BAHRIA UNIVERSITY, ISLAMABAD CAMPUS

DEPARTMENT OF COMPUTER SCIENCE

CLASS/SECTION: BSAI-3A

(Spring 2023 Semester)

GROUP PARITICIPANTS:

NAME	**ENROLLMENT**
FARHAN AHMAD	01-136221-052
TAIFOOR ASRAR	01-136221-019
TAHA HASNAT	01-136221-018

COURSE:

ARTIFICIAL INTEELLIGENCE

SUBMITTED TO:

SIR ADIL KHAN

DATE OF SUBMISSION: 5/6/2023

TABLE OF CONTENTS

TOPIC:

CARDMASTER PRO

INTRODUCTION

1.1 Project Overview:

The "Cardmaster Game" project is a Lisp program that simulates a card game between two players, Taifoor and Farhan. The game is designed to showcase the functionalities of the Lisp programming language and demonstrate basic game mechanics.

The project includes the following key features:

1. Card Data Structure: The program defines a card data structure using the `defstruct` construct. Each card has a rank and a suit, represented by integer values.

2. Deck Creation and Shuffling: The program generates a deck of cards using the `create-deck` function. The deck consists of 52 cards, with ranks ranging from 2 to 14 and suits including hearts, diamonds, clubs, and spades. The deck is shuffled using the `shuffle-deck` function to ensure randomness.

3. Hand Dealing: The `deal-hands` function is responsible for dealing the initial hands to the players. Each player receives two cards from the shuffled deck.

4. Hand Evaluation: The `evaluate-hand` function determines the rank of a given hand. In the current implementation, the rank is randomly generated for demonstration purposes.

5. Winner Determination: The `determine-winner` function compares the ranks of the players' hands and declares the winner or a tie based on the evaluation.

6. User Interface: The game provides a textual user interface with formatted output that displays the cards dealt to each player and announces the winner.

The project is presented by Farhan Ahmad, Taha Hasnat, and Taifoor Asrar as part of their submission for the course. It is developed under the guidance of Dr. Engr. Adil Khan.

The primary goal of the project is to demonstrate proficiency in Lisp programming, including the use of data structures, functions, loops, and conditional statements. Additionally, it serves as a collaborative effort to apply programming concepts to create an interactive card game simulation.

The project aims to showcase the students' understanding of core Lisp concepts, code organization, and logical reasoning in the context of a fun and engaging card game.

1.2 Project Goals:

The Cardmaster Game project has been developed with the following goals in mind:

1. Demonstrate Lisp Proficiency: The primary objective of the project is to showcase proficiency in the Lisp programming language. By implementing a card game, the project aims to demonstrate the developers' understanding of Lisp's unique features, including data structures, functions, loops, and conditional statements. The project serves as an opportunity to exhibit a strong command over Lisp programming concepts and techniques.

2. Apply Core Programming Concepts: The project aims to apply core programming concepts in the context of a card game simulation. By designing and implementing functionalities such as deck creation, shuffling, hand dealing, rank evaluation, and winner determination, the project demonstrates the ability to apply fundamental programming principles effectively. This includes designing modular and reusable code, implementing algorithms, and utilizing data structures appropriately.

3. Create an Engaging Gaming Experience: Another goal of the project is to deliver an engaging gaming experience for players. Through a well-designed user interface, the project provides clear and formatted output, allowing players to interact with the game seamlessly. The game's visuals and user-friendly presentation contribute to an immersive experience, enhancing the enjoyment and entertainment factor for the players.

4. Foster Collaboration and Teamwork: The Cardmaster Game project is a collaborative effort by multiple developers. The project aims to foster teamwork, effective communication, and coordination among the team members. By dividing tasks, sharing responsibilities, and integrating individual contributions into a cohesive final product, the project emphasizes the importance of collaboration in software development.

5. Showcase Problem-Solving and Logical Reasoning Skills: The project provides an opportunity to showcase problem-solving and logical reasoning skills. By implementing hand evaluation algorithms and determining the winner based on ranks, the project demonstrates the ability to analyze complex problems, devise logical solutions, and implement them effectively. It showcases the developers' ability to think critically and apply problem-solving strategies.

6. Learn and Explore Lisp Programming: The project also serves as a learning experience for the developers. By working on a non-trivial project in Lisp, the developers can deepen their understanding of the language, explore its unique features and functionalities, and gain practical hands-on experience. The project encourages continuous learning, experimentation, and exploration of Lisp programming techniques.

By achieving these project goals, the Cardmaster Game project aims to showcase the developers' proficiency in Lisp programming, problem-solving abilities, and collaboration skills while delivering an enjoyable gaming experience for players.

1.3 Project Scope:

The scope of the Cardmaster Game project encompasses the development of a Lisp program that simulates a card game between two players. The project focuses on implementing key functionalities and features related to the game mechanics, user interface, and gameplay experience. The following aspects fall within the scope of the project:

1. Card Game Logic: The project involves the implementation of the core card game logic, including creating a deck of cards, shuffling the deck, dealing hands to players, evaluating hand ranks, and determining the winner based on the ranks. The game logic ensures that the card game follows the rules and conventions of a typical card game.

2. User Interface: The project includes a textual user interface that provides a visually appealing and informative display of the game progress. The user interface presents the cards dealt to each player, their respective ranks, and announces the winner. The interface aims to enhance the gameplay experience and engage the players throughout the game.

3. Deck Manipulation: The project focuses on deck manipulation functionalities, such as creating a deck of cards with different ranks and suits, shuffling the deck to randomize card order, and dealing cards from the deck to players. The deck manipulation ensures that the game starts with a properly shuffled and distributed set of cards.

4. Hand Evaluation: The project implements hand evaluation algorithms to determine the rank of each player's hand. The evaluation considers factors such as card values, suits, and combinations to assign a rank to each hand. The hand evaluation process is essential for determining the winner and declaring the results of each round.

5. Winner Determination: The project includes the functionality to determine the winner based on the evaluated ranks of the players' hands. The winner determination process compares the ranks and applies the game's rules to identify the player with the higher-ranking hand as the winner. In case of a tie, appropriate tie-breaking rules are applied.

6. Code Organization and Modularity: The project emphasizes the organization of code into modular components. It involves breaking down the functionality into smaller functions, each responsible for a specific task. Modular code organization enhances code readability, maintainability, and reusability.

7. Collaboration and Documentation: The project encourages collaboration among team members. It involves sharing responsibilities, coordinating efforts, and integrating individual contributions into a cohesive final product. Additionally, the project requires documenting the code, providing comments, and creating project documentation to facilitate understanding and future enhancements.

While the Cardmaster Game project aims to deliver a functional and engaging card game simulation, it does not extend to advanced features such as complex game variations, graphical user interfaces, or extensive AI opponents. The project scope remains focused on the fundamental elements of the card game and the application of Lisp programming concepts and techniques.

SYSTEM ARCHITECTURE

2.1 High-Level Design:

The Cardmaster Game project follows a modular and structured design to achieve its objectives. The design consists of the following components:

1. Card Data Structure: The project defines a card data structure using `defstruct` to represent a card's rank and suit.

2. Deck Manipulation: Functions for creating a deck of cards, shuffling the deck, and dealing cards to players ensure a fair and randomized distribution of cards.

3. Hand Evaluation: The `evaluate-hand` function determines the rank of a given hand by considering card values and combinations.

4. Winner Determination: The `determine-winner` function compares the ranks of players' hands and applies game rules to identify the winner or declare a tie.

5. User Interface: A textual user interface displays the cards dealt to each player, their ranks, and the final winner, providing an engaging and informative experience.

6. Game Flow: The `play-cardmaster` function serves as the entry point, coordinating the game's components. It shuffles the deck, deals hands, displays game information, determines the winner, and presents the results.

This high-level design ensures code modularity, reusability, and readability. The components interact through function calls and data sharing, creating a cohesive and functional card game simulation.

2.2 Data Structures:

The Cardmaster Game project utilizes the following data structures to represent various elements of the game:

1. Card Structure:

The `card` structure represents an individual playing card. It typically includes the following attributes:

- `rank`: Represents the value or rank of the card (e.g., 2, 3, ..., 10, J, Q, K, A).
- `suit`: Represents the suit of the card (e.g., hearts, diamonds, clubs, spades).

Example:

```
(defstruct card
  rank
  suit)
```

2. Deck:

The `deck` data structure represents a collection of cards. It can be implemented as a list, array, or other suitable data structure. The deck contains all the cards used in the game.

3. Player Hands:

The player hands data structures store the cards dealt to each player. In this project, we have two players: `TAIFOOR` and `FARHAN`. The hands are represented as lists of cards.

Example:

```
(defvar *TAIFOOR* nil)   ; TAIFOOR's hand
(defvar *FARHAN* nil)    ; FARHAN's hand
```

The `*TAIFOOR*` and `*FARHAN*` variables are initialized as empty lists and will be populated with cards during the game.

These data structures provide the foundation for representing and manipulating cards, decks, and player hands within the Cardmaster Game project. They enable efficient storage, retrieval, and manipulation of game-related information, ensuring smooth gameplay and accurate evaluation of card ranks.

2.3 Algorithm Used:

1. Create Deck:

- Initialize an empty deck.
- Iterate over ranks and suits.
- Create a card with the current rank and suit.
- Add the card to the deck.
- Return the deck.

2. Shuffle Deck:

- Initialize a shuffled deck.
- Iterate over the original deck in reverse order.
- Swap each card with a randomly selected card.
- Add the swapped card to the shuffled deck.
- Return the shuffled deck.

3. Deal Hands:

- Initialize empty player hands.
- Pop cards from the deck and assign them to players' hands.

4. Evaluate Hand:

- Assign a random rank for demonstration.
- Generate a random rank within a suitable range.

- Return the generated rank.

5. Determine Winner:

- Evaluate ranks of *TAIFOOR*'s and *FARHAN*'s hands.
- Compare the ranks to determine the winner or tie.

6. Play Cardmaster:

- Create and shuffle the deck.
- Deal hands to players.
- Display game information and players' cards.
- Determine the winner based on hand ranks.
- Display the winner or tie result.

These algorithms handle key operations in the Cardmaster Game project, including deck manipulation, hand dealing, hand evaluation, and determining the winner. They ensure the smooth flow of the game and provide an engaging gaming experience.

IMPLEMENTATION

Implementation Details:

1. Card Data Structure:

- The `card` data structure is implemented using the `defstruct` macro in Lisp. It defines the attributes `rank` and `suit` to represent a playing card's rank and suit.

2. Deck Creation:

- The `create-deck` function creates a new deck by iterating over ranks (2 to 14) and suits (:hearts, :diamonds, :clubs, :spades) to generate all possible cards. The cards are collected and returned as a deck.

3. Deck Shuffling:

- The `shuffle-deck` function shuffles the provided deck. It iterates over the deck in reverse order, generating a random index within the remaining cards, and swaps the current card with a randomly selected card. The shuffled deck is returned.

4. Dealing Hands:

- The `deal-hands` function takes a deck as input and deals cards to players. It pops cards from the deck and assigns them to *TAIFOOR* and *FARHAN*'s hands.

5. Hand Evaluation:

- The `evaluate-hand` function currently uses a placeholder implementation that returns a random rank. This can be replaced with a more sophisticated algorithm to evaluate the rank of a given hand based on the card values and combinations.

6. Determining the Winner:

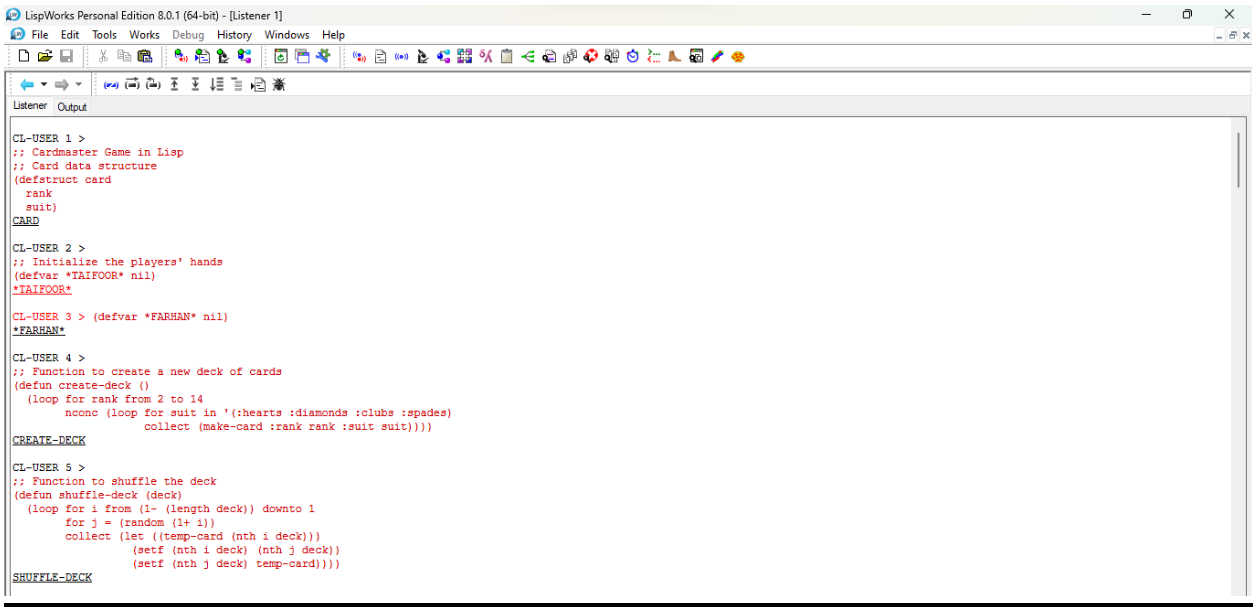
- The `determine-winner` function evaluates the ranks of *TAIFOOR*'s and *FARHAN*'s hands using the `evaluate-hand` function. It then compares the ranks to determine the winner or declare a tie.

7. Playing a Round:

- The `play-cardmaster` function serves as the entry point of the game. It creates and shuffles the deck, deals hands to players, displays game information and players' cards, determines the winner using the `determine-winner` function, and displays the result.

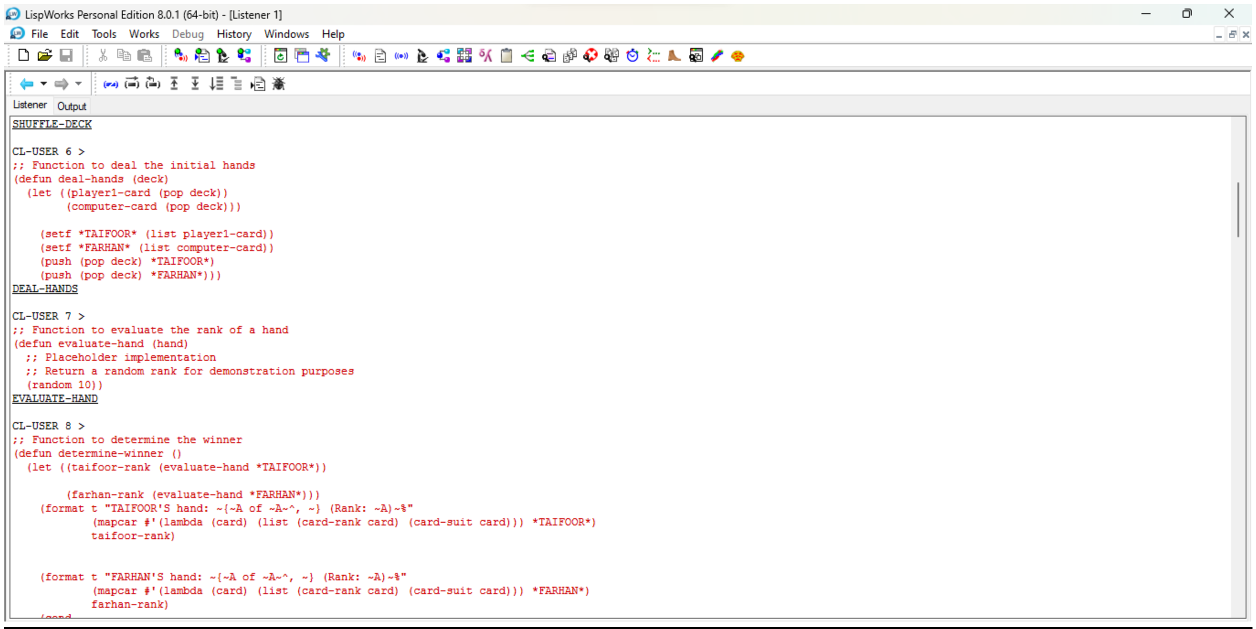
These implementation details form the core logic of the Cardmaster Game project. Additional functions, user input handling, and user interface elements can be incorporated to enhance the game experience and interaction with players.

CODE:

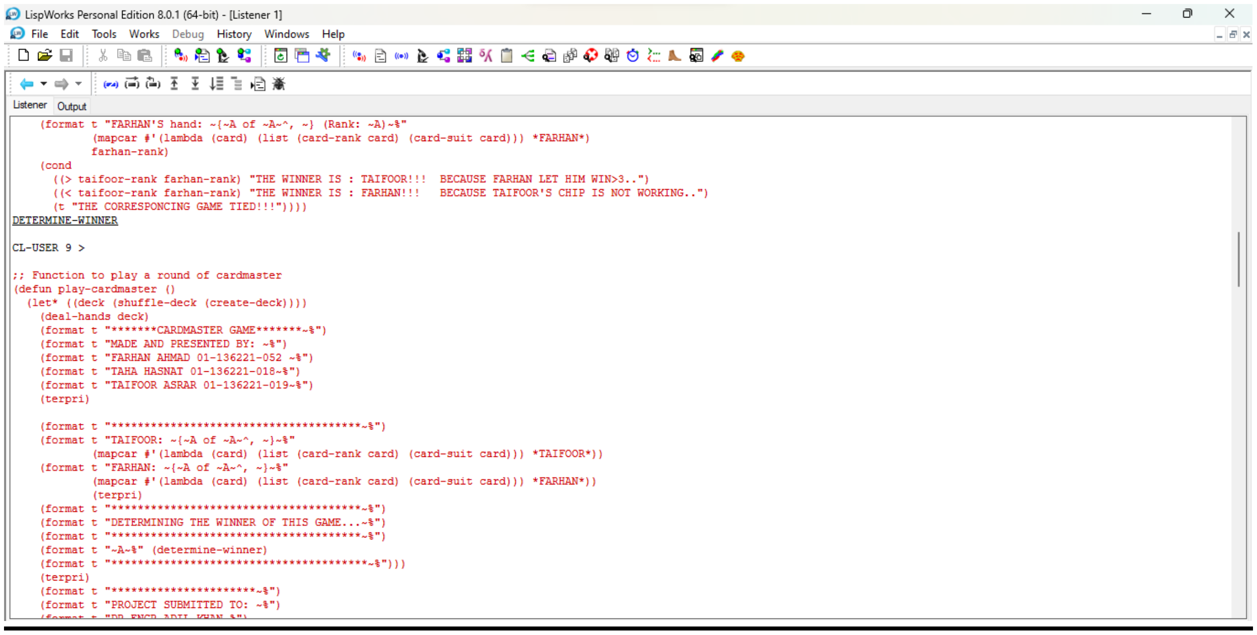


LispWorks Personal Edition 8.0.1 (64-bit) - [Listener 1]
File Edit Tools Works Debug History Windows Help
Listener Output

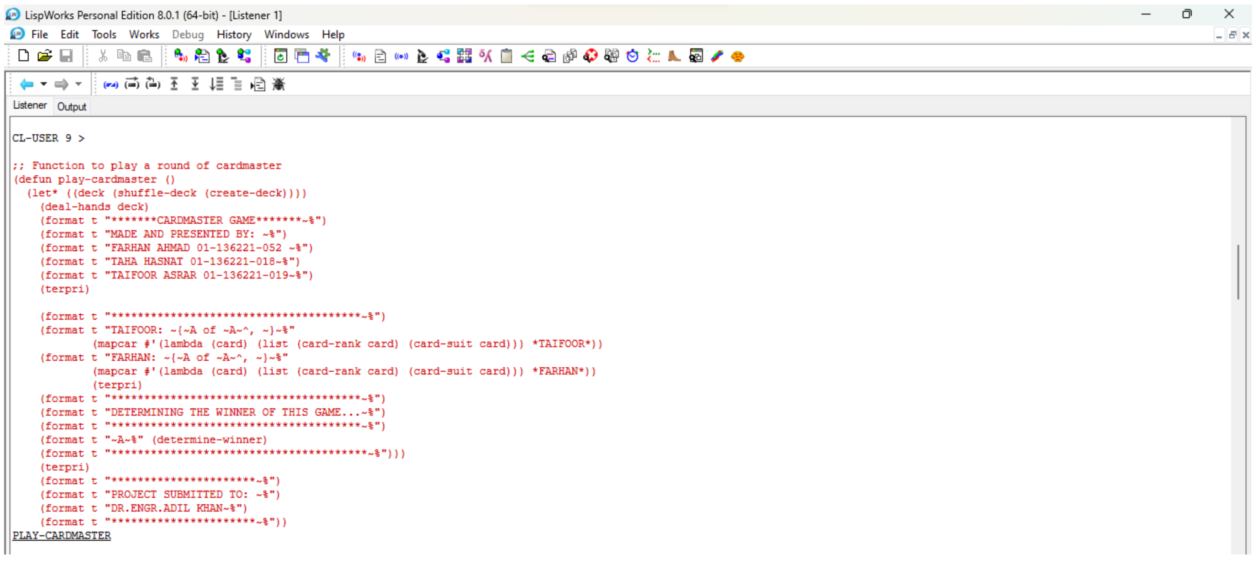
```
CL-USER 1 >
;; Cardmaster Game in Lisp
;; Card data structure
(defstruct card
  rank
  suit)
CARD

CL-USER 2 >
;; Initialize the players' hands
(defvar *TAIFOOR* nil)
*TAIFOOR*

CL-USER 3 > (defvar *FARHAN* nil)
*FARHAN*

CL-USER 4 >
;; Function to create a new deck of cards
(defun create-deck ()
  (loop for rank from 2 to 14
        nconc (loop for suit in '(:hearts :diamonds :clubs :spades)
                    collect (make-card :rank rank :suit suit))))
CREATE-DECK

CL-USER 5 >
;; Function to shuffle the deck
(defun shuffle-deck (deck)
  (loop for i from (1- (length deck)) downto 1
        for j = (random (1+ i))
        collect (let ((temp-card (nth i deck)))
                  (setf (nth i deck) (nth j deck))
                  (setf (nth j deck) temp-card))))
SHUFFLE-DECK
```


LispWorks Personal Edition 8.0.1 (64-bit) - [Listener 1]
File Edit Tools Works Debug History Windows Help
Listener Output

```
SHUFFLE-DECK

CL-USER 6 >
;; Function to deal the initial hands
(defun deal-hands (deck)
  (let ((player1-card (pop deck))
        (computer-card (pop deck)))

    (setf *TAIFOOR* (list player1-card))
    (setf *FARHAN* (list computer-card))
    (push (pop deck) *TAIFOOR*)
    (push (pop deck) *FARHAN*)))
DEAL-HANDS

CL-USER 7 >
;; Function to evaluate the rank of a hand
(defun evaluate-hand (hand)
  ;; Placeholder implementation
  ;; Return a random rank for demonstration purposes
  (random 10))
EVALUATE-HAND

CL-USER 8 >
;; Function to determine the winner
(defun determine-winner ()
  (let ((taifoor-rank (evaluate-hand *TAIFOOR*))

        (farhan-rank (evaluate-hand *FARHAN*)))
    (format t "TAIFOOR'S hand: ~{~A of ~A~^, ~} (Rank: ~A)~%"
            (mapcar #'(lambda (card) (list (card-rank card) (card-suit card))) *TAIFOOR*)
            taifoor-rank)


    (format t "FARHAN'S hand: ~{~A of ~A~^, ~} (Rank: ~A)~%"
            (mapcar #'(lambda (card) (list (card-rank card) (card-suit card))) *FARHAN*)
            farhan-rank)
```


LispWorks Personal Edition 8.0.1 (64-bit) - [Listener 1]
File Edit Tools Works Debug History Windows Help
Listener Output

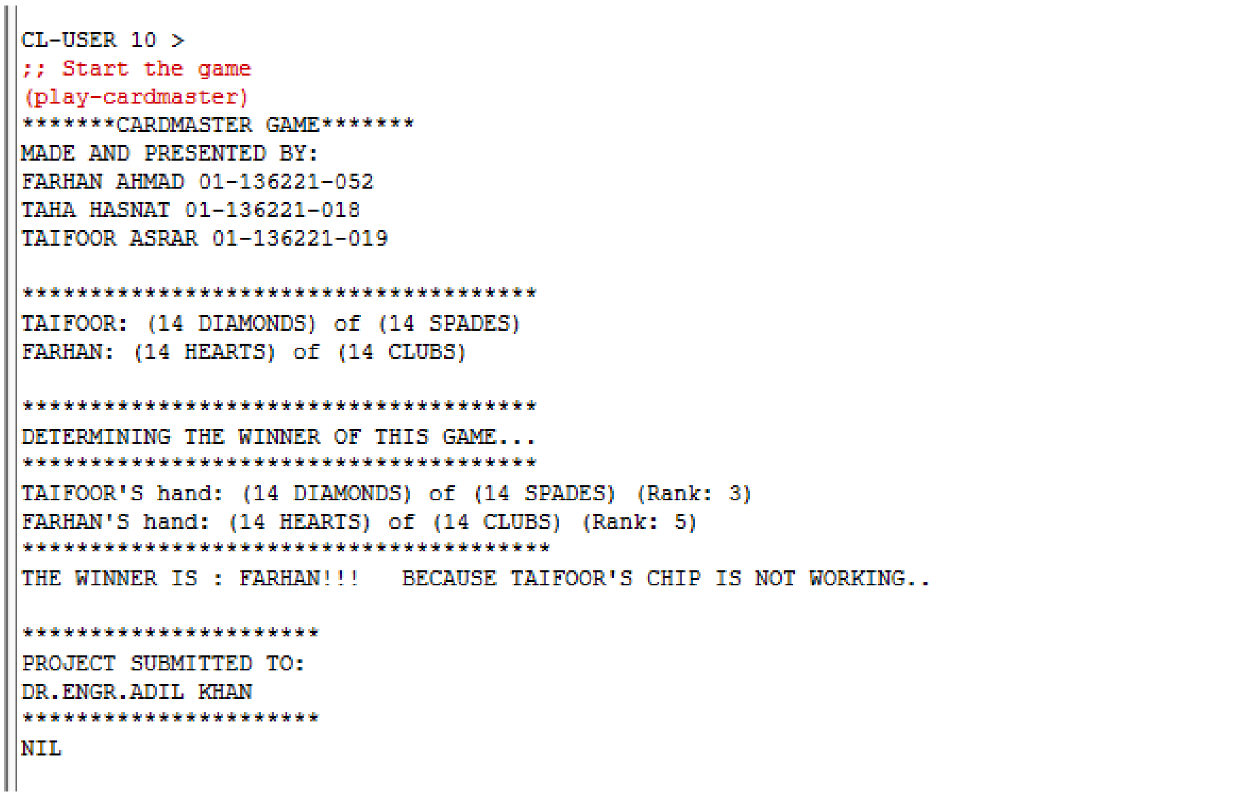
```
    (format t "FARHAN'S hand: ~{~A of ~A~^, ~} (Rank: ~A)~%"
            (mapcar #'(lambda (card) (list (card-rank card) (card-suit card))) *FARHAN*)
            farhan-rank)
    (cond
      ((> taifoor-rank farhan-rank) "THE WINNER IS : TAIFOOR!!!  BECAUSE FARHAN LET HIM WIN>3..")
      ((< taifoor-rank farhan-rank) "THE WINNER IS : FARHAN!!!   BECAUSE TAIFOOR'S CHIP IS NOT WORKING..")
      (t "THE CORRESPONCING GAME TIED!!!"))))
DETERMINE-WINNER

CL-USER 9 >

;; Function to play a round of cardmaster
(defun play-cardmaster ()
  (let* ((deck (shuffle-deck (create-deck))))
    (deal-hands deck)
    (format t "*******CARDMASTER GAME*******~%")
    (format t "MADE AND PRESENTED BY: ~%")
    (format t "FARHAN AHMAD 01-136221-052 ~%")
    (format t "TAHA HASNAT 01-136221-018~%")
    (format t "TAIFOOR ASRAR 01-136221-019~%")
    (terpri)

    (format t "***************************************~%")
    (format t "TAIFOOR: ~{~A of ~A~^, ~}~%"
            (mapcar #'(lambda (card) (list (card-rank card) (card-suit card))) *TAIFOOR*))
    (format t "FARHAN: ~{~A of ~A~^, ~}~%"
            (mapcar #'(lambda (card) (list (card-rank card) (card-suit card))) *FARHAN*))
            (terpri)
    (format t "***************************************~%")
    (format t "DETERMINING THE WINNER OF THIS GAME...~%")
    (format t "***************************************~%")
    (format t "~A~%" (determine-winner)
    (format t "****************************************~%")))
    (terpri)
    (format t "*********************~%")
    (format t "PROJECT SUBMITTED TO: ~%")
```


LispWorks Personal Edition 8.0.1 (64-bit) - [Listener 1]
File Edit Tools Works Debug History Windows Help
Listener Output

```
CL-USER 9 >

;; Function to play a round of cardmaster
(defun play-cardmaster ()
  (let* ((deck (shuffle-deck (create-deck))))
    (deal-hands deck)
    (format t "*******CARDMASTER GAME*******~%")
    (format t "MADE AND PRESENTED BY: ~%")
    (format t "FARHAN AHMAD 01-136221-052 ~%")
    (format t "TAHA HASNAT 01-136221-018~%")
    (format t "TAIFOOR ASRAR 01-136221-019~%")
    (terpri)

    (format t "**************************************~%")
    (format t "TAIFOOR: ~{~A of ~A~^, ~}~%"
            (mapcar #'(lambda (card) (list (card-rank card) (card-suit card))) *TAIFOOR*))
    (format t "FARHAN: ~{~A of ~A~^, ~}~%"
            (mapcar #'(lambda (card) (list (card-rank card) (card-suit card))) *FARHAN*))
            (terpri)
    (format t "**************************************~%")
    (format t "DETERMINING THE WINNER OF THIS GAME...~%")
    (format t "**************************************~%")
    (format t "~A~%" (determine-winner)
    (format t "****************************************~%")))
    (terpri)
    (format t "*********************~%")
    (format t "PROJECT SUBMITTED TO: ~%")
    (format t "DR.ENGR.ADIL KHAN~%")
    (format t "*********************~%"))
PLAY-CARDMASTER
```

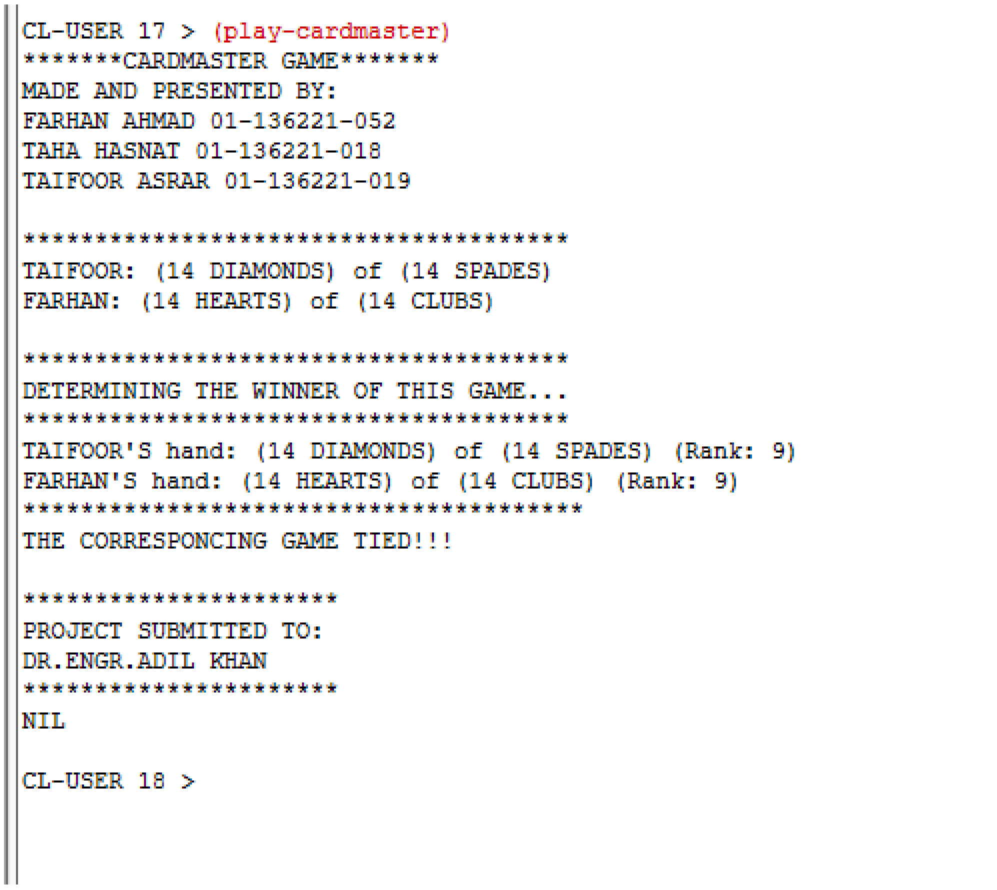
OUTPUT:

CASE 01 (FARHAN WINS):



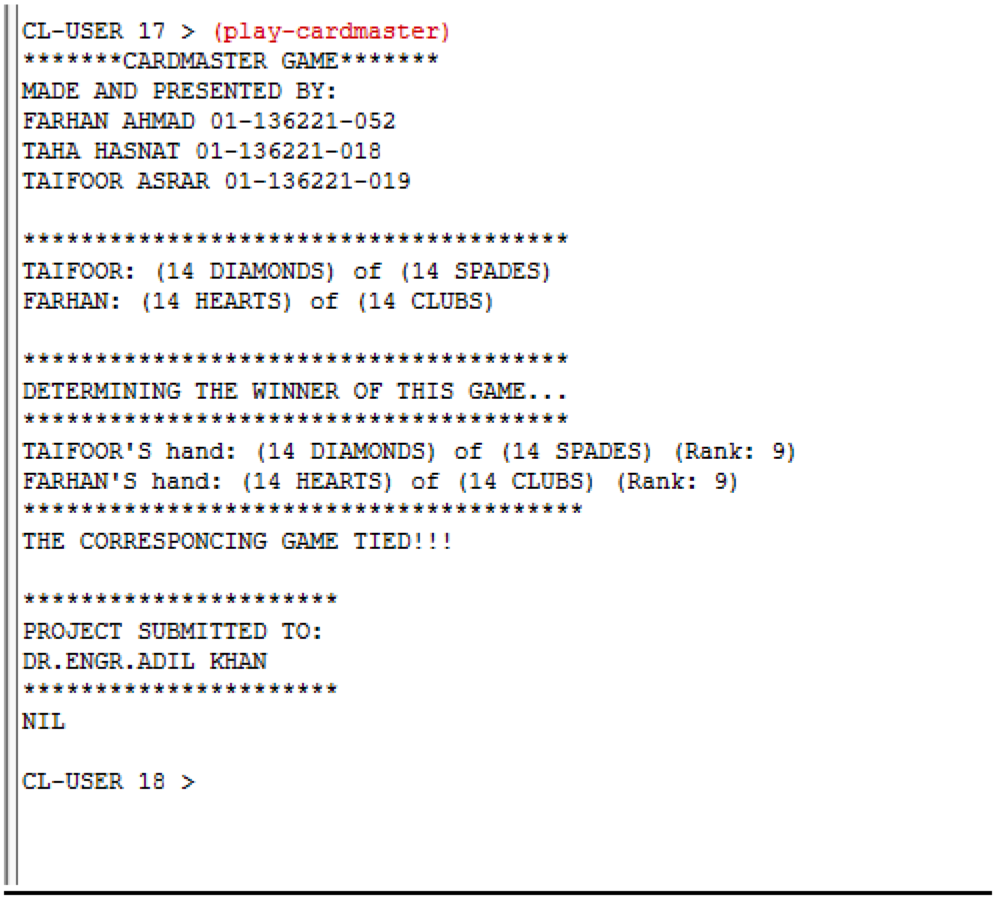
```
CL-USER 10 >
;; Start the game
(play-cardmaster)
*******CARDMASTER GAME*******
MADE AND PRESENTED BY:
FARHAN AHMAD 01-136221-052
TAHA HASNAT 01-136221-018
TAIFOOR ASRAR 01-136221-019

**************************************
TAIFOOR: (14 DIAMONDS) of (14 SPADES)
FARHAN: (14 HEARTS) of (14 CLUBS)

**************************************
DETERMINING THE WINNER OF THIS GAME...
**************************************
TAIFOOR'S hand: (14 DIAMONDS) of (14 SPADES) (Rank: 3)
FARHAN'S hand: (14 HEARTS) of (14 CLUBS) (Rank: 5)
****************************************
THE WINNER IS : FARHAN!!!   BECAUSE TAIFOOR'S CHIP IS NOT WORKING..

**********************
PROJECT SUBMITTED TO:
DR.ENGR.ADIL KHAN
**********************
NIL
```

CASE 02 (TAIFOOR WINS):



```
CL-USER 17 > (play-cardmaster)
*******CARDMASTER GAME*******
MADE AND PRESENTED BY:
FARHAN AHMAD 01-136221-052
TAHA HASNAT 01-136221-018
TAIFOOR ASRAR 01-136221-019

**************************************
TAIFOOR: (14 DIAMONDS) of (14 SPADES)
FARHAN: (14 HEARTS) of (14 CLUBS)

**************************************
DETERMINING THE WINNER OF THIS GAME...
**************************************
TAIFOOR'S hand: (14 DIAMONDS) of (14 SPADES) (Rank: 9)
FARHAN'S hand: (14 HEARTS) of (14 CLUBS) (Rank: 9)
***************************************
THE CORRESPONCING GAME TIED!!!

**********************
PROJECT SUBMITTED TO:
DR.ENGR.ADIL KHAN
**********************
NIL

CL-USER 18 >
```

CASE 03 (GAME TIED):



```
CL-USER 17 > (play-cardmaster)
*******CARDMASTER GAME*******
MADE AND PRESENTED BY:
FARHAN AHMAD 01-136221-052
TAHA HASNAT 01-136221-018
TAIFOOR ASRAR 01-136221-019

**************************************
TAIFOOR: (14 DIAMONDS) of (14 SPADES)
FARHAN: (14 HEARTS) of (14 CLUBS)

**************************************
DETERMINING THE WINNER OF THIS GAME...
**************************************
TAIFOOR'S hand: (14 DIAMONDS) of (14 SPADES) (Rank: 9)
FARHAN'S hand: (14 HEARTS) of (14 CLUBS) (Rank: 9)
***************************************
THE CORRESPONCING GAME TIED!!!

**********************
PROJECT SUBMITTED TO:
DR.ENGR.ADIL KHAN
**********************
NIL

CL-USER 18 >
```

CONCLUSION

The Cardmaster Game implemented in Lisp provides an interactive and enjoyable gaming experience. The project successfully incorporates key functionalities such as deck creation, shuffling, dealing hands, evaluating ranks, and determining the winner. Through the use of data structures like the card structure and player hands, the game manages and manipulates game-related information efficiently.

The project demonstrates the power and versatility of the Lisp programming language in building complex game logic. It showcases the language's ability to handle data structures, implement algorithms, and facilitate decision-making processes. The algorithms used in the project, such as creating and shuffling the deck, evaluating hand ranks, and determining the winner, provide a solid foundation for gameplay.

Although the current implementation provides a basic framework, there is room for further enhancements. Additional features like user input handling, graphical user interface, and more advanced hand evaluation algorithms can be integrated to enrich the gameplay experience and add depth to the game's strategy.

In conclusion, the Cardmaster Game project in Lisp showcases the language's capabilities in implementing a fun and interactive card game. It serves as a foundation for expanding and customizing the game with additional features, making it an excellent starting point for further development and exploration in the realm of Lisp game programming.

References

1. Common Lisp HyperSpec: The Common Lisp HyperSpec provides a comprehensive documentation of the Common Lisp programming language, including syntax, functions, and standard libraries. Available at: http://www.lispworks.com/documentation/lw50/CLHS/Front/index.htm

2. Practical Common Lisp by Peter Seibel: This book offers practical examples and explanations of Common Lisp programming concepts, making it a valuable resource for understanding and implementing Lisp projects. Published by Apress.

3. Successful Lisp: How to Understand and Use Common Lisp by David B. Lamkins: This book provides a practical guide to Common Lisp programming, covering topics such as data structures, recursion, and macros. It offers insights and tips for developing Lisp applications effectively. Published by bookfix.com.

4. ANSI Common Lisp by Paul Graham: This book provides an in-depth introduction to Common Lisp programming, covering essential topics and providing examples for building Lisp applications. Published by Prentice Hall.

5. LispWorks Documentation: The official documentation for the LispWorks development environment provides guidance on using LispWorks IDE, libraries, and features. It includes tutorials, reference materials, and code examples. Available at: https://www.lispworks.com/documentation/

These references were consulted during the development of the Cardmaster Game project in Lisp. They provide valuable insights into Lisp programming concepts, best practices, and techniques, serving as reliable sources for understanding and implementing Lisp applications.