Project 2

Title
Go Fish

course
CIS-5

Author
Paola Alcala

Game introduction

12 cards

The cards are divided in suits and ranks. The goal is to collect as many complete sets of 2 suit matching cards. Collect cards needed from players or go fishing from the remaining cards.

How to play Go Fish

1. Deal 3 cards to each player
2. Player 1 chooses a player and asks for a card.
3. Either other players gives up the card or says "Go fish".
4. If player told to "Go fish!" they grab a card from remaining deck and add it to their hand.

5. If player gets set of 2 matching cards remove from hand and add to matching cards pile.
6. The player's turn will end when they went "fishing" for card.
7. If player secures a card from another player they get another turn.

8. Continue until one player runs out of cards.
9. The game is won by the player who has collected the most matching set of 4 cards.

Development Summary Lines of code: 316
Comment Lines: 120
Total lines of code: 465

Version 1:

I started by creating a simple program to greet players. I used dependent if statements and defaulted arguments to create a function that could greet players individually or collectively, depending on the number of players. This gave me a flexible way to handle different scenarios and set the stage for player interaction.

Version 2:

I then decided to initialize a deck of 52 cards. I created a function to initialize the deck of cards, using arrays to represent suits and ranks. Then, I implemented a function to shuffle the deck, ensuring that the distribution of cards would be random and fair. This added an element of chance to the game and made each playthrough unique.

Version 3:

Distributing the cards between players seemed like the logical next step. I developed a function to deal cards, ensuring that each player received an equal share of cards from the deck. Any remaining cards were placed in the center pile, creating a reservoir from which players could draw cards during the game.

Version 4:

I wanted to add some complexity to the game, so I introduced the option to play against the computer. To make the gameplay smoother, I implemented functions to sort players' hands in ascending order and track matches using linear search. This made it easier for players to manage their hands and strategize their moves, while also providing a challenge when playing against the computer.

Version 5:

To streamline the game further, I made some adjustments to the deck size and introduced a turn-based system using a switch case. This ensured that players took turns asking each other for cards, enhancing the strategic aspect of the game. Additionally, I added functionality for players to ask each other for cards and implemented scoring mechanisms to provide feedback on players' performance.

Version 6:

As I continued to refine the game, I focused on managing and displaying scores more effectively. I introduced parallel arrays to store player names and their corresponding scores, making it easier to track and display this information. I also added functionality to greet players using overloaded functions, accommodating different scenarios such as single or multiplayer games. Finally, I implemented an end() function to handle various end-game conditions and ensure proper termination of the game.

Version 7:

I wanted to experiment with different functionalities and algorithms, so I decided to add a sorting feature using selection sort and display a themed message using a 2D array. This added some visual appeal to the game and showcased my understanding of different programming concepts.

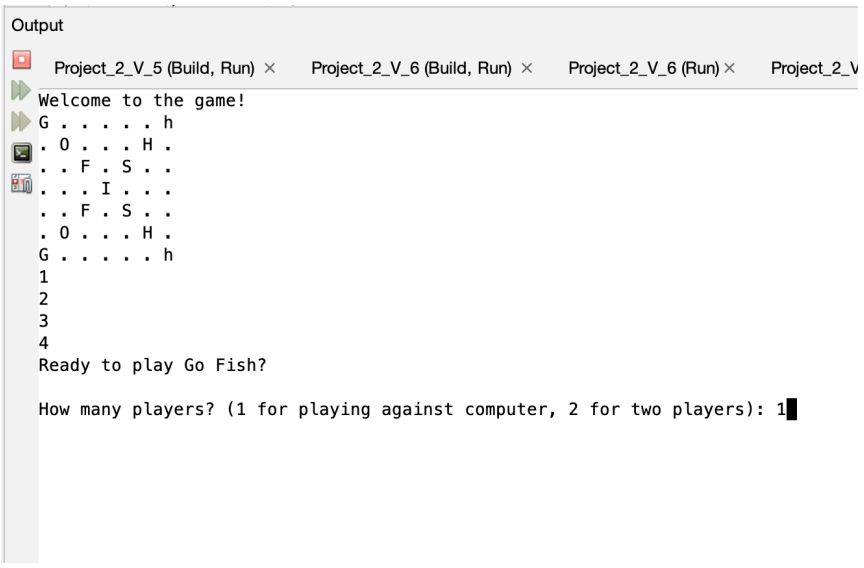
Version 8:

Moving forward, I plan to further improve the game by integrating error handling mechanisms to handle unexpected user inputs or edge cases. I also want to explore more advanced algorithms or game mechanics, such as artificial intelligence for the computer player or additional game modes, to enhance the depth and replayability of the game.

Bugs that need to be fixed

I made the deck of cards smaller because the find match function will recognize teh third matching suit in a players hand as a new match. This is why I made the deck smaller and the players hand to only hold 3 cards at a time so the probability of that happening is very low. Another bug that I found was when it is a players turn to ask for a card the function shows the opponents hand rather than your own. I attempted to debug but could not figure it out without breaking my code. I also attempted to end the game when the deck was empty and players could no longer getting matching cards from each other but it would continue to ask players for cards although no more existed. The game right now ends when the center pile is empty.

Example Inputs and Outputs

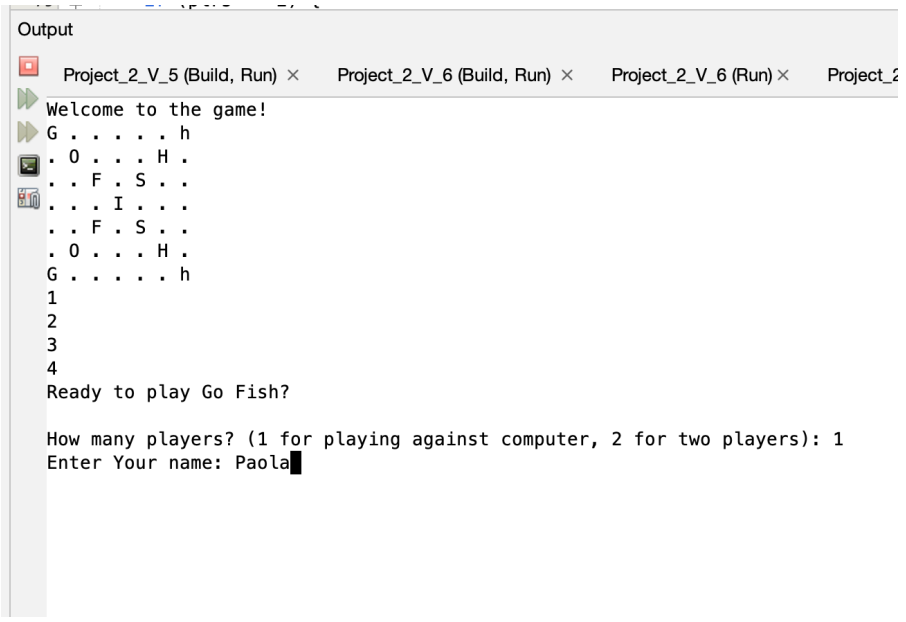


example output:
"How many players?"

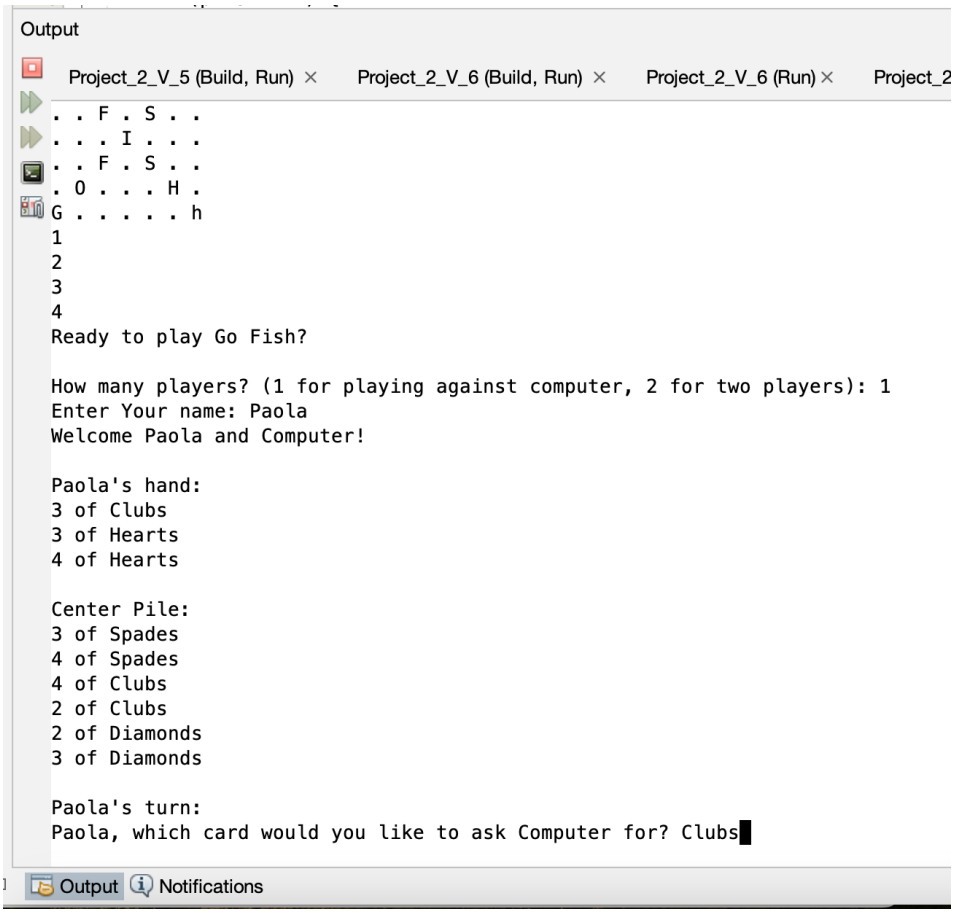
example input: 1

example output:
"Enter your Name"

example input: "Paola"



Example Inputs and Outputs

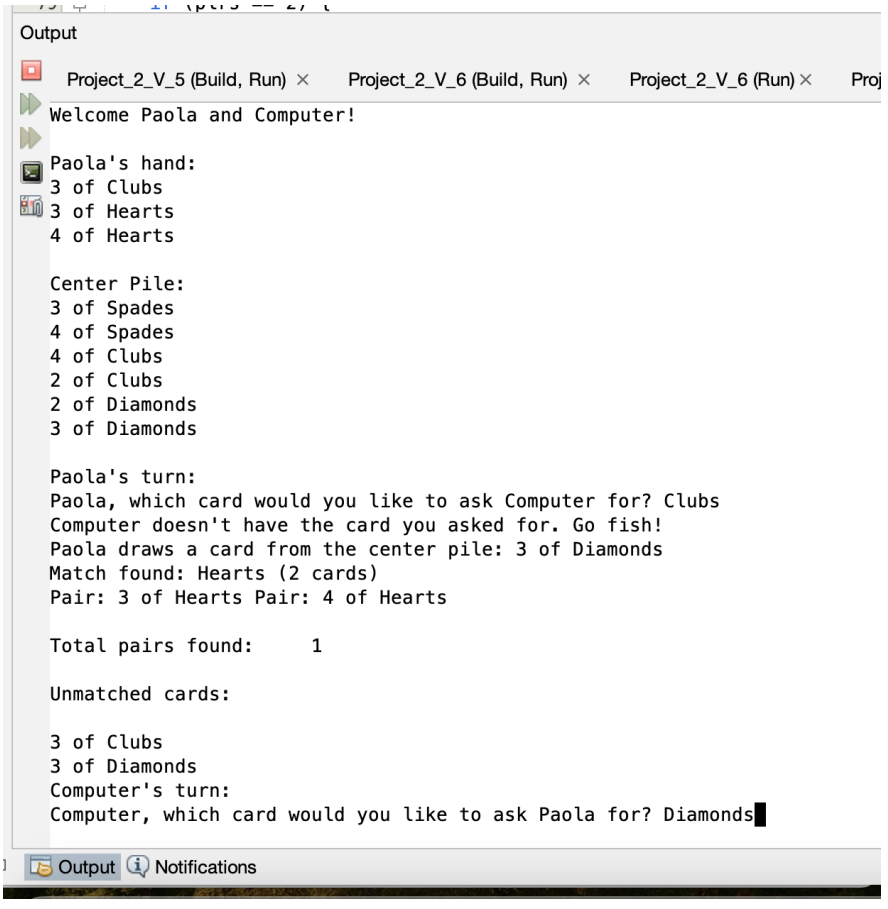


example output:
shows your hand
and asks you to ask
for a card

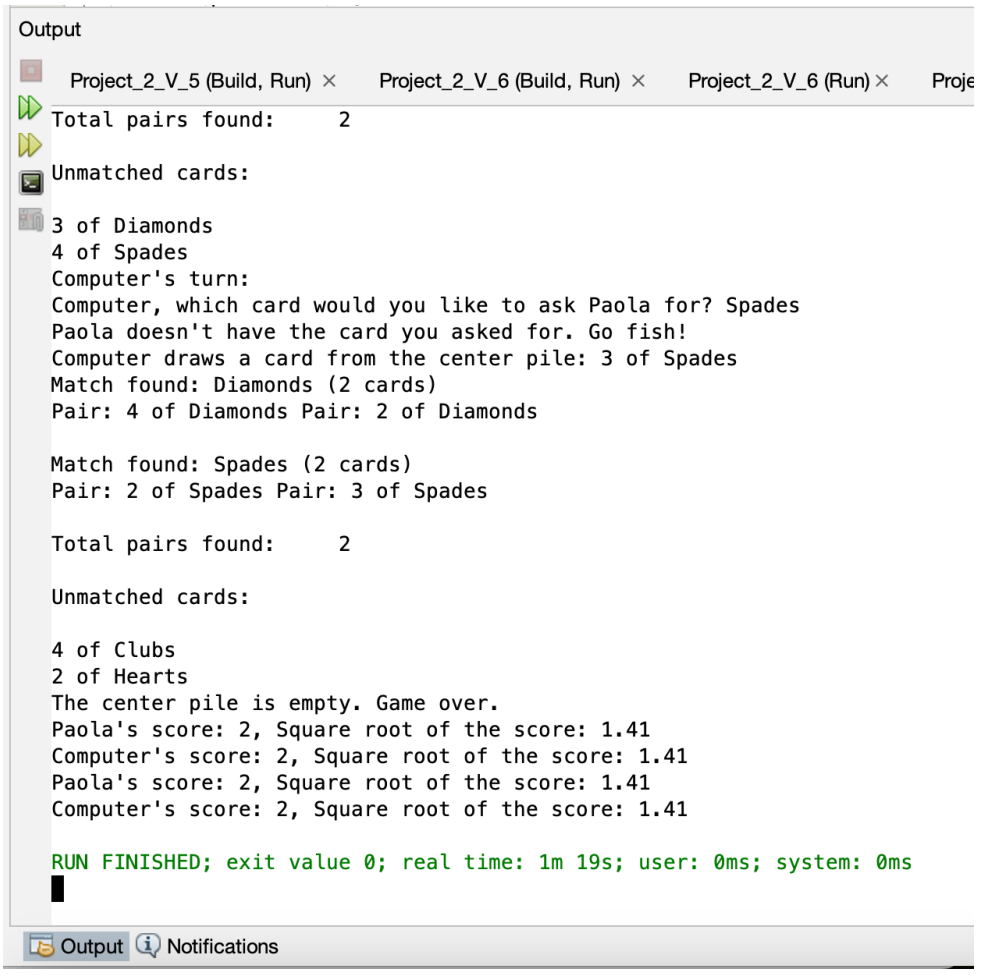
example input: Clubs
case sensitive
all suits start with an
uppercase letter

example output:
which card?
asks you to play as
the computer

example input:
Diamonds
case sensitive
all suits start with an
uppercase letter



Example Inputs and Outputs

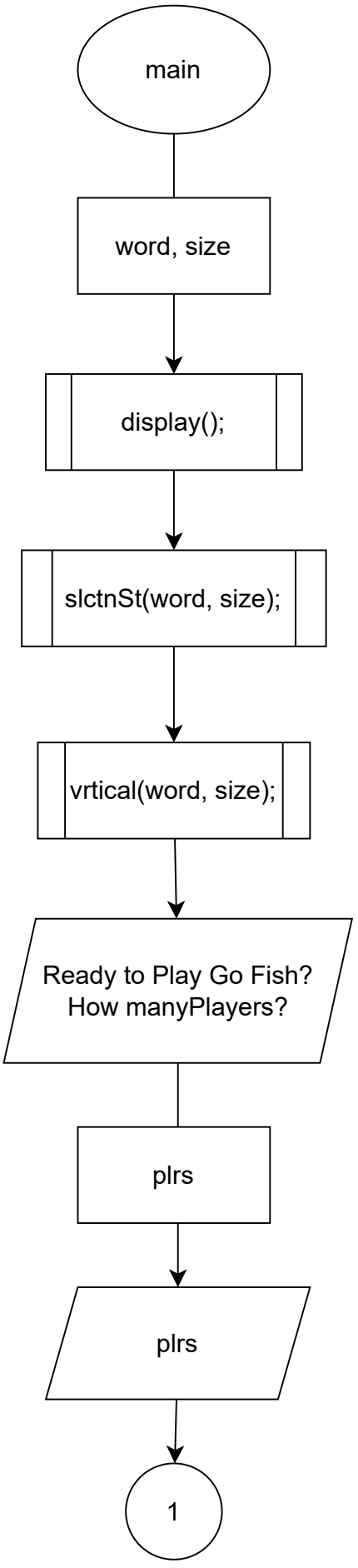


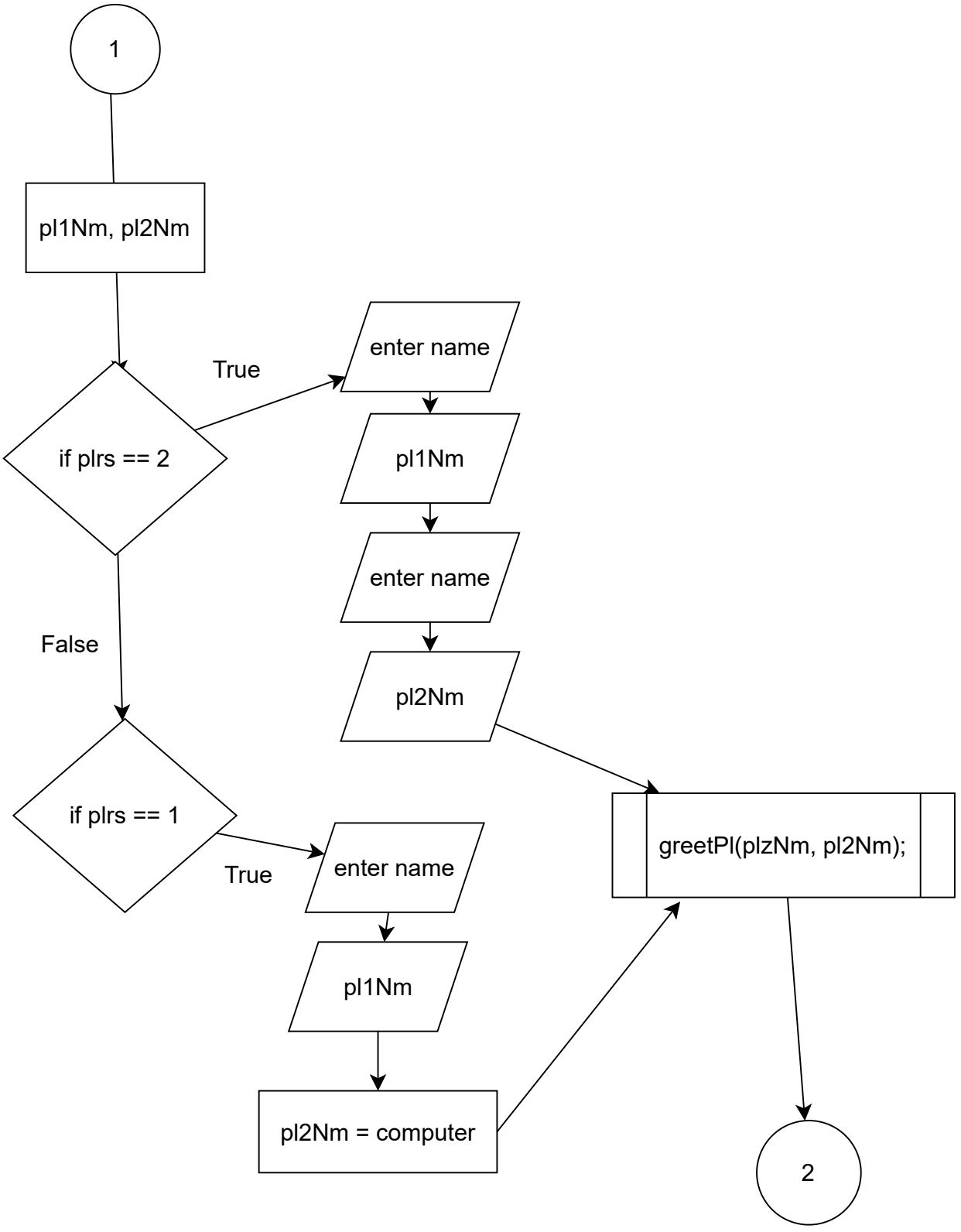
game will continue to ask for card input until center pile is empty
example output:
"The center pile is empty. Game over."

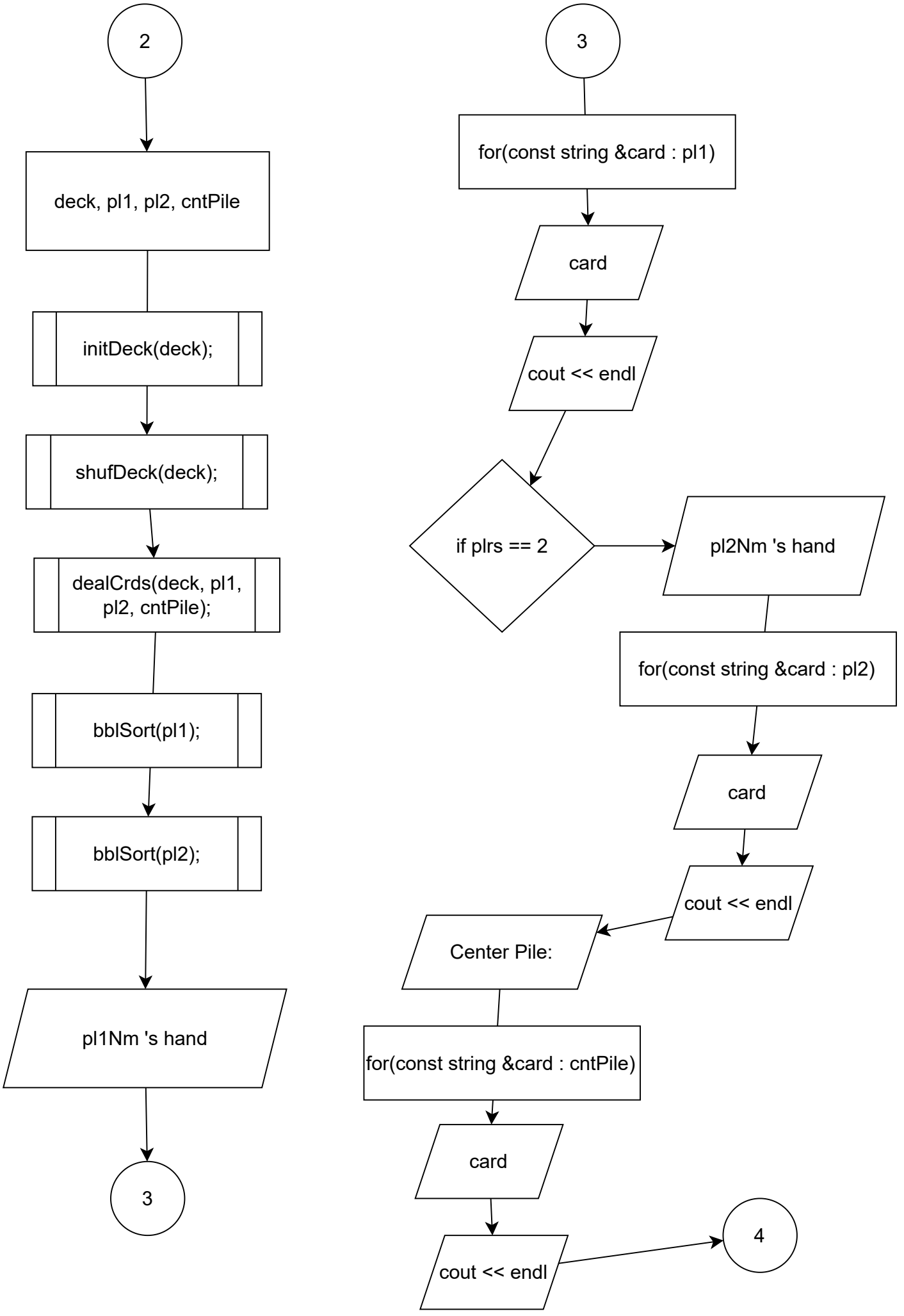
returns a score and the square root

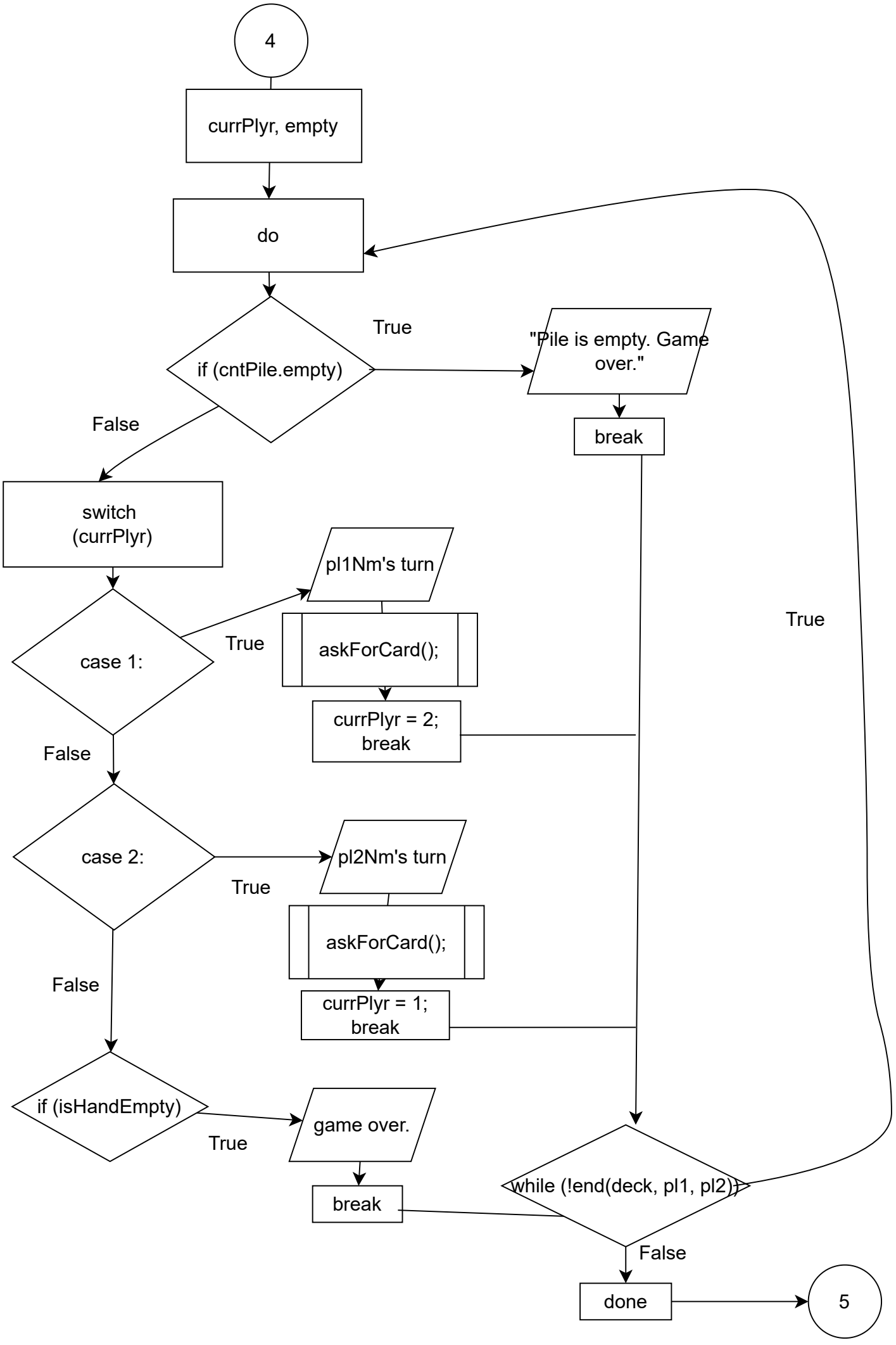
square root was implemented to incorporate cmath

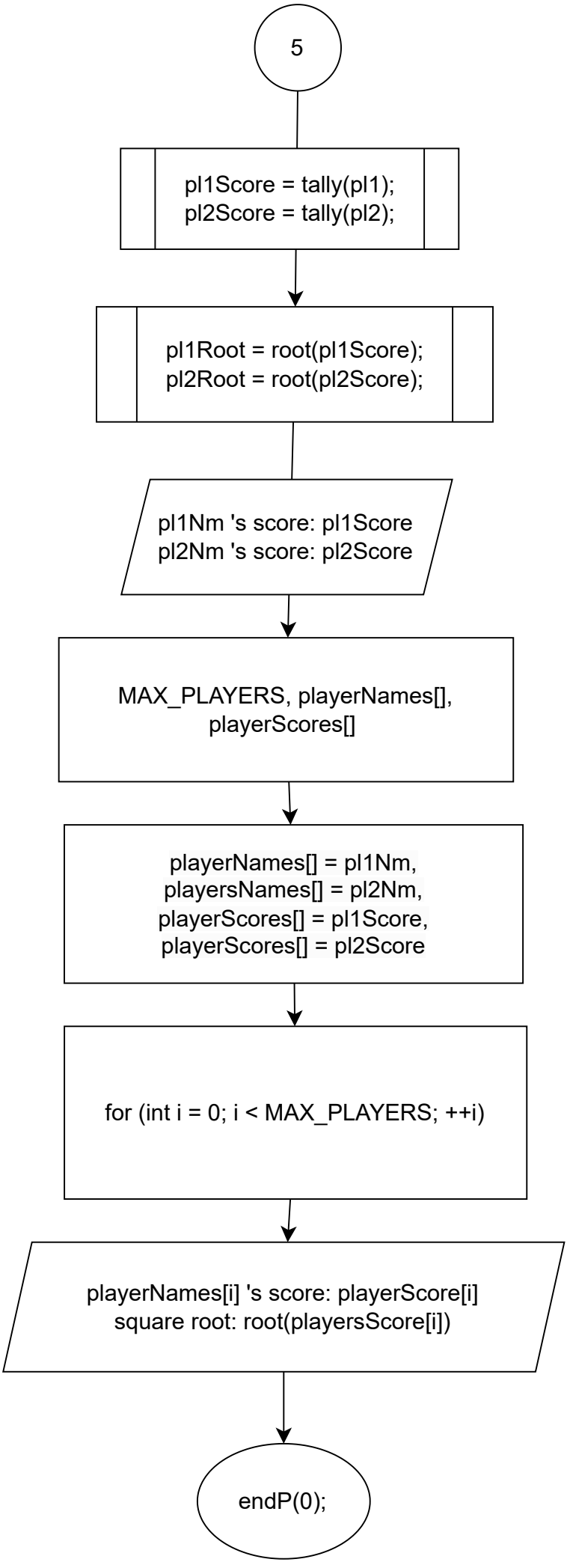
Flowchart


main
word, size
display();
slctnSt(word, size);
vrtical(word, size);
Ready to Play Go Fish?
How manyPlayers?
plrs
plrs
1


1
pl1Nm, pl2Nm
if plrs == 2
True
enter name
pl1Nm
enter name
pl2Nm
False
if plrs == 1
True
enter name
pl1Nm
pl2Nm = computer
greetPl(plzNm, pl2Nm);
2


2
deck, pl1, pl2, cntPile
initDeck(deck);
shufDeck(deck);
dealCrds(deck, pl1, pl2, cntPile);
bblSort(pl1);
bblSort(pl2);
pl1Nm 's hand
3
3
for(const string &card : pl1)
card
cout << endl
if plrs == 2
pl2Nm 's hand
for(const string &card : pl2)
card
cout << endl
Center Pile:
for(const string &card : cntPile)
card
cout << endl
4


4
currPlyr, empty
do
if (cntPile.empty)
True
"Pile is empty. Game over."
break
False
switch (currPlyr)
case 1:
True
pl1Nm's turn
askForCard();
currPlyr = 2; break
False
case 2:
True
pl2Nm's turn
askForCard();
currPlyr = 1; break
False
if (isHandEmpty)
True
game over.
break
while (!end(deck, pl1, pl2))
True
False
done
5


5
pl1Score = tally(pl1);
pl2Score = tally(pl2);
pl1Root = root(pl1Score);
pl2Root = root(pl2Score);
pl1Nm 's score: pl1Score
pl2Nm 's score: pl2Score
MAX_PLAYERS, playerNames[],
playerScores[]
playerNames[] = pl1Nm,
playersNames[] = pl2Nm,
playerScores[] = pl1Score,
playerScores[] = pl2Score
for (int i = 0; i < MAX_PLAYERS; ++i)
playerNames[i] 's score: playerScore[i]
square root: root(playersScore[i])
endP(0);