

BLACKJACK
A CARD GAME

Ireoluwa Dairo

CSC/CIS 17A

47538

FALL 2024

INTRODUCTION

This documentation gives an overview of a Blackjack game written in C++. The code is a classic card game where the goal is to reach a hand value of 21 or as close as possible without exceeding it. The player and dealer start with two cards each, and the player can choose to 'hit' or 'stand' to adjust their total hand value. The dealer must draw until it reaches a minimum of 17. Whoever is closest to 21 without exceeding wins the round. It shows some key C++ concepts, including memory allocation with arrays and structures, file handling, pointers, and dynamic arrays

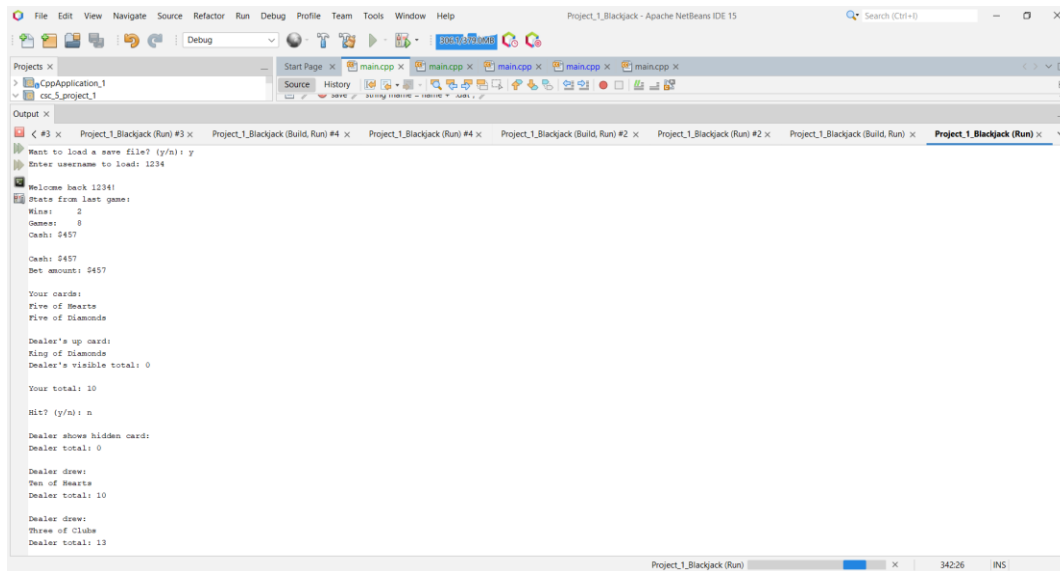
Summary

This 429-line project contains 27 functions and shows key programming concepts. It includes various variable types (integers, characters, strings, Booleans), dynamic memory allocation, arrays, structs, enums, and file I/O operations. Multiple loops and conditional statements are used in the program to implement game logic and validate user input. The most difficult aspect was breaking down the main function into smaller, more manageable functions, which took around three days to complete, including documentation and a flowchart. Overall, it meets the requirements for a first project based off the checklist

Description

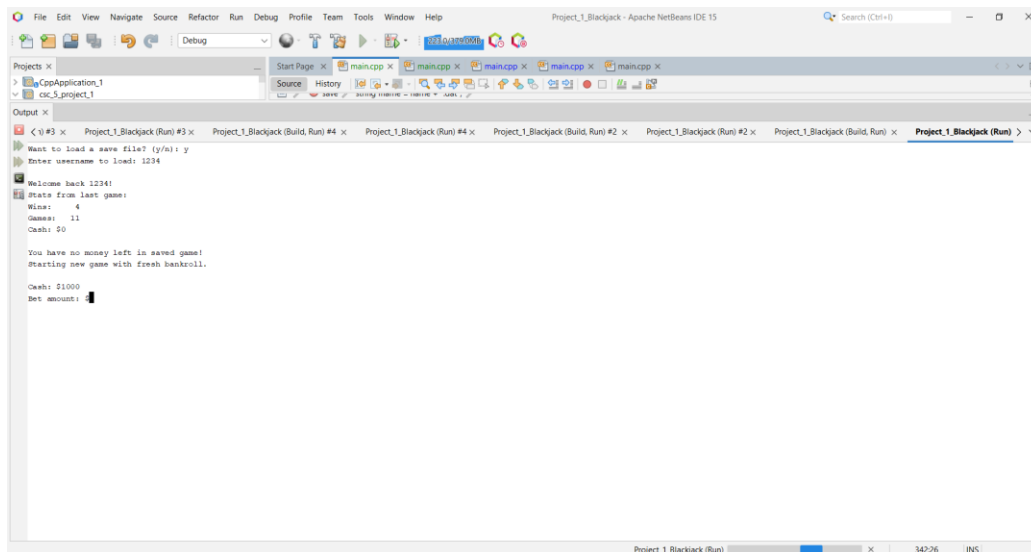
I programmed the game by breaking it into smaller functions, using structures for data organization, implementing dynamic memory for the deck and hands, and handling user input as well as the game logic. Once the program is loaded, it prompts you to decide whether to load a save file. Afterward, you continue with the Blackjack game. You start with a maximum account balance of \$1000, place bets, and play rounds until you either decide to quit or run out of money. A save file named '1234' is included for testing.

Sample Input/Output



```
Project_1_Blackjack (Run) #3 x Project_1_Blackjack (Build, Run) #4 x Project_1_Blackjack (Run) #4 x Project_1_Blackjack (Build, Run) #2 x Project_1_Blackjack (Run) #2 x Project_1_Blackjack (Build, Run) x Project_1_Blackjack (Run) x
> CppApplication.1
> csc_5_project.1
Source History
Want to load a save file? (y/n): y
Enter username to load: 1234
Welcome back 1234!
State from last game:
Wins: 2
Games: 8
Cash: $457
Cash: $457
Bet amount: $457
Your cards:
Five of Hearts
Five of Diamonds
Dealer's up card:
King of Diamonds
Dealer's visible total: 0
Your total: 10
Hit? (y/n): n
Dealer shows hidden card:
Dealer total: 0
Dealer drew:
Ten of Hearts
Dealer total: 10
Dealer drew:
Three of Clubs
Dealer total: 13
Project_1_Blackjack (Run) x 34226 INS
```

```
Final dealer total: 24
Final totals:
Your total: 20
Dealer total: 24
Dealer bust! You win $914
Current cash: $1828
Play again? (y/n): n
Cash: $1828
Bet amount: $1828
Your cards:
Ten of Spades
Three of Spades
Dealer's up card:
Four of Spades
Dealer's visible total: 4
Your total: 33
Final totals:
Your total: 33
Bust! You lose $1828
Current cash: $0
You're broke! Game over!
SUCCESSFUL (total time: 1m 24s)
```



```
Project_1_Blackjack - Apache NetBeans IDE 15
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Start Page x maincpp x maincpp x maincpp x maincpp x maincpp x
Source History
Want to load a save file? (y/n): y
Enter username to load: 1234
Welcome back 1234!
State from last game:
Wins: 4
Games: 11
Cash: $0
You have no money left in saved game!
Starting new game with fresh bankroll.
Cash: $1000
Bet amount:
Project_1_Blackjack (Run) x 34226 INS
```

Flowcharts/Pseudocode/UML

Start program

Initialize random number generator

Create and initialize deck, player's hand, dealer's hand, and other necessary variables

Call function to initialize game or load saved game

Repeat the game loop:

- Initialize the deck and shuffle it
- Get the player's bet
- Deal initial cards to the player and dealer
- Display the initial cards
- Play the player's turn:
 - Show current hand value
 - If the player wants to hit, draw a card and update hand value
 - If the player busts (value > 21), stop their turn
- If the player hasn't busted, play the dealer's turn:
 - Reveal the dealer's hidden card
 - The dealer must draw cards until their total is 17 or greater
- Handle the outcome of the game:
 - Compare the player's and dealer's totals
 - Update the player's cash and wins based on the result
- Save the game state
- Check if the player can continue based on remaining cash
- If the player chooses to play again, reset necessary variables for the next round

End game and clean up memory

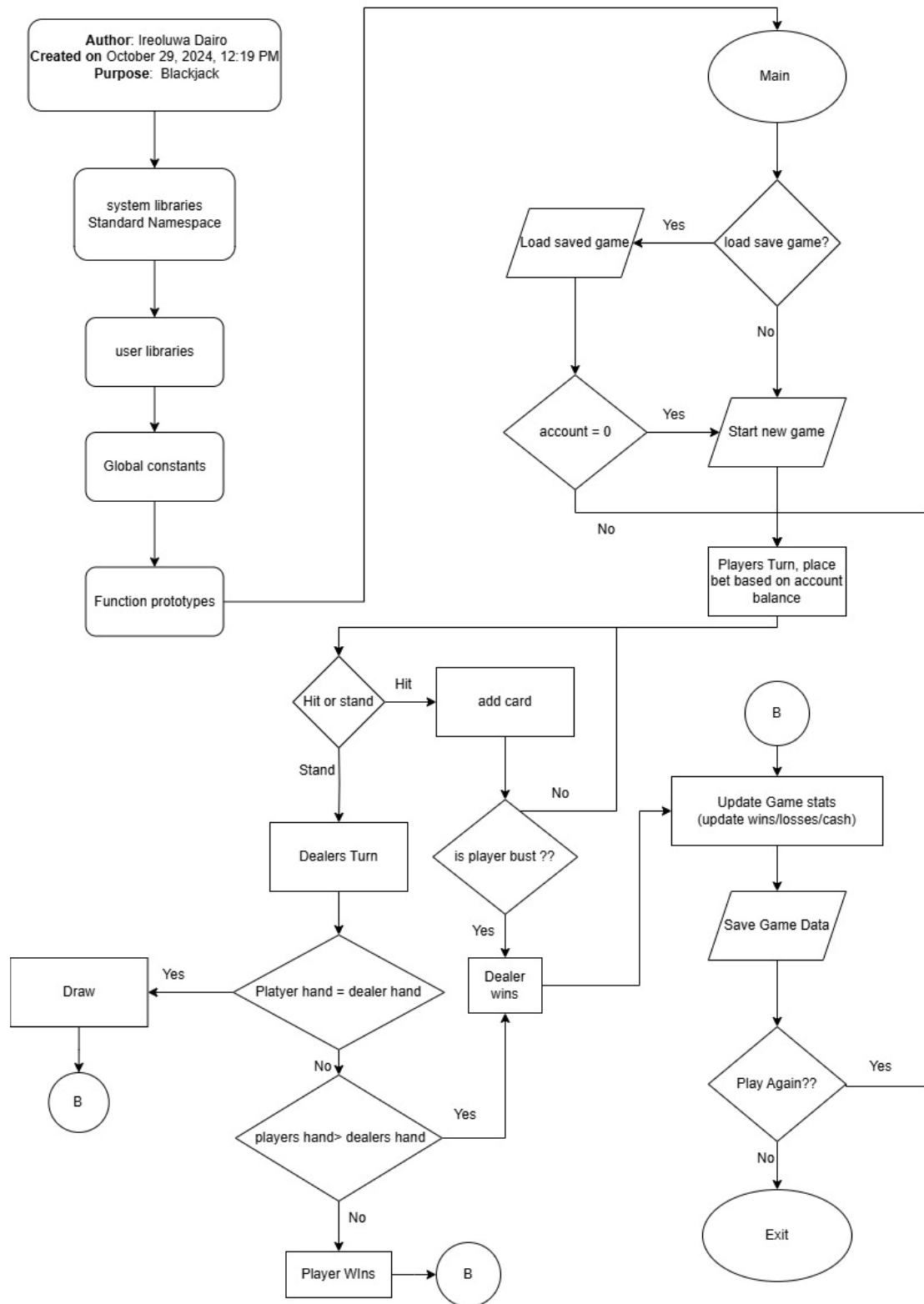
End program

Functions:

init(deck, numCards) - Initialize the deck
shuf(deck, numCards) - Shuffle the deck
draw(deck, numCards) - Draw a card from the deck
pCard(card, suit) - Print card's value and suit
load(userData, filename) - Load a saved game
save(userData, filename) - Save the current game state
sum(hand, numCards) - Calculate the sum of the hand
hit() - Ask player if they want to hit
vName(name) - Validate the username input
clrBuf() - Clear the input buffer
flSuit(suit) - Get the full name of the suit
cardVal(value) - Get the card's value as a string
inzeGme(userData, filename) - Initialize a new game or load a saved game
hdSVFLd(userData, filename) - Check if a saved file exists
crtNwUr(userData, filename) - Create a new user profile
dpUsrSt(userData) - Display user statistics
gtBtAnt(userData) - Get the bet amount from the player
dlInCrd(deck, numCards, hand, numHand, dHand, numDHand) - Deal initial cards
dspInDI(hand, numHand, dHand, deck, numCards) - Display the initial hands
plyPrTn(deck, numCards, hand, numHand, pSum) - Player's turn
plyDrTn(deck, numCards, dHand, numDHand, dSum) - Dealer's turn
hndlGOt(userData, bet, pSum, dSum) - Handle the game outcome
chcGmCe(userData) - Check if the player can continue
plyARnd() - Ask the player if they want to play another round
cNuGame(deck, hand, dHand) - Clean up for a new game

End pseudocode

FLOWCHART



Variables

The program has several major variables. In the "main" function, there are global variables like "deck", a pointer to an array of "Card" structs representing the deck, and "hand" and "dHand", which store the player's and dealer's hands as arrays of integers. There are also integer variables like "pSum" and "dSum" to track the total values of the player's and dealer's hands, and "nCards", "nHand", and "nDHand" to manage the number of cards in the deck and hands. Additionally, a "Save" struct stores the player's profile information, such as their name, wins, games, and cash. Local variables like "bet", "resp", and "cont" are used for player decisions, while the "Card" and "Save" structs define the card details and user profile, respectively.

Concepts

All but 1(Function return) concept from sections in Chapter 9 to Chapters 12 on the checklist

References

I made modifications to one of my CSC-5 project codes.

Program

```
/* Author: Ireoluwa
 * Created on October 29, 2024, 12:19 PM
 * Purpose: blackjack
 */

//System Libraries
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <string>
#include <cctype>
#include <cstring>
using namespace std;

//User Libraries
struct Card {
    char suit;
    int val;
};

struct Save {
    char name[30];
    int wins;
    int games;
    int cash;
};

//Global Constants
enum Suit {HEART, DIAM, SPADE, CLUB};

//Function Prototypes
void init(Card*, int&); //Initialize deck
void shuf(Card*, int); //Shuffle deck
int draw(Card*, int&); //Draw card
void pCard(int, char); //Print card
bool load(Save&, string); //Load game
void save(Save, string); //Save game
int* sum(int*, int); //Sum hand
bool hit(); //Ask hit/stand
bool vName(string); //Validate name
```



```

void clrBuf();          //Clear buffer
string flSuit(int);     // Get full name of suit
string cardVal(int);    // Get card value as string
void inzeGme(Save&, string&); // Initialize new or load existing game
bool hdSVFLd(Save&, string&); // Load user save file
void crtNwUr(Save&, string&); // Create new user profile
void dpUsrSt(Save&);    // Display user stats
int gtBtAnt(const Save&); // Get bet amount from player
void dlInCrd(Card*, int&, int*, int&, int*, int&); // Deal initial cards
void dspInDl(const int*, int, const int*, const Card*, int); // Show initial cards
void plyPrTn(Card*, int&, int*, int&, int&); // Player's turn
void plyDrTn(Card*, int&, int*, int&, int&); // Dealer's turn
void hndlGOt(Save&, int, int, int); // Handle game outcome
bool chcGmCe(const Save&); // Check if player can continue
bool plyARnd(); // Ask if player wants to play again
void cNuGame(Card*, int*, int*); // Clean up for a new game

//Execution Begins here
int main(int argc, char** argv) {
    //Setting the random number seed
    srand(time(0));

    //Declaring Variables
    const int SIZE = 52; // Deck size
    Card* deck = new Card[SIZE]; // Create deck array
    int* hand = new int[10]; // Player's hand array
    int* dHand = new int[10]; // Dealer's hand array
    int nCards = SIZE; // Number of cards in deck
    int pSum = 0, dSum = 0; // Player and dealer totals
    int nHand = 0, nDHand = 0; // Number of cards in each hand
    Save user; // User profile data
    string fname; // Save file name

    inzeGme(user, fname); // Initialize game

    do {
        init(deck, nCards); // Initialize deck
        shuf(deck, nCards); // Shuffle deck
        int bet = gtBtAnt(user); // Get player's bet

        dlInCrd(deck, nCards, hand, nHand, dHand, nDHand); // Deal initial cards
        dspInDl(hand, nHand, dHand, deck, nCards); // Show initial hands

        plyPrTn(deck, nCards, hand, nHand, pSum); // Player's turn
    } while (true);
}

```

```

// Dealer's turn if player didn't bust
if (pSum <= 21) {
    plyDrTn(deck, nCards, dHand, nDHand, dSum);
}

hdlGOt(user, bet, pSum, dSum); // Handle outcome
save(user, fname);           // Save game state

if (!chcGmCe(user)) break; // Check if player can continue

if (plyARnd()) { // Reset if player wants to play again
    nHand = 0;
    nDHand = 0;
    nCards = SIZE;
}

} while (true); // Game loop

cNuGame(deck, hand, dHand); // Clean up memory
return 0;
}

// Function prototype to initialize a new game or load a previous game
void inzeGme(Save& user, string& fname) {
    char resp;
    cout << "Want to load a save file? (y/n): ";
    cin >> resp;
    clrBuf();

    if (resp == 'y') {
        if (hdSVFLd(user, fname)) {
            dpUsrSt(user);
        } else {
            crtNwUr(user, fname);
        }
    } else {
        crtNwUr(user, fname);
    }
}

// Function prototype to handle loading of save file if it exists
bool hdSVFLd(Save& user, string& fname) {
    do {

```

```

        cout << "Enter username to load: ";
        getline(cin, fname);
    } while (!vName(fname));

    if (load(user, fname)) {
        return true;
    }
    cout << "Save file not found.\n";
    return false;
}

// Function prototype to create a new user profile
void crtNwUr(Save& user, string& fname) {
    do {
        cout << "Enter new username (3-30 chars, alphanumeric): ";
        getline(cin, fname);
    } while (!vName(fname));

    strncpy(user.name, fname.c_str(), 29);
    user.name[29] = '\0';
    user.wins = 0;
    user.games = 0;
    user.cash = 1000;
    cout << "\nWelcome " << user.name << "!\n";
    cout << "Starting cash: $" << user.cash << "\n\n";
}

// Function prototype to display user stats
void dpUsrSt(Save& user) {
    cout << "\nWelcome back " << user.name << "!\n";
    cout << "Stats from last game:\n";
    cout << "Wins: " << setw(5) << user.wins << endl;
    cout << "Games: " << setw(4) << user.games << endl;
    cout << "Cash: $" << user.cash << endl << endl;

    if (user.cash <= 0) {
        cout << "You have no money left in saved game!\n";
        cout << "Starting new game with fresh bankroll.\n\n";
        user.cash = 1000; // Reset cash to 1000
    }
}

// Function prototype to get the player's betting amount
int gtBtAnt(const Save& user) {

```

```

int bet;
do {
    cout << "Cash: $" << user.cash << endl;
    cout << "Bet amount: $";
    if (!(cin >> bet)) {
        cout << "Invalid bet. Enter a number.\n";
        clrBuf();
        bet = 0;
        continue;
    }
    if (bet > user.cash) cout << "Can't bet more than you have!\n";
    if (bet < 1) cout << "Minimum bet is $1\n";
} while (bet > user.cash || bet < 1);
clrBuf();
return bet;
}

// Function prototype to deal initial cards to the player and dealer
void dlInCrd(Card* deck, int& nCards, int* hand, int& nHand,
             int* dHand, int& nDHand) {
    cout << "\nYour cards:\n";
    hand[nHand] = draw(deck, nCards);
    pCard(hand[nHand], deck[nCards].suit);
    nHand++;

    hand[nHand] = draw(deck, nCards);
    pCard(hand[nHand], deck[nCards].suit);
    nHand++;
}

// Function prototype to display the initial cards of player and dealer
void dspInDl(const int* hand, int nHand, const int* dHand,
             const Card* deck, int nCards) {
    cout << "\nDealer's up card:\n";
    pCard(dHand[0], deck[nCards].suit);
    cout << "Dealer's visible total: " << dHand[0] << endl;
}

// Function prototype to play the player's turn
void plyPrTn(Card* deck, int& nCards, int* hand, int& nHand, int& pSum) {
    pSum = *sum(hand, nHand);
    cout << "\nYour total: " << pSum << endl;

    while (pSum < 21 && hit()) {

```

```

        hand[nHand] = draw(deck, nCards);
        cout << "\nYou drew:\n";
        pCard(hand[nHand], deck[nCards].suit);
        nHand++;
        pSum = *sum(hand, nHand);
        cout << "\nYour total: " << pSum << endl;
    }
}

// Function prototype to play the dealer's turn
void plyDrTn(Card* deck, int& nCards, int* dHand, int& nDHand, int& dSum) {
    dSum = *sum(dHand, nDHand);
    cout << "\nDealer shows hidden card:\n";
    for (int i = 0; i < nDHand; i++) {
        pCard(dHand[i], deck[nCards + i].suit);
    }
    cout << "Dealer total: " << dSum << endl;

    while (dSum < 17) {
        dHand[nDHand] = draw(deck, nCards);
        cout << "\nDealer drew:\n";
        pCard(dHand[nDHand], deck[nCards].suit);
        nDHand++;
        dSum = *sum(dHand, nDHand);
        cout << "Dealer total: " << dSum << endl;
    }
    cout << "\nFinal dealer total: " << dSum << endl;
}

// Function prototype to handle the game outcome and update the player's stats
void hndLGOT(Save& user, int bet, int pSum, int dSum) {
    cout << "\nFinal totals:\n";
    cout << "Your total: " << pSum << endl;
    if (pSum <= 21) cout << "Dealer total: " << dSum << endl;

    user.games++;
    if (pSum > 21) {
        cout << "\nBust! You lose $" << bet << endl;
        user.cash -= bet;
    }
    else if (dSum > 21) {
        cout << "\nDealer bust! You win $" << bet << endl;
        user.cash += bet;
        user.wins++;
    }
}

```

```

    }
    else if (pSum > dSum) {
        cout << "\nYou win $" << bet << endl;
        user.cash += bet;
        user.wins++;
    }
    else if (pSum < dSum) {
        cout << "\nYou lose $" << bet << endl;
        user.cash -= bet;
    }
    else cout << "\nPush!\n";

    cout << "Current cash: $" << user.cash << endl;
}

// Function prototype to check if the player can continue the game
bool chcGmCe(const Save& user) {
    if (user.cash <= 0) {
        cout << "\nYou're broke! Game over!\n";
        return false;
    }
    return true;
}

// Function prototype to prompt the player if they want to play another round
bool plyARnd() {
    char cont;
    do {
        cout << "\nPlay again? (y/n): ";
        cin >> cont;
        cont = tolower(cont);
        if (cont != 'y' && cont != 'n')
            cout << "Please enter y or n\n";
    } while (cont != 'y' && cont != 'n');
    clrBuf();
    if (cont == 'n') { cout << "Exiting the game. Goodbye!" << endl;
        exit(0); return true;}}

// Function prototype to clean up memory and end the game
void cNuGame(Card* deck, int* hand, int* dHand) {
    delete[] deck;
    delete[] hand;
    delete[] dHand;
}

```

```

// Function prototype to validate the player's username
bool vName(string name) {
    if (name.length() < 3 || name.length() > 30) {
        cout << "Username must be 3-30 characters\n";
        return false;
    }

    for (char c : name) {
        if (!isalnum(c) && c != '_') {
            cout << "Username can only contain letters, numbers, and underscore\n";
            return false;
        }
    }
    return true;
}

// Function prototype to clear the input buffer
void clrBuf() {
    cin.clear();
    cin.ignore(1000, '\n');
}

// Function prototype to load a saved game
bool load(Save& user, string name) {
    string fname = name + ".dat";
    ifstream in(fname, ios::binary | ios::in);
    if (!in) return false; in.seekg(0, ios::end); long fileSize = in.tellg();
    in.seekg(0, ios::beg); in.read(reinterpret_cast<char*>(&user), sizeof(Save));
    in.close();
    return true;
}

// Function prototype to save the current game state
void save(Save user, string name) {
    string fname = name + ".dat";
    ofstream out(fname, ios::binary | ios::out);
    out.write(reinterpret_cast<char*>(&user), sizeof(Save));
    out.close();
}

// Function prototype to initialize the deck of cards
void init(Card* deck, int& size) {
    for (int i = 0; i < size; i++) {

```

```

        deck[i].val = i % 13 + 1;
        deck[i].suit = static_cast<Suit>(i / 13);
    }
}

// Function prototype to shuffle the deck of cards
void shuf(Card* deck, int size) {
    for (int i = 0; i < size; i++) {
        int j = rand() % size;
        Card temp = deck[i];
        deck[i] = deck[j];
        deck[j] = temp;
    }
}

// Function prototype to draw a card from the deck
int draw(Card* deck, int& size) {
    return deck[--size].val > 10 ? 10 : deck[size].val;
}

// Function prototype to return the full name of a suit (e.g., Hearts)
string flSuit(int s) {
    switch (s) {
        case HEART: return "Hearts";
        case DIAM: return "Diamonds";
        case SPADE: return "Spades";
        default: return "Clubs";
    }
}

// Function prototype to return the full name of a card's value (e.g., Ace)
string cardVal(int val) {
    switch (val) {
        case 1: return "Ace";
        case 2: return "Two";
        case 3: return "Three";
        case 4: return "Four";
        case 5: return "Five";
        case 6: return "Six";
        case 7: return "Seven";
        case 8: return "Eight";
        case 9: return "Nine";
        case 10: return "Ten";
        case 11: return "Jack";
    }
}

```



```

        case 12: return "Queen";
        default: return "King";
    }
}

// Function prototype to print a single card's value and suit
void pCard(int val, char suit) {
    cout << cardVal(val) << " of " << flSuit(suit) << endl;
}

// Function prototype to calculate the sum of a hand
int* sum(int* hand, int size) {
    static int total = 0;
    total = 0;
    int aces = 0;

    for (int i = 0; i < size; i++) {
        if (hand[i] == 1) {
            aces++;
            total += 11;
        } else {
            total += hand[i];
        }
    }

    while (total > 21 && aces > 0) {
        total -= 10;
        aces--;
    }

    return &total;
}

// Function prototype to prompt the player for hit or stand
bool hit() {
    char ch;
    do {
        cout << "\nHit? (y/n): ";
        cin >> ch;
        ch = tolower(ch);
        if (ch != 'y' && ch != 'n')
            cout << "Please enter y or n\n";
    } while (ch != 'y' && ch != 'n');
    return ch == 'y';
}

```