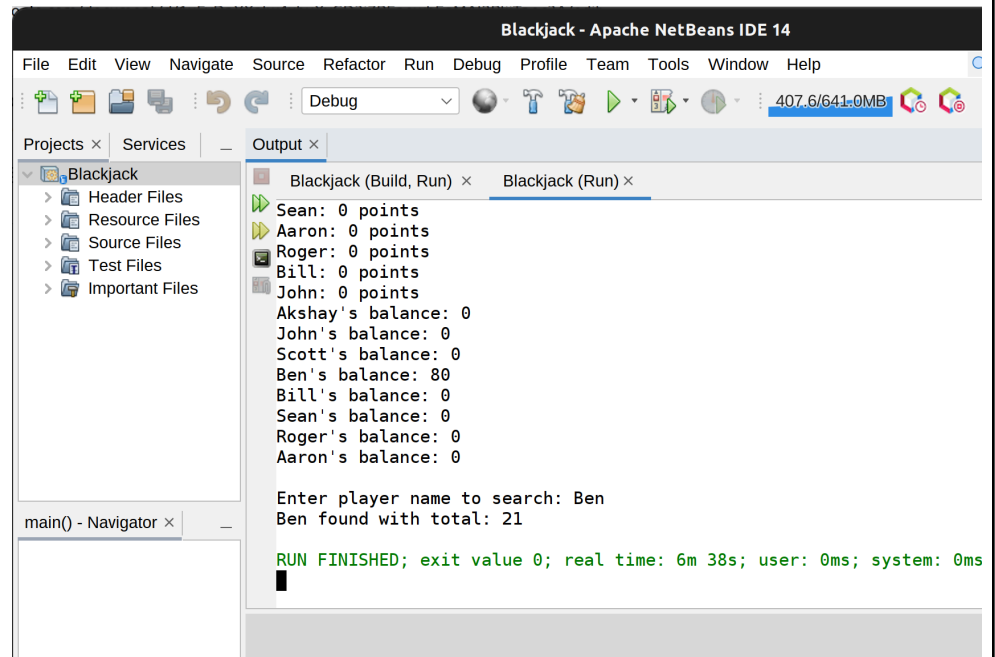
```

Blackjack - Apache NetBeans IDE 14
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Debug 340.1/641.0MB
Projects Services Output
Blackjack
  > Header Files
  > Resource Files
  > Source Files
  > Test Files
  > Important Files
main() - Navigator
Blackjack (Build, Run) x Blackjack (Run) x
Sorted Player Winnings (Selection Sort):
Ben: 90 points
Scott: 0 points
Akshay: 0 points
Sean: 0 points
Aaron: 0 points
Roger: 0 points
Bill: 0 points
John: 0 points
Akshay's balance: 0
John's balance: 0
Scott's balance: 0
Ben's balance: 80
Bill's balance: 0
Sean's balance: 0
Roger's balance: 0
Aaron's balance: 0

Enter player name to search: 

```

Search for a  
player



Blackjack - Apache NetBeans IDE 14

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Debug 407.6/641.0MB

Projects Services Output

Blackjack (Build, Run) × Blackjack (Run) ×

```
Sean: 0 points
Aaron: 0 points
Roger: 0 points
Bill: 0 points
John: 0 points
Akshay's balance: 0
John's balance: 0
Scott's balance: 0
Ben's balance: 80
Bill's balance: 0
Sean's balance: 0
Roger's balance: 0
Aaron's balance: 0

Enter player name to search: Ben
Ben found with total: 21

RUN FINISHED; exit value 0; real time: 6m 38s; user: 0ms; system: 0ms
```

## 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, 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

// 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 cardIdx = 2;
    do {
        cout << pNames[i] << "s total: " << pTotals[i] << endl;
        cout << "Do you want to hit (h/H) or stand (s/S)? ";
        cin >> choice;
        vallInput(choice);

        if (choice == 'h' || choice == 'H') {
            int newCrd = getCard();
            string newSut = getSuit();
            if (cardIdx < MAX_CARDS) {
                updHand(hands, pTotals, i, cardIdx, newCrd, hSuits, newSut);
                cardIdx++;
                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 dCrdIdx = 2;
while (dTotal < DEALER_LIMIT && dCrdIdx < MAX_CARDS) {
    int newCrd = getCard();
    string newSut = getSuit();
    dTotal += newCrd;
    cout << "Dealer got " << newCrd << " of " << newSut << endl;
    updHand(hands, pTotals, nPlyrs, dCrdIdx, newCrd, hSuits, newSut);
    dCrdIdx++;
}

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;
}

// Get a random card value
int getCard() {
    // 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;
}

// 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
2	3	Libraries	5, 6, 7, 8, 9	Including necessary libraries
	6	Integers	15, 16, 17, 18, 19	Integer variables used for scores, bets, and counts
	8	Strings	28, 32, 33, 35, 53	Strings used for player names and card suits
	12	Variables 7 characters or less	79, 81, 84, 90, 97	All variables adhere to naming constraints
	14	Arithmetic operators	277, 287, 388, 393, 422	Arithmetic operations for card totals and bets
	15	Comments	1, 4, 5, 10, 15	Comments added throughout the code
	16	Named Constants	15, 16, 17, 18, 19	Constants for max score, dealer limit, etc.
3	2	Math Expression	266, 282, 285, 289, 308	Calculations for card values and totals
	4	Overflow/Underflow	277, 287, 388, 393, 422	Checking for busts
	11	Validating user input	152, 153, 154, 155, 156	Functions for validating input
4	1	Relational Operators	422, 425, 427, 429, 435	Comparing player totals to dealer totals
	2, 4, 6	if, If-else, If-else-if	277, 287, 305, 309, 312	Conditional logic for game flow
	11	Validating user input	152, 153, 154, 155, 156	Input validation logic

5	1	Increment/Decrement	179, 180, 181, 182, 183	Loops for iterating over players and cards
	2	While	277, 278, 279, 280, 281	Loop for dealer's turn
	5, 6	Do-while, For loop	236, 237, 238, 239, 240	Loops for player turns and sorting
6	3	Function Prototypes	40, 41, 42, 43, 44	Prototypes for all functions
	5, 8	Pass by Value, return	254, 258, 263, 271, 278	Functions with value passing and return values
	9	returning boolean	244, 245, 246, 247, 248	Function validation
	12	defaulted arguments	128, 129, 130, 131, 132	Default arguments in updFile
	13	pass by reference	255, 262, 271, 298, 325	Passing vectors and arrays by reference
7	1-6	Single Dimensional Arrays	111, 112, 113, 114, 115	Arrays for player hands and totals
	7	Parallel Arrays	82, 83, 84, 85, 86	Parallel arrays for player data
	8	Single Dimensional as Function Args	126, 135, 137, 172, 177	Arrays passed to functions
	9	2 Dimensional Arrays	111, 112, 113, 114, 115	2D array for card hands
	12	STL Vectors	79, 80, 81, 82, 83	Vectors for dynamic player data
8	3	Bubble Sort, Selection Sort	481, 482, 483, 484, 485	Sorting functions for player totals and winnings

	1	Linear or Binary Search	535, 536, 537, 538, 539	Linear search for player names
--	---	-------------------------	-------------------------	--------------------------------