Project 1

Title

Blackjack Single Player Simulator

Course

CIS-17A

Section

43320

Due Date

May 17, 2021

Author

Annabelle Tamano

Table of Contents

1 Introduction

As the guideline for this project expect a recreation of existing card game, dice game, and/or board game to be recreated as a C++ program, I chose one to code that I am relatively familiar with, Blackjack.

Blackjack is a gambling game that typically allows multiple players, adopting rules that vary from where they are played. For the most part, I chose to base the project off the basic rules of the game. However, because this project is intended to be a text-based simulation, there are some limitations that I am working around through the process of developing the program. For those familiar with Blackjack, I chose to make significant changes as follows: this version is limited as a single player game between the user and the dealer, there are no doubles or splits, and the cards are represented by certain characters rather than pictures of full cards, omitting information such as the color of the card.

2 Game Play and Rules

When running the program, the user must first enter their name. Then, the user is sent to the main menu where they are given three options: to see the rules for Blackjack (‘1’), to play the game (‘2’), or to see the scoreboard (‘3’). The program keeps count of how many games are played and how much money is won out of all games played. The menu will flag that it recommends viewing the rules before playing if the number of games played is 0. Else, it will display the number of games played and the total money won out of all the games. The user then must enter ‘1’, ‘2’, or ‘3’ to indicate which action they would like to happen. After each action is executed, it sends user back to the main menu. To exit the program, the user must press any other key than ‘1,’ ‘2,’ or ‘3.’

2.1 Blackjack (Single Player) Rules

- The overall premise is to beat the dealer's hand without going over 21.
- To start the game, you must place a bet at minimum $20 and at maximum $2000.
- Certain cards are worth different values. Face cards (K, Q, J) are worth 10. Aces are worth 1 OR 11 (Whichever adds to a better hand). The rest of cards are worth the number that they display.
- Both you and the dealer start with two cards, however one of the dealer's cards is hidden until the end.
- At each play you have one of two options. First, you can 'Hit' which is just asking for another card. OR you can 'Stand' which signifies holding your total and ending your turn.
- If the value of your cards go over 21 you bust, and the dealer wins regardless of their hand. Thus, you lose all the money that you bet.

- If you are dealt 21 from the start, you got a blackjack! Therefore, you automatically win back 1.5x the value of your bet.
- The dealer will hit until his/her cards total 17 or higher. Both your hands will be compared, and if you beat the dealer's hand you win back 1.5x the value of your original bet. Else, you lose the value of your original bet.
- If you tie with the dealer or if your dealer busts, you neither win nor lose, and you get no money back.
- NOTE: In order to simplify the game, this version does not deal with splits or doubles.

2.2 Card Display Format

- Each card will be printed encapsulated by brackets [].
- First, the card name {A (Ace), 2, 3, 4, 5, 6, 7, 8, 9, T (10), J (Jack), Q (Queen), K (King)} will be printed.
- Then a character representative of the card's suit {C (Clubs), H (Hearts), D (Diamonds), S (Spades)} will be printed in the bracket.
- Example: [A C] represents the card "Ace of Clubs."

3 Development Summary

This project is a text-based C++ program, utilizing multiple files to create the game. I developed the project with the NetBeans 8.2 IDE, as required by this course. This project displays a mastery of concepts from Chapter 9 to Chapter 12 of the course textbook, Gaddis's *Starting Out with C++ from Control Structures to Objects*.
Total Lines of Code: 340 (Not including comments and black lines, there's still over 200.)

3.1 Version 1 Comments on Development

This version of the program is an incomplete version that does not run as intended, strictly to focus on the most important part of the program, the menu system. Allowing the user to select from the options presented by entering a character.

3.2 Version 2 Comments on Development

Version 2 builds off the previous version, implementing two specific functions: reading rules from a separate text file and shuffling the indexes of the deck. It is missing parts the parts that allow Blackjack to be played, however. But by isolating the shuffle algorithm, I was able to test it first before trying to get it to work for the blackjack game, thus helping me more efficiently create the program.

3.2.1 Rules: *cout* Statements Vs. Reading from Text File

In the previous version, the rules are printed from a series of *cout* statements within the *prntR* function. However, in this version, I wrote the rules in a separate text file, read them into a dynamically allocated string array, and printed them out.

3.2.2 Shuffling Deck

This version of the program implements the shuffling index algorithm found on the Geeksforgeeks.org website. To attempt to mimic the shuffling of cards in person, this algorithm first fills an array of indexes from 0-51. Then it goes through the array and exchanges the current element with another randomly selected one, ranging from its current index to the end, at 51. Of course, cards in real life are not shuffled with that extent of randomness, neither do they usually start in ascending order typically. In order to confirm that the deck indexes were shuffled, I printed the Card deck structure in the *playG* function.

3.3 Version 3 Comments on Development

The biggest addition to version 3 is the inclusion of a working blackjack game within the *playG* function, which follows the rules of as stated in the rules section. When creating this section, I had to look further into the rules of Blackjack to account for situations such as when the dealer busts or if the player and the dealer tie.

3.4 Version 4 Comments on Development

This version differs from its previous version by grouping certain variables together into a struct called *User* rather than storing them through independent variables.

3.4.1 Purpose of *User* Structure

Through setting up the struct, *User*, this acts as a setting stone to build the *seeSB* and *setSB* functions that deal with comparing user data from the past times that this function is ran. The Scoreboard is meant to display the top 5 players who have won the most money. Consequentially, it stores information like the player's name, the money won, and the number of games played in total. This version that I built utilizes the struct *User* to account for the information that will later be asked for once the functions relating to the scoreboard will be built so that it all can be found in the same place.

3.5 Version 5 Comments on Development

This version aims to strengthen my display on my ability to read and write files, both plain text files and binary files. Although, I do struggle a little bit to get some parts of the binary file work to function as intended. Despite that, it was a nice challenge to include these specific concepts into my program.

3.5.1 Reading *.txt* Files: For Loop vs. While Loop

Different from past versions, this one specifically takes it a step further by reading in the "rules.txt" file in with a while loop, allowing the file to have as many lines as possible. I thought this made the program more efficient and it was originally intended to be coded this way when I first thought of the project. Therefore, it felt necessary to include.

3.5.2 Binary File for Scoreboard

In this version, this is the first time I attempt to include the scoreboard. I utilize two functions, *setSB()* and *seeSB()*. The first is just a test function that allows the user to input 5 players and their information, name, total money won, and total games played. This is written to a binary file "sb.bin" and the function is finished. The second reads "sb.bin" into a pointer of the struct User, then it prints out all the consequential information from *setSB()*. I had a hard time getting this part to work as intend, as the first name that I inputted would not print properly. However, I figured that the issue was that in the *User* struct, name was a char pointer and not simply a C-string.

4 Specifications

4.1 Pseudo-code

The following pseudo-code was developed before the completion of the entire project. Despite this not exactly matching with the naming conventions of all parts of the program, it still represents what specific parts of the program are meant to do.

4.1.1 *main* Function

```
Ask user for name and store in tempN
Initialize User plyr
do {
   Print the main menu with three options 1. See rules, 2. Play game, 3. See scoreboard
   Ask user for option 1,2, or 3 and read in opt
   opt is 1? Call prntR function
   opt is 2? Call playG function
   opt is 3? Call seeSB function
   opt is none of the above? great user goodbye.
} while (opt is 1-3);
delete tempN
return 0 to end Function
```

4.1.2 *prntR* Function

```
Declare variables outpF and n.
Open in outpF the file "rules.txt"
```

Allocate memory into outpS
while you can read line from outpF into outpS[n] {
  increment n }
close outpF
delete outpS

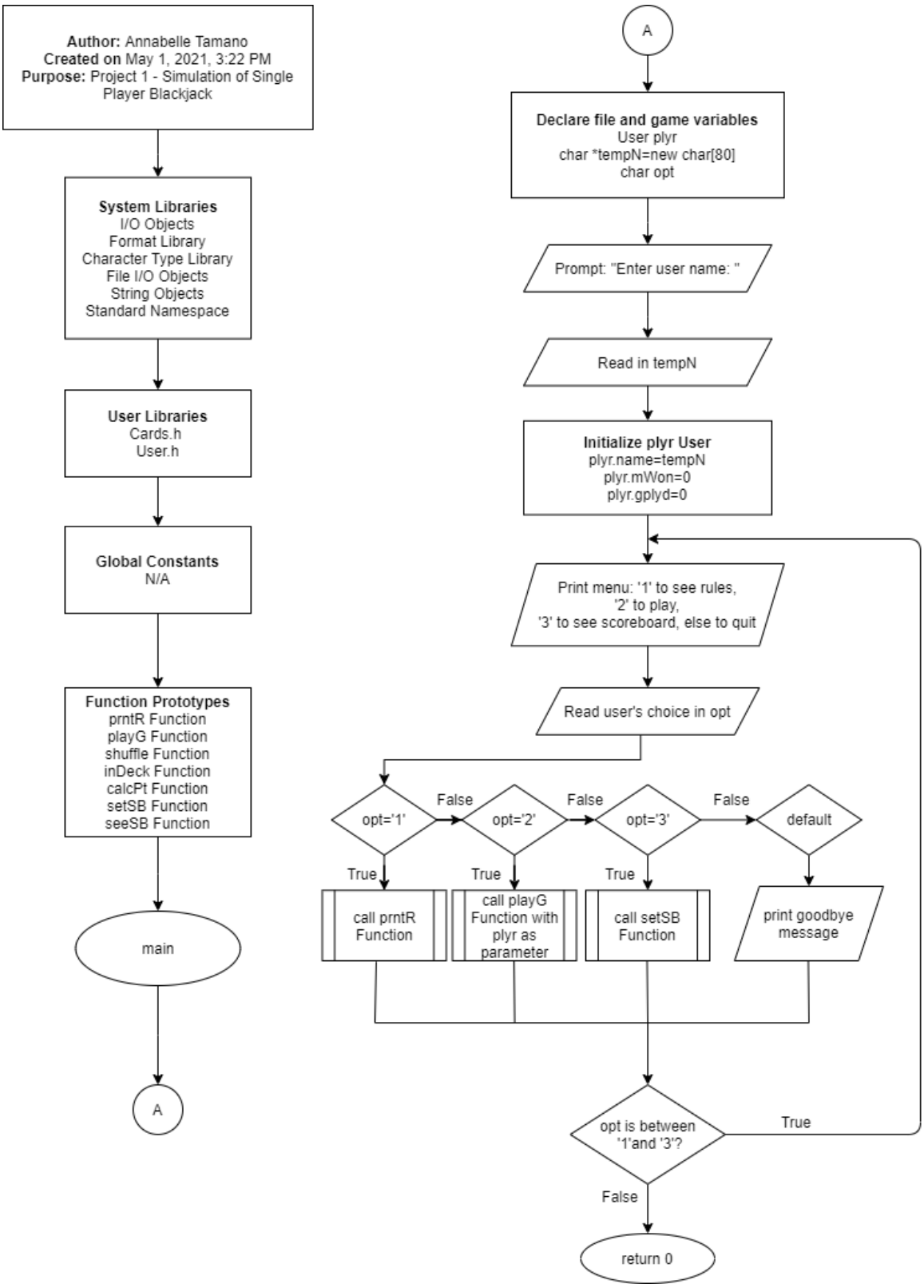
4.1.3 *playG* Function

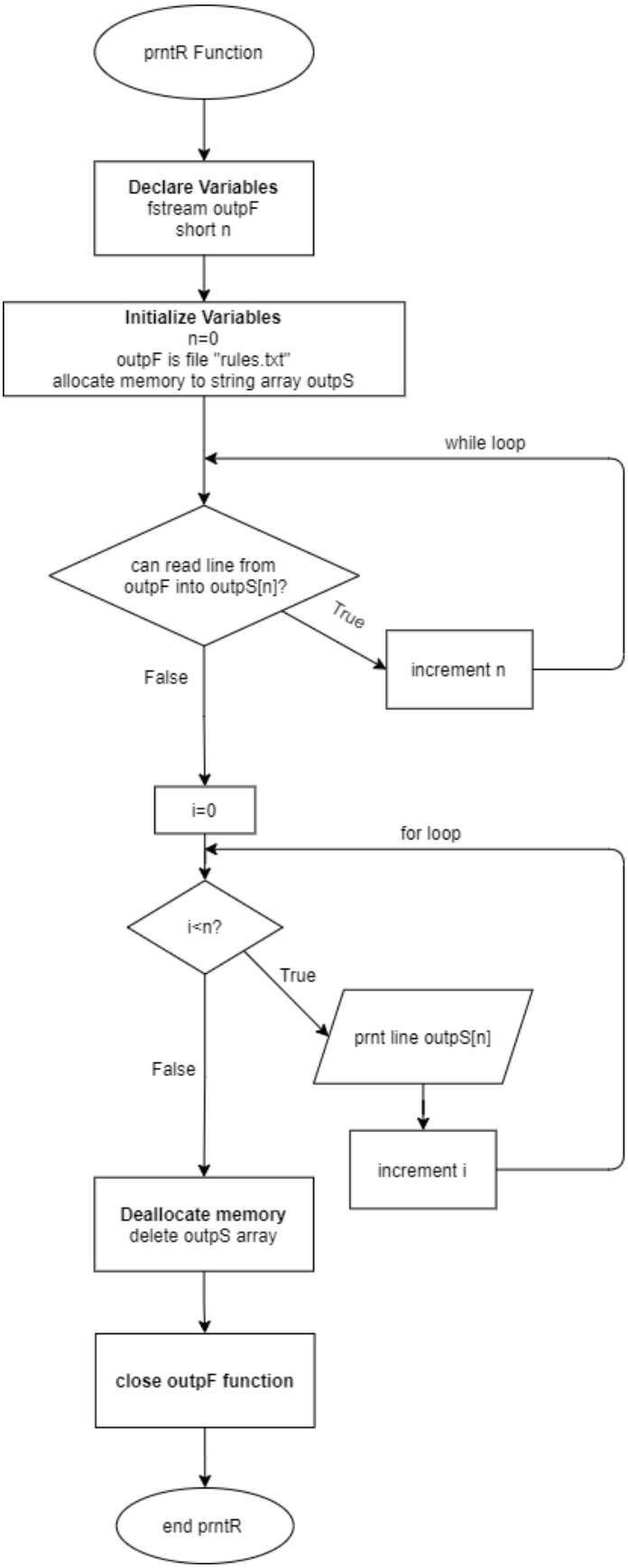
Initialize Card deck and shuffle deck indexes
Set space for user and dealer's hands
Deal the first two cards for both user and dealer
Calculate total points for user and dealer
Start game prompting user to place bet and reading in bet
do {
  Use for loop to print user's deck and then print total
  Print dealer's first card and indicate that second card is hidden
  usrPt is over 21? User automatically loses all the money they originally bet, run ends
  usrPt is exactly 21? User automatically wins 1.5x the money they originally bet, run ends
  else?
    User is asked to hit or stand; read in usrM
    If hit? A new card is added to user's hand and total points are calculated
    If stand?
      Dealer adds card to hand until point value is greater of equal to 17
      Compares both hands
      If user's hand greater than dealer's? You win 1.5x the money you bet
      If dealer's hand greater than user's? You lose the money you bet
      If you tie or if they dealer busts? You neither win nor lose any money
      run ends
} while run does not end
Increment gPlyd in plyr
Set scoreboard
Deallocate all the memory created for this function

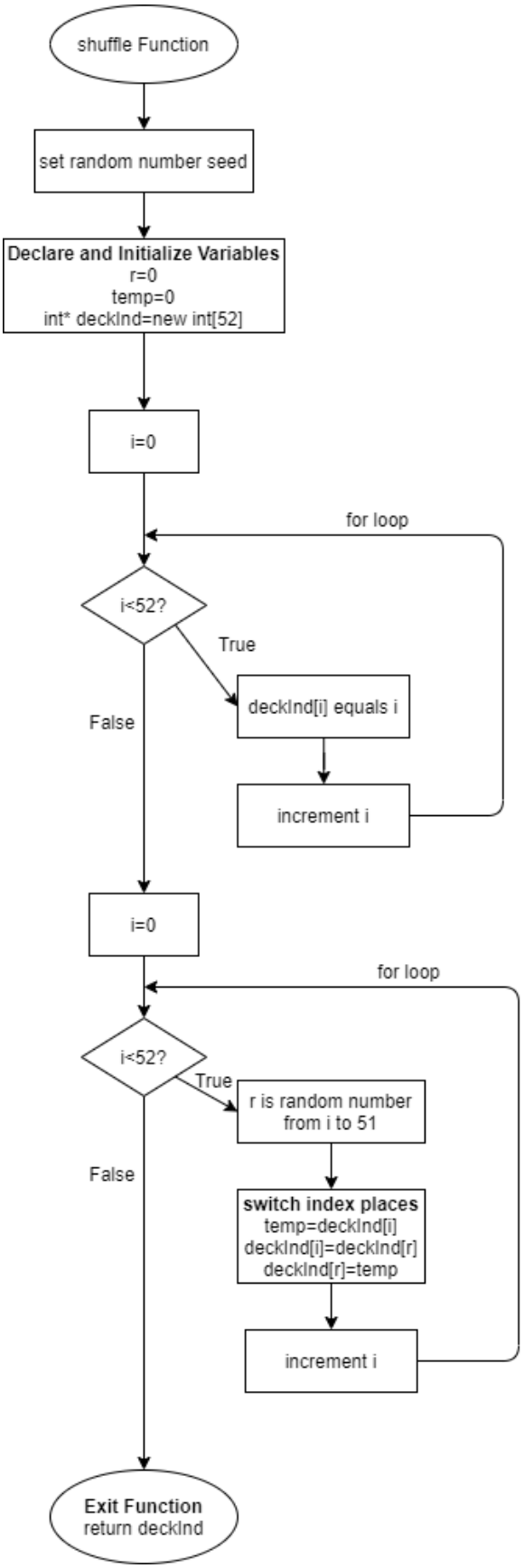
4.1.4 *seeSB* Function

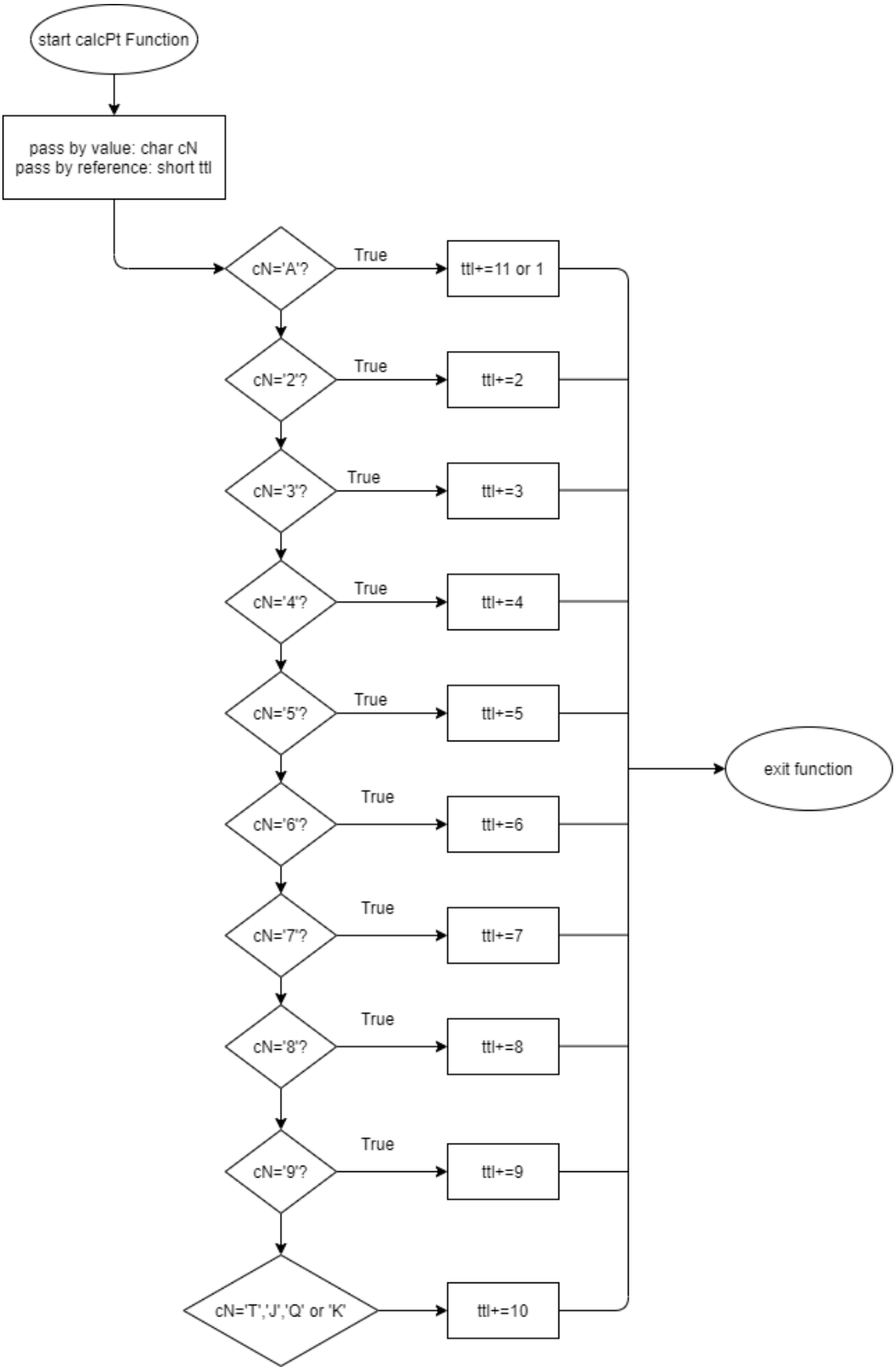
Open file "sb.bin"
Allocate Memory for scoreboard User array
Read file "sb.bin" into sb array
Print User, Games Played, and Money Won at top of table
for loop from i=0 to i<5 {
  Print name, games played and money won }
Deallocate memory
Close file

4.2 Program Flowcharts




prntR Function
Declare Variables
fstream outpF
short n
Initialize Variables
n=0
outpF is file "rules.txt"
allocate memory to string array outpS
while loop
can read line from
outpF into outpS[n]?
True
increment n
False
i=0
for loop
i<n?
True
prnt line outpS[n]
False
increment i
Deallocate memory
delete outpS array
close outpF function
end prntR


shuffle Function
set random number seed
Declare and Initialize Variables
r=0
temp=0
int* deckInd=new int[52]
i=0
for loop
i<52?
True
deckInd[i] equals i
False
increment i
i=0
for loop
i<52?
True
r is random number
from i to 51
False
switch index places
temp=deckInd[i]
deckInd[i]=deckInd[r]
deckInd[r]=temp
increment i
Exit Function
return deckInd


start calcPt Function
pass by value: char cN
pass by reference: short ttl
cN='A'?
True
ttl+=11 or 1
cN='2'?
True
ttl+=2
cN='3'?
True
ttl+=3
cN='4'?
True
ttl+=4
cN='5'?
True
ttl+=5
cN='6'?
True
ttl+=6
cN='7'?
True
ttl+=7
cN='8'?
True
ttl+=8
cN='9'?
True
ttl+=9
cN='T','J','Q' or 'K'
ttl+=10
exit function

4.3 Concepts Used

Chapter 9: Pointers/Memory Allocation

- ~~Pointer Variables~~
- ~~Arrays/Pointers~~
- ~~Function Parameters~~
- ~~Memory Allocation~~
- ~~Return Parameters~~

Chapter 10: Char Arrays and Strings

- ~~C-Strings~~
- ~~Strings~~

Chapter 11: Structured Data

- ~~Arrays~~
- Nested Structures
- ~~Function Arguments~~
- ~~Function Return~~
- ~~Pointers~~
- Enumeration

Chapter 12: Binary Files

- Formatting
- Function Parameters
- Member Functions
- ~~Multiple Files~~
- ~~Binary Files~~
- Records with Structures
- Random Access Files
- ~~Input/Output Simultaneous~~

5 Errata Report

There are a few errors that deviated from my original vision for this project. Such errors altered the flow of the program and changed the overall purpose of some functions.

The main error is that I planned to set the scoreboard every time a game is played, to include the current player's stats on the list, in the case that they make the top five. However, I did not know how to incorporate an already existing binary file so that list would show.

Next, I wanted to store the deck as a record of the structure Card that could be searched through. I feel as if this would make the function slightly more efficient. However, I failed to give myself allotted time to get this test to work, ultimately deleting it out of my Version 5.

6 References

To refresh my knowledge of the basic rules of Blackjack, I utilized the following website: http://www.hitorstand.net/strategy.php

I utilized the following reference to understand a basic algorithm on how to shuffle the deck of cards: https://www.geeksforgeeks.org/shuffle-a-deck-of-cards-3/