**Project 2**

Title

**Drewbage Cribbage Game V.2**

Course

**CSC-5**

Section

**40651**

Due Date

**Feb 12, 2014**

Author

**Drew Allensworth**

Table of Contents

[Introduction 3](#_Toc379960195)

[Game Play and Rules 3](#_Toc379960196)

[Flowcharts 10](#_Toc379960197)

[Pseudocode 11](#_Toc379960198)

[Version 1 14](#_Toc379960199)

[Version 2 Additions 18](#_Toc379960200)

[Function Descriptions and Comments 20](#_Toc379960201)

[Version 1 20](#_Toc379960202)

[Version 2 Additions 25](#_Toc379960203)

[Major Variables and Objects 30](#_Toc379960204)

[Version 1 30](#_Toc379960205)

[Version 2 Additions 31](#_Toc379960206)

[Concepts Used 32](#_Toc379960207)

[Program Listing 34](#_Toc379960208)

**Drewbage** *Cribbage... Distilled.*

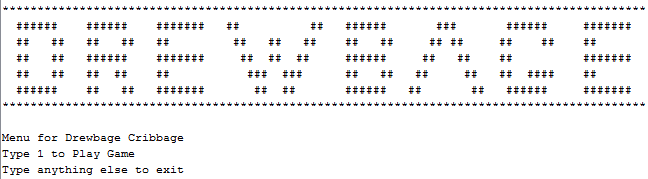
# Introduction

At the outset of this project, I knew that one of the most difficult games for me to implement with any modicum of success would be a card game. The difficulty, as I saw it, would be in attempting to accurately model the behavior of having cards in one’s hand and then using them. That is, to assign cards to a hand from a “deck”—insuring no duplicates could go to any player—and then removing those cards from one’s hand to use in gameplay. Even if that was implemented successfully, I still needed to create ways to count the points in a hand, allowing the player to use the hand to their advantage. Despite the difficulty, I desired to impress. In addition, Cribbage holds a special place for me as it’s the only card game that my family played during my childhood and still continues to play. It’s the game that I would go to my grandmother’s house to play before she became ill. Any day that we would travel to her house, we would wait till after dinner and her, my family, and I would play cribbage together.

# Game Play and Rules

Drewbage cribbage is two-player cribbage. The Player (user) will be playing against the computer. There is nothing the user needs to do for the computer—it will take care of its own play.

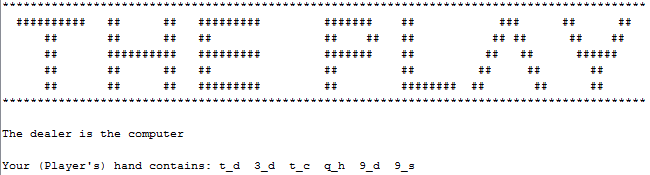
The game starts with a Menu. “Type 1 to Play Game” and pressing any key to quit the program.



Before the game begins, the results of the previous game are displayed above the banner announcing the beginning of “the play”.



At the outset of the game, the dealer is determined randomly and both the computer and the Player are dealt six cards. The Player should notice that the dealer is displayed.



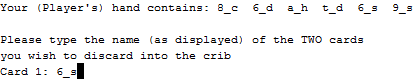
The first action the Player will undertake is discarding two cards into the “crib”. The “crib” hand is not used during the pegging portion of the game. However, rest assured that although you as the Player are not using it and will not see its contents for a little while, the crib has been filled properly and contains the cards given to it. As it stands right now, the game that the Player will be playing next is the portion of cribbage called The Play or “pegging”. This occurs after the dealer is announced and two cards from each player have been given to the crib. Any points later derived from the crib will go to the dealer. So, if you’re the dealer you want to maximize your crib points potential. However, if you aren’t the dealer you want to minimize the crib points potential.

At this point, I would like to take the time to detail some of the codes used in the program:

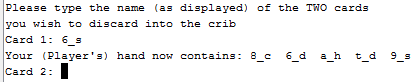
Each card name is composed of three characters; the rank, the “of”, and the suit. For example: a\_s equals “ace of spades”. The breakdown is as follows: “a” stands for “ace”, the underscore for “of”, and the “s” for spades.

|  |  |
| --- | --- |
| \_ underscore character | “of” |
| ***Ranks*** | |
| a | Ace |
| 2 | two |
| 3 | three |
| 4 | four |
| 5 | five |
| 6 | six |
| 7 | seven |
| 8 | eight |
| 9 | nine |
| t | ten |
| j | Jack |
| q | Queen |
| k | King |
| ***Suits*** | |
| c | clubs |
| d | diamonds |
| h | hearts |
| s | spades |

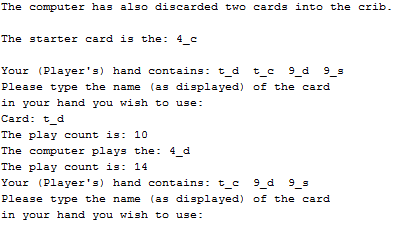
Now that we have the card codes detailed, let’s continue with the game play. Assuming the following:



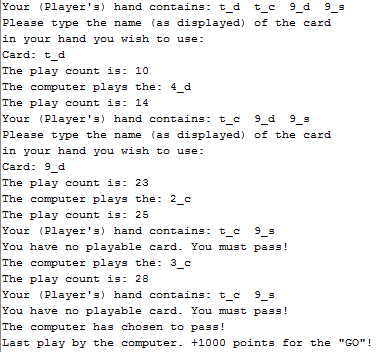
Let’s enter that “6\_s” (which is the “six of spades”)



And there you have it! The computer accepted the card, put it in the “crib”, and removed it from “your” hand. You’re going to do the same for the second card and then move on to The Play portion of the cribbage game. At that point, the “Up Card” will be displayed. The Up Card will come into play after The Play portion of the game is finished. The following image gives more information. Note also that the computer put its two cards into the crib, just like you did. Because the computer is the dealer in this particular game (it’s assigned randomly) you get to play your card first.



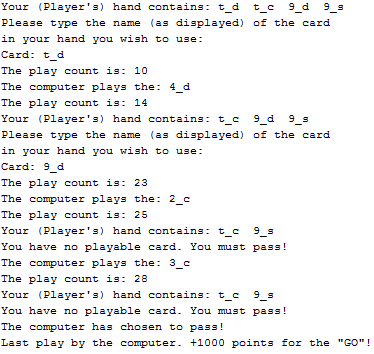
Two turns later:

  
Since you need to start thinking in terms of points, here is the table of values for each card rank:

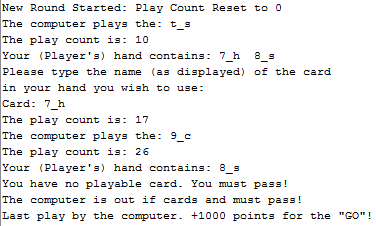
You play the ten of diamonds. The card is removed from your hand. The count is 10. The computer plays the four of diamonds and the count is now 10 + 4 = 14 This then continues until the score reaches as close to 31 as possible without going over 31.

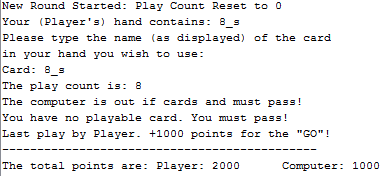
|  |  |
| --- | --- |
| **Rank** | **Value** |
| Ace | 1 |
| two | 2 |
| three | 3 |
| four | 4 |
| five | 5 |
| six | 6 |
| seven | 7 |
| eight | 8 |
| nine | 9 |
| ten | 10 |
| Jack | 10 |
| Queen | 10 |
| King | 10 |

Before I leave you to play the game for yourself, I would like to go over a few final things:



If you follow the count, you’ll notice that at a certain point your hand may only contain a card or cards that will send the count over 31. Don’t worry; the program will know that you don’t have a playable card in your hand for that given count. In this case it will “pass” for you. That is, you don’t have to pass—the computer knows that you can’t play your cards with the particular play count.If you still have cards at the end of a particular “round”, you will play another round until both you and the computer run out of cards. At that point, the points will be tallied and displayed.

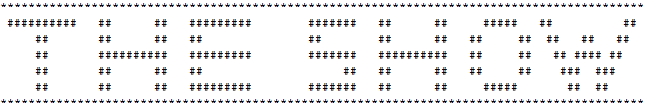




Finally, the following is the scoring breakdown for The Play. It’s non-traditional, but so is Drewbage.

|  |  |
| --- | --- |
| Make the play count equal to 15 | 2000 points |
| Make the play count equal to 31 | 4000 points |
| Last player in a game that ends at < 31  What’s known as a “GO” | 1000 points |

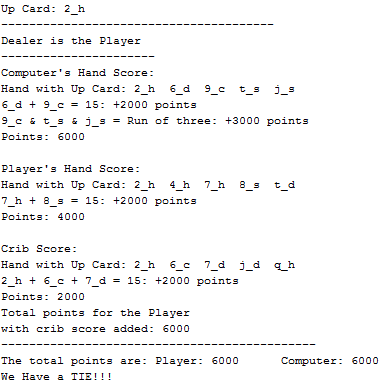
After The Play is finished, the next portion of the game is The Show. This portion is decidedly less involved from a player’s point-of-view, but it is the result of your decisions earlier in the game. Specifically, how many points your receive is based on what you kept in your hand after discarding to the crib, what the up card is, and whether you are the dealer and get to add the crib score to your points total.



The following is the scoring breakdown for The Show. Once again, it’s non-traditional.

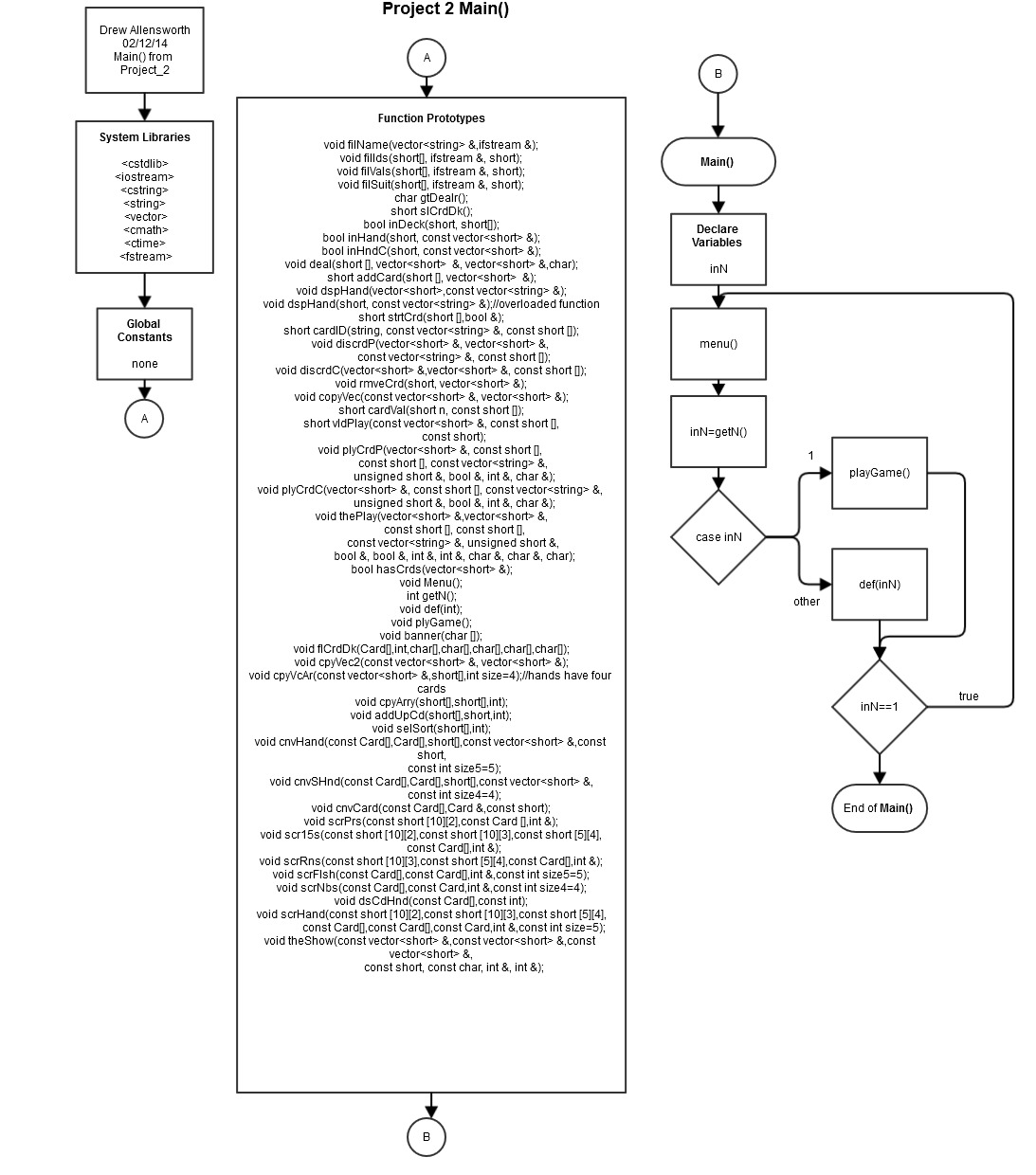
|  |  |
| --- | --- |
| A pair of any two cards | 2000 points |
| Any combination of cards equal to 15 | 2000 points |
| Any run of three or more cards | 1000 points for each card |
| A flush (same suit) of all five cards | 5000 points |
| A flush of all four cards in one’s hand not including the Up Card | 4000 points |
| Having a Jack in one’s hand when the Up Card is a Jack (“nibs”) | 1000 points + a bonus of 20% of the points total |

Below is an image of all three hands being scored and the crib hand’s points being added to the dealer. The Points below the hand scores reflects the scores from the hand being added to the points from The Play.



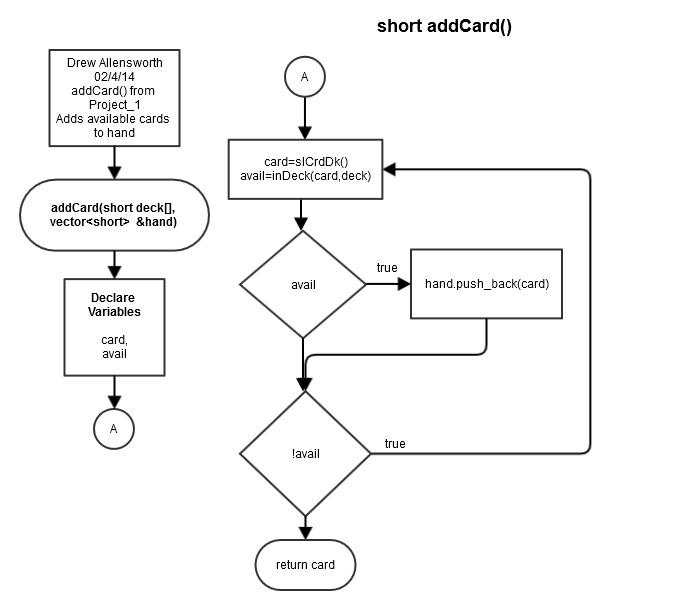
You can see from the above image that in this game both the Player and the computer finished the game in a tie. This result will be saved and displayed the next time the game is played.

# Flowcharts



# 

# 



# Pseudocode

## Version 1

***Four parallel arrays\**** *Example:*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sub pos. | Sub 0 | Sub 1 | Sub 2 | Sub 3 | Sub 4 | Sub 5 |
| crdID | 0 | 1 | 2 | 3 | 4 | 5 |
| crdName | “a\_c” | “a\_d” | “a\_h” | “a\_s” | “2\_c” | “2\_d” |
| crdSuit | 1 | 2 | 3 | 4 | 1 | 2 |
| crdVal | 1 | 1 | 1 | 1 | 2 | 2 |

\*crdName must be a vector to hold strings

**Pseudocode**

1. ~~Display splash graphics~~
2. Display menu
   1. 1. Play game
   2. ~~2. Help~~
   3. ~~3. High Scores~~
   4. Any other key to Quit
3. Get menu choice
4. (Assuming case == “Play Game”)
   1. Initialize and fill the card arrays and vector for card ID, Values, Suits, and Names
   2. Initialize playerPoints=0, computerPoints=0
   3. Have the computer randomly determine who gets dealt first and who gets “nibs”. Store the outcome dealer.
      1. Alert the player
   4. Deal the player and the computer six cards each
      1. Start with a “deck” of 52 unique cards
         1. If dealer is for the player
            1. Loop for six times

Randomly pick a card ID for the player

Insure that card hasn’t been chosen

Assign that card ID to the player’s “hand”

Remove the possibility of that card being picked again

Pick a card ID for the computer

Insure that card hasn’t been chosen

Assign that card ID to the computer’s “hand”

Remove the possibility of that card being picked again

* + - * 1. End loop
      1. Else dealer is for the computer
         1. Loop for six times

Pick a card ID for the computer

Insure that card hasn’t been chosen

Assign that card ID to the computer’s “hand”

Remove the possibility of that card being picked again

Randomly pick a card ID for the player

Insure that card hasn’t been chosen

Assign that card ID to the player’s “hand”

Remove the possibility of that card being picked again

* + - * 1. End loop
  1. For each card in the player’s “hand”, display ~~a graphic of the card and~~ the card’s name, i.e. ace\_S or queen\_H (ace of spades and queen of hearts respectively)
     1. ~~Using the card name and files that separately store each card’s graphic~~
     2. Display the card name which the player will use to reference the card ~~next to the image~~
  2. Randomly select a card ID as the starter card from the “deck” and store the selection
     1. Remove the possibility of that card being picked again
     2. Display ~~a graphic of the card and~~ the card’s name
     3. If the card is a Jack
        1. Give two points to either the player or computer based on dealer
        2. Alert the player of who got the points
  3. Have the player discard two cards
     1. Loop two times
        1. Show Player’s “hand”
        2. Ask player to type a card name to discard
        3. Use the card name input to find the number ID
           1. Loop –If card does not exist, alert player and get card name again
           2. Loop –If card is not in player’s “hand”, alert player and get card name again
        4. Assign card ID to crib “hand”
        5. Remove the ID from the player’s “hand”
     2. End loop
  4. Have the computer discard two cards
     1. Loop two times
        1. Randomly select a card to discard using the sub values
           1. Loop –If card ID is not valid, select another sub value
        2. Assign card ID to crib “hand”
        3. Remove the ID from the computer’s “hand”
     2. End loop
  5. Start The Play
     1. Make a copy of the Player’s “hand”
     2. Make a copy of the computer’s “hand”
     3. Initialize unsigned short playCnt=0 as the variable to hold the count for the gameplay
     4. Initialize bool computerPass=false, playerPass=false
     5. Declare char lastPlay//valid values are ‘p’ for player and ‘c’ for computer
        1. If dealer is for the player—start with the computer
           1. Loop do {

Have the computer select a card from “hand” to use using the sub values

Validate: Loop –Start at sub 0. If card ID <0, move to next sub value < vector.size()

Validate: Loop –if playCnt + the card selection > 31, move to next sub value < vector.size()

Validate: If no card meets these criteria , display that the computer has passed and return computerPass=true to allow player to select a card

If computerPass=false

Display the card name of computer’s selection

Remove the selected card from the computer’s “hand”

Set playCnt = playCnt + crdVal

If playCnt == 15 then give the computer 2 points

If playCnt == 31 then give the computer 2 points

Set lastPlay=’c’

Display playCnt

Let the Player select a card

Display Player’s “hand”

Ask player to type a card name to use

Use the card name input to find the number ID

Validate: Loop –If card does not exist, alert player and get card name again

Validate: Loop –If card is not in player’s “hand”, alert player and get card name again

Validate: Loop –if playCnt + the card selection > 31, alert player and get card name again

If player types ‘P’, set playerPass=true

If playerPass=false

Remove the selected card from the Player’s “hand”

Set playCnt = playCnt + crdVal

If playCnt == 15 then give the Player 2 points

If playCnt == 31 then give the Player 2 points

Set lastPlay=’p’

Display playCnt

Display Player’s “hand”

* + - * 1. } while(playCnt<31&&(playerPass==false||computerPass==false)
      1. If dealer is for the computer—start with the player
         1. Loop do {

Display Player’s “hand”

Display playCnt

Ask player to type a card name to use

Use the card name input to find the number ID

Validate: Loop –If card does not exist, alert player and get card name again

Validate: Loop –If card is not in player’s “hand”, alert player and get card name again

Validate: Loop –if playCnt + the card selection > 31, alert player and get card name again

If player types ‘P’, set playerPass=true

If playerPass=false

Remove the selected card from the Player’s “hand”

Set playCnt = total + crdVal

If playCnt == 15 then give the Player 2 points

If playCnt == 31 then give the Player 2 points

Set lastPlay=’p’

Display playCnt

Display Player’s “hand”

Let the computer select a card

Have the computer select a card to use using the sub values

Validate: Loop –Start at sub 0. If card ID <0, move to next sub value < vector.size()

Validate: Loop –if playCnt + the card selection > 31, move to next sub value < vector.size()

Validate: If no card meets these criteria , display that the computer has passed and return computerPass=true to allow player to select a card

If computerPass=false

Display the card name of computer’s selection

Remove the selected card from the computer’s “hand”

Set playCnt = playCnt + crdVal

If playCnt == 15 then give the computer 2 points

If playCnt == 31 then give the computer 2 points

Set lastPlay=’c’

Display playCnt

* + - * 1. } while(playCnt<31&&(playerPass==false||computerPass==false)
      1. If(playerPass==true && computerPass==true)
         1. If lastPlay==’p’ then give the player one point
         2. Else give the computer one point
      2. While ((Player still has cards) ||(computer still has cards)){
         1. This time, instead of choosing who goes first based on dealer, it’s based on the value in lastPlay.
         2. Reset computerPass = False and playerPass = False
         3. Start “The Play” above
         4. }
    1. Output scores for both the computer and the player
       1. If there is a nibs bonus, apply it to the dealer
    2. Calculate winner
       1. If playerPoints>computerPoints then Player wins
       2. Else computer wins
  1. ~~Get a player’s name or handle~~
  2. ~~Store name in a file with the score~~

1. ~~End of case “Play Game”~~
2. ~~(Assuming case == “Help”)~~
   1. ~~Use an outputfile object to output help text~~
3. ~~(Assuming case == “High scores”)~~
   1. ~~Use an outputfile object to output all scores in the scores file~~

## Version 2 Additions

**Pseudocode**

1. Create card structure
   1. Int ID
   2. Int Rank
   3. Int Suit
   4. String Name
   5. Int Val
2. Create and initialize the “sets” of possible card combinations for scoring
3. Create an array with 52 card structure elements
4. Fill the crdDeck array element members from files
5. Using “hands” passed from plyGame()
   1. Create copies to use for the program
6. Make copies of the copied hands and sort them in ascending order
7. For each hand, take the sorted hand and loop through the array filling an array of card structure elements based on the ids in the sorted hand.
8. Create copies of both the player’s hand, the computer’s hand, and the crib with the up card added.
9. Based on dealer, start with either Player first or computer
   1. Score hands for pairs
   2. Score hands for 15s
      1. Sets of two
      2. Sets of three
      3. Sets of four
      4. Sets of five
   3. Score hands for runs
      1. Start with runs of five—if found stop.
      2. Then Four—if found stop.
      3. Then Three
   4. Score hands for flush
      1. Score the hand for a flush with the up card
         1. If even one card suit is different, no flush
      2. Score the hand for a flush without the up card
         1. If even one card suit is different, no flush
   5. Score hands for nibs
      1. Use hands with the card structure that don’t contain the up card
      2. If the up card is a Jack and the hand contains a jack—Add nibs points
10. Score the crib with the same methods as above.
    1. Assign points to dealer
11. Output scores for both the computer and the player
    1. If there is a nibs bonus, apply it to the dealer
12. Calculate winner
    1. If playerPoints>computerPoints then Player wins
    2. Else computer wins
13. Store winner in a file with the score
14. End of case “Play Game”

# Function Descriptions and Comments

## Version 1

|  |  |
| --- | --- |
| void filName(vector<string> crdName, ifstream &names) | Parameters: vector<string> crdName is an empty vector from plyGame(); ifstream &names is a files with string names  Uses names file to fill the crdName vector |
| void filIds(short []crdID, ifstream &ids,const short DK\_SIZE) | Parameters: empty short array sized to DK\_SIZE, ifstream object connected to a file of short ints; const short DK\_SIZE is the size of the array  Uses the referenced file to fill the crdID array with values |
| void filVals(short []crdVal, ifstream &vals,const short DK\_SIZE) | Parameters: empty short array sized to DK\_SIZE, ifstream object connected to a file of short ints; const short DK\_SIZE is the size of the array  Uses the referenced file to fill the crdVal array with values |
| void filSuit(short []crdSuit, ifstream &suits,const short DK\_SIZE) | Parameters: empty short array sized to DK\_SIZE, ifstream object connected to a file of short ints; const short DK\_SIZE is the size of the array  Uses the referenced file to fill the crdSuit array with values |
| void plyGame() | Main function for gameplay |
| char GtDealr() | Uses rand() to generate a character which will decide who gets dealt first—the computer or the Player. ‘C’ for Comp; ‘P’ for Player. The one who gets dealt first also gets “nibs”. This information will be outputted as a message.  Output: A char with ‘P’ for player and ‘C’ for computer |
| void deal(short deck[], vector<short> &handC, vector<short> &handP ,char dealer) | Parameters: deck[],vector<short> &handC, vector<short> &handP, and dealer from plyGame().  Uses and changes the deck array, the computer’s hand vector, and the player’s hand vector using addCard(). Char dealer contains ‘P’ for player or ‘C’ for computer. Either starts dealing to the computer or Player based on the value of dealer. Copies the value to a hand, and then changes it in the deck to -1 to indicate that it was used. Loops for six cards to each player. |
| short addCard(short deck[], vector<short> &) | Parameters: deck[] from plyGame(); vector<short> is an empty vector  Uses and the deck array. Changes the computer’s hand vector, and the player’s hand vector. Uses SlCrdDk() for array sub number to select. Uses inDeck() to determine if card is available.  Output: card ID between 0-51 |
| short SlCrdDk() | Uses rand() to select a number between 0 and 51, that value is returned as a short. The number is used for an array sub number |
| bool inDeck(short cardSub, const short[]) | Parameters: cardSub is a short between 0-51 and the short[] is the deck array from plyGame().  Uses cardSub to determine if the card is available for selection or has already been selected  Output: boolean, true if card is available; otherwise false |
| bool inHand(short id, const vector<short> &) | Parameters: cardSub is a short between 0-51 and the vector is a “hand” with card ID’s  Uses id to search through the vector to determine if the card is available for selection  Output: boolean, true if card is available; otherwise false |
| bool inHndC(short sub, const vector<short> &hand) | Parameters: sub is a short between 0-51 and the vector is the computer’s hand from plyGame().  Uses sub to determine if the card is available for selection or has already been selected. Used for computer’s hand only.  Output: boolean, true if card is available; otherwise false |
| short cardID(string, const vector<string> &crdName, const short crdID[], const short DK\_SIZE) | Parameters: string is a card name; const string crdName[] from plyGame(); const short crdID[] from plyGame(); const short DK\_SIZE for looping through array  First validates that the string name is a string in the array. If yes, it returns the card ID value in the corresponding sub position of crdID[]. If no, returns -1 to indicate that the card name does not exist  Output: short card ID between 0-51 or if invalid card name -1 |
| void dspHand(vector<short> handP , vector<string> crdName) | Parameters: vector handP is the player’s hand containing card ID’s; vector<string> crdName from plyGame() is a vector containing string names for cards.  Uses the values from the handP vector to look up values in the crdName[] array and displays each card ID which is > 0 |
| short strtCrd(short deck[],bool &isStJak) | Parameters: uses deck[] from plyGame() which holds available card values; uses reference to boolean isStJak from plyGame().  Uses SlCrdDk() to get a random value. Validates that the corresponding card is available using inDeck(). Assigns a value of -1 in the deck to indicate that it was used. Checks if the card equals values for Jacks, if so isStJak=True, else false.  Output: Returns a short containing a card ID |
| void dspHand(short, vector<string> crdName) | Parameters: the short is a card ID from 0-51; vector<string> crdName from plyGame() is a vector containing string names for cards.  !OVERLOADED FUNCTION! Uses the short to display the name for the card ID. |
| void rmveCrd(short, const vector<short> &) | Parameters: the short is a card ID 0-51; vector<short> is any vector being used as a hand with card values  Searches through the vector for the card ID and changes it to -1, effectively removing the card from the hand. |
| void discrdP(vector<short> &handP,vector<short> &crib, const vector<string> crdName , const short cardID[]) | Parameters: vector<short> &handP is Player’s hand from plyGame(); vector<short> &crib from plyGame();const vector<string> crdName from plyGame() used for dspHand() and cardID(); const short cardID[] from plyGame() used for cardID()  Loops twice. Displays Player’s hand using dspHand(). Gets a card name from Player, validates and gets card ID from cardID(). Validates that card is in Player’s hand using inHand(). Assigns card ID to vector<short> &crib. Changes card ID to -1 in Player’s hand using rmveCrd(). |
| void discrdC(vector<short> &handC,vector<short> &crib, const cardID[]) | Parameters: vector<short> &handC is computer’s hand from plyGame(); vector<short> &crib is crib hand from plyGame(); const cardID[] is card ID’s from plyGame()  Loops twice. Randomly selects a card to discard to crib hand. Uses rand()%6 to select a value 0-5 to use as a sub number. Validates that sub number has a valid card ID using inHand(). Assigns card ID to vector<short> &crib. Changes card ID to -1 in computer’s hand. |
| void copyVec(vector<short> &, vector<short> &) | Parameters: two vectors of the same datatype  Takes the values in the first vector and copies them to the second vector. |
| short cardVal(short, const short crdVal[]) | Parameters: a short int card ID from 0-51; crdVal[] is the array of card values from plyGame()  Uses the card ID to determine the card’s value  Output: a short with the value 1-10 |
| void plyCrdC(vector<short> &c\_handC, const short crdVal[], const vector<string> &crdName, unsigned short &playCnt, bool &cmpPass, int &pointsC, char &lastPly) | Parameters: vector<short> &c\_handC is a copy of the computer’s hand from plyGame(); const crdVal[] is an array of card values from plyGame() for cardVal(); const vector<string> crdName[] is an vector of card names for dspHand() from plyGame(); unsigned short &playCnt has value 0-31 from plyGame(); bool &cmpPass initialized to false from plyGame(); int &pointsC is points total from plyGame(); char &lastPly holds ‘C’ or ‘P’ from plyGame()  Used to have the computer play a card from its “hand”. Loops through the hand looking for a card ID >=0. Uses cardVal() to get the card’s value. Checks if card’s value plus playCnt > 31. Returns cmpPass=True if loop ends without selecting a card. If cmpPass=False—displays card using dspHand(), assigns -1 to card ID in hand, calculates playCnt, and sets lastPly = ‘C’. Displays playCnt. |
| void plyCrdP(vector<short> &c\_handP, const short crdID[], const short crdVal[], const vector<string> &crdName, unsigned short &playCnt, bool &plrPass, int &pointsP, char &lastPly) | Parameters: vector<short> &c\_handP is a copy of the Player’s hand from plyGame(); const short crdID[] is an array of card id’s for cardID() from plyGame(); const crdVal[] is an array of card values from plyGame() for cardVal();const vector<string> crdName is a vector of card names for dspHand() from plyGame(); unsigned short &playCnt has value 0-31 from plyGame(); bool &plrPass initialized to false from plyGame(); int &pointsP is points total from plyGame(); char &lastPly holds ‘C’ or ‘P’ from plyGame()  Used to have Player play a card from their hand. Displays Player’s hand using dspHand(). Gets a card name from Player, validates and gets card ID from cardID(). Validates that card is in Player’s hand using inHand().Uses cardVal() to get the card’s value. Checks if card’s value plus playCnt > 31. Sets plrPass=True if Player enters “P”. If plrPass=False—assigns -1 to card ID in hand, calculates playCnt, and sets lastPly = ‘C’. Displays playCnt. Displays card using dspHand(). |
| short vldPlay(const vector<short> &hand, const short crdVal[],const short playCnt) | Parameters: vector<short> hand is a hand containing values -1 through 51; const crdVal[] is an array of card values from plyGame(); unsigned short &playCnt has value 0-31 from plyGame()  Determines if there is a playable card in a particular hand given the current playCnt.  Output: The card ID of the first playable card in the hand, else -1 if no playable cards are found |
| bool hasCrds(vector<short>) | Parameters: Uses a short int vector  Checks each value in the vector to see if any one value is >=0 meaning there is a valid card left to play  Output: returns true if there are one or more cards left, else returns false |
| void thePlay((vector<short> &c\_handC,vector<short> &c\_handP,const short crdVal[], const short crdID[],  const vector<string> &crdName, unsigned short &playCnt,  bool &cmpPass, bool &plrPass, int &pointsC, int &pointsP, char &dealer, char &lastPly, char option) | Parameters: vector<short> &c\_handP is a copy of the Player’s hand from plyGame();vector<short> &c\_handC is a copy of the computer’s hand from plyGame(); const short crdID[] is an array of card id’s for cardID() from plyGame(); const crdVal[] is an array of card values from plyGame() for cardVal();const vector<string> crdName is a vector of card names for dspHand() from plyGame(); unsigned short &playCnt has value 0-31 from plyGame(); bool &plrPass initialized to false from plyGame();int &pointsC is computer’s points total from plyGame(); int &pointsP is player’s points total from plyGame(); char &lastPly holds ‘C’ or ‘P’ from plyGame(); option controls which variable controls the execution order  thePlay() executes plyCrdC() and plyCrdP() using looping constructs. The order of execution is based on either char dealer or char lastPly using char option to determine which variable will be used. Checks if either the player or computer has “passed” and keeps track of the playCnt. Adds “GO” scoring functionality based on lastPly. |

## 

## Version 2 Additions

|  |  |
| --- | --- |
| void theShow(const vector<short> &vecHndP,const vector<short> &vecHndC,  const vector<short> &vecCrib,const short starter, const char dealer, int &pointsP, int &pointsC) | Parameters: const vector<short> &vecHndP is the player’s hand vector from plyGame; const vector<short> &vecHndC is the computer’s hand vector from plyGame();const vector<short> &vecCrib is the crib vector from plyGame(); const short starter is the up card from plyGame; const char dealer from plyGame; int &pointsP is player’s points from plyGame; int &pointsC is computer’s points from plyGame  Takes the various hands, dealer, up card, and points from plyGame() and uses them to generate the scores for each hand. It changes the points totals of both computer and player to reflect the hand scores. |
| void scrHand(const short combi2[10][2],const short combi3[10][3],const short combi4[5][4], const Card lngHnd[],const Card shrtHnd[], const Card cdUpCrd,int & points, const int size) | Parameters: const short combi2[10][2] 2D array containing card combinations from theShow(); const short combi3[10][3] 2D array containing card combinations from theShow(); const short combi4[5][4] 2D array containing card combinations from theShow(); const Card lngHnd[] is an array of card structures from theShow() with the up card included; const Card shrtHnd[]is an array of card structures from theShow() without the up card; const Card cdUpCrd is the Up card from theShow(); int & points from theShow(); const int size is a defaulted parameter set to 5  scrHand() is a function acting as a container for a set of function calls for scoring a single hand. It calls dsCdHnd() to display hand, scrPrs() to score pairs, scr15s() to score 15s, scrFlsh() to score flushes, scrNbs to score nibs, and scrRns to score runs. |
| void dsCdHnd(const Card hand[],const int size) | Parameters: const Card hand[] is an array of card structures; const int size is the size of the array  Iterates through the card structure array displaying the name member of each card |
| void scrRns(const short combi3[10][3],const short combi4[5][4],const Card hand[],int &points) | Parameters: const short combi3[10][3] 2D array containing card combinations from theShow(); const short combi4[5][4] 2D array containing card combinations from theShow(); const Card hand[] is an array of card structures from theShow() with the up card included; int &points are from theShow()  Looks for runs (i.e. 1,2,3,4,5) in the hand starting with the largest runs and then looking for the smaller if larger runs are not found. Uses the card combination arrays to search all possible card combinations for runs. Adds points when any run is found. |
| void cnvCard(const Card crdDeck[],Card &card,const short id) | Parameters: const Card crdDeck[] is an array of card structure with information on all cards from theShow(); Card &card is a card structure; const short id is a short with a value between 0-51  cnvCard()takes a card Id and fills a card structure element with the proper values for that specific ID. |
| void scrNbs(const Card shrtHnd[],const Card cdUpCrd,int &points,const int size4) | Parameters: const Card shrtHnd[] is an array of card structures without the Up Card from theShow(); const Card cdUpCrd is the Up Card from theShow(); int &points is from theShow(); const int size4 is a defaulted parameter = 4  scrNbs() first checks if the Up Card is a Jack. If so, it adds points for each jack in the hand. |
| void scrFlsh(const Card hand[],const Card shrtHnd[],int &points, const int size5) | Parameters: const Card hand[]is an array of card structures from theShow() with the up card included; const Card shrtHnd[]is an array of card structures without the Up Card from theShow(); int &points from theShow(); const int size5 is a defaulted parameter = 5  scrFlsh() first checks if all five cards have the same suit (a flush) and if so adds points. If only four cards are the same suit it checks if the hand without the up card has a flush of all four cards. If so, it adds points. |
| void scr15s(const short combi2[10][2],const short combi3[10][3],const short combi4[5][4], const Card hand[],int &points) | Parameters: const short combi2[10][2] 2D array containing card combinations from theShow(); const short combi3[10][3] 2D array containing card combinations from theShow(); const short combi4[5][4] 2D array containing card combinations from theShow(); const Card hand[] is an array of card structures from theShow() with the up card included; int &points is from theShow()  scr15s() uses the three 2D card combination arrays to check for all possible unique combinations of cards whose values add up to 15. Points are added for each combination making 15. |
| void scrPrs(const short combi[10][2],const Card hand[],int &points) | Parameters: const short combi[10][2] 2D array containing card combinations from theShow(); const Card hand[] is an array of card structures from theShow() with the up card included; int &points is from theShow()  scrPrs() uses the card combination array to iterate through all possible pair combinations searching through the hand for a pair of cards with the same rank. |
| void cnvSHnd(const Card crdDeck[],Card crdSHnd[],short arHnd[],const vector<short> &vcHnd,const int size4) | Parameters: const Card crdDeck[] is an array of card structure with information on all cards from theShow(); Card crdSHnd[]is an array of card structures without the Up Card from theShow(); short arHnd[] is an array of card IDs from theShow(); const vector<short> &vcHnd is a vector of card IDs that can be traced to plyGame(); const int size4 is a defaulted parameter = 4  cnvSHnd()creates a "Small" 4 card hand without the up card from the vcHnd vector using cpyVcAr() to convert the hand to an array, cpyArry to copy the array for sorting, selSort() to sort the hand, and then crdDeck[] for card member values. |
| void cnvHand(const Card crdDeck[],Card crdHand[],short arHnd[], const vector<short> &vcHnd,const short upCard,const int size5) | Parameters: const Card crdDeck[] is an array of card structure with information on all cards from theShow(); Card crdHand[]is an array of card structures with the Up Card from theShow(); short arHnd[] is an array of card IDs from theShow(); const vector<short> &vcHnd is a vector of card IDs that can be traced to plyGame();const Card cdUpCrd is the Up Card from theShow(); const int size5 is a defaulted parameter = 5  cnvSHnd()creates a five card hand with the up card added from the vcHnd vector using cpyVcAr() to convert the hand to an array, addUpCd() to append the Up Card ID to the array, cpyArry to copy the array for sorting, selSort() to sort the hand, and then crdDeck[] for card member values. |
| void cpyArry(short a[],short b[],const int size) | Parameters: Two arrays of short ints and the size of the arrays  Simply copies the contents of short a[] to short b[] up to the size of the arrays. |
| void selSort(short a[], const int size) | Parameters: a[] is an array of short integers; const int size is the size of the array  selSort() is a selection sort function that accepts an array of shorts and sorts the array in ascending order. |
| void addUpCd(short hand[],short upCard,const int size){ | Parameters: short hand[] is an array of short ints holding card IDs; short upCard is the card ID of the Up Card; const int size is the size of the array  addUpCd() appends the card ID of the Up Card to the end of the array of card IDs. |
| void cpyVcAr(const vector<short> &vcHnd, short aHnd[],const int size) | Parameters: const vector<short> &vcHnd is a vector of card IDs as short ints; short aHnd[] is an array of short ints; const int size is the size of the array  cpyVcAr() copies the contents of the vector into the array up to the size of the array. |
| void banner(char fName[]) | Parameters: char fName[] is a character array holding a filename for an input stream object  banner() uses the fName[] to open a file and read in and output seven lines (or rows) of characters |
| void flCrdDk(Card crdDeck[],const int size,char fnIds[],char fnRanks[],char fnSuits[],char fnNames[],char fnVals[]) | Parameters: Card crdDeck[] is an array of card structure elements; const int size is the size of the array; char fnIds[] is a character array with the filename for the card IDs; char fnRanks[]is a character array with the filename for the card ranks; char fnSuits[]is a character array with the filename for the card suits; char fnNames[]is a character array with the filename for the card names; char fnVals[]is a character array with the filename for the card numerical values  flCrdDk() opens file streams and simultaneously fills all the members for each card structure in the array up to the size of the array |

# Major Variables and Objects

## Version 1

|  |  |  |  |
| --- | --- | --- | --- |
| **Major Variables and Objects** | | | |
| **Function Location** | **Type** | **Name** | **Description** |
| plyGame() | short array | crdID | An array to hold all of the card id’s 0-51; card ID == sub number |
| plyGame() | string vector | crdName | An array to hold all of the card names |
| plyGame() | short array | crdSuit | An array to hold the suit for each card |
| plyGame() | short array | crdVal | An array to hold the numerical value for each card |
| plyGame() | ifstream | names | Input stream object to stream card names from file |
| plyGame() | ifstream | suits | Input stream object to stream card suits from file |
| plyGame() | ifstream | vals | Input stream object to stream numerical card values from file |
| plyGame() | ifstream | ids | Input stream object to stream card ID’s from file |
| plyGame() | character | dealer | ‘C’ for computer, ‘P’ for Player. For determining who is dealer |
| plyGame() | short array | deck | Copy of crdID[] used to hold available card values for the “deck” |
| plyGame() | short vector | handC | Used to hold card values for the computer’s hand; MAX size == 6 |
| plyGame() | short vector | handP | Used to hold card values for the Player’s hand; MAX size == 6 |
| plyGame() | short vector | c\_handC | Copy of handC used to hold card values for the computer’s hand |
| plyGame() | short vector | c\_handP | Copy of handP used to hold card values for the Player’s hand |
| plyGame() | short vector | crib | Used to hold card values for the crib; MAX size == 4 |
| plyGame() | const short | DK\_SIZE | Holds the size of the Deck arrays == 52 |
| plyGame() | integer | pointsC | Holds the computer’s point total; |
| plyGame() | integer | pointsP | Holds the Player’s point total; |
| plyGame() | short | starter | Holds the starting card’s ID number |
| plyGame() | bool | isStJak | Holds whether the starting card is a Jack. True if a Jack. |
| plyGame() | const float | NIB\_MUL | The nibs multiplier == 1.2; Used as a bonus |
| plyGame() | unsigned short | playCnt | Holds the count for The Play; values 0-31 |
| plyGame() | bool | cmpPass | Initialized to false; indicates if computer has “passed” during play |
| plyGame() | bool | plrPass | Initialized to false; indicates if Player has “passed” during play |
| plyGame() | character | lastPly | ‘C’ for computer, ‘P’ for Player. For determining who played last |

## Version 2 Additions

|  |  |  |  |
| --- | --- | --- | --- |
| **Major Variables and Objects** | | | |
| **Function Location** | **Type** | **Name** | **Description** |
| plyGame() | char array[25] | fName | An array of size 25 to hold the characters of a filename. |
| plyGame() | ifstream | fInput | An input stream object to read in data from a file |
| plyGame() | string | lstGame | Holds the results of the last played game |
| plyGame() | ofstream | fOutput | An output stream object to write data to a file |
| theShow() | const int | DK\_SIZE | The size of a deck |
| theShow() | const int | HND\_SZ5 | The size of a hand |
| theShow() | const int | HND\_SZ4 | The size of a crib and a hand w/o the up card |
| theShow() | const int | COLS\_2 | Constant to hold the column size for 2D array |
| theShow() | const int | COLS\_3 | Constant to hold the column size for 2D array |
| theShow() | const int | COLS\_4 | Constant to hold the column size for 2D array |
| theShow() | const int | ROWS\_10 | Constant to hold the row size for 2D array |
| theShow() | const int | ROWS\_5 | Constant to hold the row size for 2D array |
| theShow() | int | pointsK | Holds the points total for the crib hand |
| theShow() | Card array | handP | Player’s hand with up card |
| theShow() | Card array | handC | Computer’s hand with up card |
| theShow() | Card array | crib | Crib hand with up card |
| theShow() | Card array | sHndP | Player’s hand without up card |
| theShow() | Card array | sHndC | Computer’s hand without up card |
| theShow() | Card array | sCrb | Crib hand without up card |
| theShow() | short array | arHandP | Holds the sorted card ids for player's hand |
|  |  |  |  |
| theShow() | short array | arHandC | Holds the sorted card ids for computer's hand |
| theShow() | short array | arCrib | Holds the sorted card ids for crib hand |
| theShow() | short array | arSHndP | Holds the sorted card ids for player's hand w/o Up Card |
| theShow() | short array | arSHndC | Holds the sorted card ids for computer's hand w/o Up Card |
| theShow() | short array | arSCrib | Holds the sorted card ids for crib hand w/o Up Card |
| theShow() | char array | fnIds | Holds filename for card IDs |
| theShow() | char array | fnRanks | Holds filename for card ranks |
| theShow() | char array | fnSuits | Holds filename for card suits |
| theShow() | char array | fnNames | Holds filename for card names |
| theShow() | char array | fnVals | Holds filename for card vals |

# Concepts Used

|  |  |  |
| --- | --- | --- |
| **Concepts Used-References Gaddis Chapters** | | |
| **Chapter** | **Concept** | **Line # of first instance** |
| Chapter 2 | cout object  #include  variables  integer data type  character data type  c++ string class  floating point data type  bool data type  arithmetic operators  comments  named constants | 114  13  100  100  112  163  134  183  301  1  134 |
| Chapter 3 | cin object  type conversion  type casting  mathematical expressions | 124  157  157  301 |
| Chapter 4 | Relational operators  if statement  if/else  nested if statements  if/else if  logical operators  menus  validating user input  comparing characters and strings  the conditional operator  switch statement | 107  262  275  320  247  216  102  497  275  165  104 |
| Chapter 5 | while loop  do-while loop  for loop  nested loops  breaking a loop | 213  107  259  472  282 |
| Chapter 6 | defining functions  calling functions  function prototypes  sending data into a function  passing data by value  return statement  reference variables  overloading functions  default arguments | 99  103  34  106  35  125  34  46  76 |
| Chapter 7 | array initialization  parallel arrays  processing array contents  arrays as function arguments  STL vector  multidimensional arrays | 136  136,137,138  454  150  139  692 |
| Chapter 8 | searching arrays  sorting arrays  searching vectors | 882  1031  259 |
| Chapter 10 | C-strings  library functions for C-strings (strcpy()) | 112  226 |
| Chapter 11 | Abstract Data Types  accessing structure members  arrays of structures  structures as function arguments | 24  801  712  732 |
| Chapter 12 | ifstream object- reading from a file  passing file stream objects to functions  ofstream object-writing to a file | 161  150  241 |

# Program Listing

/\*

\* File: main.cpp

\* Author: Drew Allensworth

\*

\* Created on February 8, 2014, 4:05 PM

\* Purpose: Program for CSC-5 Project 2

\* Drewbage cribbage game. It's two-hand cribbage,

\* 1 player against the computer.

\*

\*/

//System Libraries

#include <cstdlib>

#include <iostream>

#include <string>

#include <cstring>

#include <vector>

#include <cmath>//for rand()

#include <ctime>//for srand()

#include <fstream>//for reading/writing files

using namespace std;

//Structure Declarations not in User Defined Libraries

struct Card{//Declare Card Structure

short id;//Card id number

short rank;//Rank of card. From ace to king using numbers

short suit;//Suit of card. From clubs to spades using numbers

string name;//Name of card. Ex: a\_s (ace of spades)

short val;//Count value of card. 1 to 10

};

//Global Constants

//Function Prototypes

void filName(vector<string> &,ifstream &);

void filIds(short[], ifstream &, short);

void filVals(short[], ifstream &, short);

void filSuit(short[], ifstream &, short);

char gtDealr();

short slCrdDk();

bool inDeck(short, short[]);

bool inHand(short, const vector<short> &);

bool inHndC(short, const vector<short> &);

void deal(short [], vector<short> &, vector<short> &,char);

short addCard(short [], vector<short> &);

void dspHand(vector<short>,const vector<string> &);

void dspHand(short, const vector<string> &);//overloaded function

short strtCrd(short [],bool &);

short cardID(string, const vector<string> &, const short []);

void discrdP(vector<short> &, vector<short> &,

const vector<string> &, const short []);

void discrdC(vector<short> &,vector<short> &, const short []);

void rmveCrd(short, vector<short> &);

void copyVec(const vector<short> &, vector<short> &);

short cardVal(short n, const short []);

short vldPlay(const vector<short> &, const short [],

const short);

void plyCrdP(vector<short> &, const short [],

const short [], const vector<string> &,

unsigned short &, bool &, int &, char &);

void plyCrdC(vector<short> &, const short [], const vector<string> &,

unsigned short &, bool &, int &, char &);

void thePlay(vector<short> &,vector<short> &,

const short [], const short [],

const vector<string> &, unsigned short &,

bool &, bool &, int &, int &, char &, char &, char);

bool hasCrds(vector<short> &);

void Menu();

int getN();

void def(int);

void plyGame();

//Function Prototypes-Version 2 additions

void banner(char []);

void flCrdDk(Card[],int,char[],char[],char[],char[],char[]);

void cpyVec2(const vector<short> &, vector<short> &);

void cpyVcAr(const vector<short> &,short[],int size=4);//hands have four cards

void cpyArry(short[],short[],int);

void addUpCd(short[],short,int);

void selSort(short[],int);

void cnvHand(const Card[],Card[],short[],const vector<short> &,const short,

const int size5=5);

void cnvSHnd(const Card[],Card[],short[],const vector<short> &,

const int size4=4);

void cnvCard(const Card[],Card &,const short);

void scrPrs(const short [10][2],const Card [],int &);

void scr15s(const short [10][2],const short [10][3],const short [5][4],

const Card[],int &);

void scrRns(const short [10][3],const short [5][4],const Card[],int &);

void scrFlsh(const Card[],const Card[],int &,const int size5=5);

void scrNbs(const Card[],const Card,int &,const int size4=4);

void dsCdHnd(const Card[],const int);

void scrHand(const short [10][2],const short [10][3],const short [5][4],

const Card[],const Card[],const Card,int &,const int size=5);

void theShow(const vector<short> &,const vector<short> &,const vector<short> &,

const short, const char, int &, int &);

//Execution Begins Here!!!

int main(int argv,char \*argc[]){

int inN;

do{

Menu();

inN=getN();

switch(inN){

case 1: plyGame();break;

default: def(inN);}//More options go here!!!

}while(inN==1);//<- don't forget to change this!!!

return 0;

}

void Menu(){

char fName[25]="./Title.dat";

banner(fName);

cout<<endl;

cout<<"Menu for Drewbage Cribbage"<<endl;

cout<<"Type 1 to Play Game"<<endl;

// cout<<"Type 2 for Help"<<endl;

// cout<<"Type 3 for High Scores"<<endl;

cout<<"Type anything else to exit \n"<<endl;

}

int getN(){

int inN;

cin>>inN;

return inN;

}

void def(int inN){

cout<<"You typed "<<inN<<" to exit the program"<<endl;

}

void plyGame(){

//Declare Variables

const float NIB\_MUL=1.2;//Multiplier for nibs bonus

const short DK\_SIZE=52;//Holds the size of the Deck arrays == 56

short crdID[DK\_SIZE];//All of the card id’s 0-51; card ID == sub number

short crdVal[DK\_SIZE];//An array to hold the numerical value for each card

short crdSuit[DK\_SIZE];//An array to hold the suit for each card

vector<string> crdName;//An array to hold all of the card names

short deck[DK\_SIZE];//An array to hold the "deck" cards for gameplay

vector<short> handC;//Vector for computer's hand

vector<short> handP;//Vector for player's hand

vector<short> crib;//Vector for the crib

//file stream objects to fill arrays and vector

ifstream names;

ifstream ids;

ifstream vals;

ifstream suits;

//fill the arrays and vectors

filName(crdName,names);

filIds(crdID,ids,DK\_SIZE);

filVals(crdVal,vals,DK\_SIZE);

filSuit(crdSuit,suits,DK\_SIZE);

//fills the deck

filIds(deck,ids,DK\_SIZE);

//seeds the random number generator

srand(static\_cast<unsigned int>(time(0)));

//Display Results of last game

cout<<"----------------------------"<<endl;

cout<<"Results of Most Recent Game:"<<endl;

ifstream fInput;

fInput.open("./lastGame.dat");

string lstGame;

getline(fInput,lstGame);

(lstGame.length()>0)? cout<<lstGame<<endl:

cout<<"Game has not been played before"<<endl;

fInput.close();

cout<<"----------------------------"<<endl;

//Game Header

char fName[25]="./thePlay.dat";

banner(fName);

cout<<endl;

//choose the dealer

char dealer=gtDealr();

dealer=='P'?cout<<"The dealer is Player":

cout<<"The dealer is the computer";

cout<<endl;

//deal the cards

deal(deck,handC,handP,dealer);

//select starting card

bool isStJk=false;//initialized to false, true if starting card is a Jack

short starter;//variable containing the starting card ID

starter=strtCrd(deck,isStJk);//sets both starter and isStJk

cout<<endl;

//display the player's hand

cout<<"Your (Player's) hand contains: ";

dspHand(handP,crdName);

cout<<endl;

//Begins the discard process

discrdP(handP,crib,crdName,crdID);//discard for Player

discrdC(handC,crib,crdID);//discard for Computer

cout<<"The computer has also discarded two cards into the crib."

<<endl<<endl;

//display the starter card

cout<<"The starter card is the: ";

dspHand(starter,crdName);

cout<<endl<<endl;

//Make copies of the hands to use for The Play

vector<short> c\_handC;//Vector for copy of computer's hand

copyVec(handC,c\_handC);//Fully copy handC to c\_handC

vector<short> c\_handP;//Vector for copy of player's hand

copyVec(handP,c\_handP);//Fully copy handP to c\_handP

//Variables for The Play

unsigned short playCnt=0;//holds count for The Play. Values 0-31

bool plrPass=false,//Indicates if Player has “passed” during play

cmpPass=false;//Indicates if computer has “passed” during play

char lastPly;//‘C’ = computer, ‘P’ = Player. For determining who played last

int pointsP=0,//holds total point for the player

pointsC=0;//holds total points for the computer

//'D' means run The Play using lastPly rather than dealer

thePlay(c\_handC,c\_handP,crdVal,crdID,crdName,playCnt,

cmpPass,plrPass,pointsC,pointsP,dealer,lastPly,'D');

//Continue game if cards remain after the first

while(hasCrds(c\_handP)||hasCrds(c\_handC)){

//reset variables for new game

plrPass=false;

cmpPass=false;

playCnt=0;

cout<<endl<<"New Round Started: Play Count Reset to 0"<<endl;

//'P' means run The Play using lastPly rather than dealer

thePlay(c\_handC,c\_handP,crdVal,crdID,crdName,playCnt,

cmpPass,plrPass,pointsC,pointsP,dealer,lastPly,'P');

}

//Header for The Play results

cout<<"---------------------------------------------\n"

<<"The total points are: Player: "<<pointsP<<"\tComputer: "

<<pointsC<<endl;

//Header for The Show

strcpy(fName,"./theShow.dat");

banner(fName);

cout<<endl;

//Clean the hands of negative values for theShow

vector<short> clnHndC;//Vector for copy of computer's hand

cpyVec2(handC,clnHndC);//Fully copy handC to clnHandC

vector<short> clnHndP;//Vector for copy of player's hand

cpyVec2(handP,clnHndP);//Fully copy handP to clnHandP

//Start The Show--score the hands

theShow(clnHndP,clnHndC,crib,starter,dealer,pointsP,pointsC);

//Header for final score results

cout<<"---------------------------------------------\n"

<<"The total points are: Player: "<<pointsP<<"\tComputer: "

<<pointsC<<endl;

//determine who wins and output results to a file

ofstream fOutput;

fOutput.open("./lastGame.dat");

if(pointsP>pointsC){

cout<<"Player WINS!!!";

fOutput<<"Player won by "<<(pointsP-pointsC)<<" points!"<<endl;

}else if(pointsC>pointsP){

cout<<"Computer WINS!!!";

fOutput<<"Computer won by "<<(pointsC-pointsP)<<" points!"<<endl;

}else{

cout<<"We Have a TIE!!!";

fOutput<<"The Computer and Player tied!"<<endl;

}

fOutput.close();

cout<<endl<<endl;

}

bool hasCrds(vector<short> &hand){

for(int i=0;i<hand.size();i++){

//if any sub has a valid card ID return true

if(hand[i]>=0)return true;

}

return false;//else return false

}

void thePlay(vector<short> &c\_handC,vector<short> &c\_handP,

const short crdVal[], const short crdID[],

const vector<string> &crdName, unsigned short &playCnt,

bool &cmpPass, bool &plrPass,

int &pointsC, int &pointsP, char &dealer,

char &lastPly, char option){

char dlFrst;//allows one to set the program to run based on either

//either dealer or lastPly depending on the option

if(option=='D'){

dlFrst=dealer;

}else dlFrst=lastPly;

//Start The Play

if(dlFrst=='P'){

do{

plyCrdC(c\_handC,crdVal,crdName,playCnt,cmpPass,

pointsC,lastPly);

if(playCnt==31)break;//get out of the loop if 31

if(plrPass==true&&cmpPass==true)break;//if both players pass

plyCrdP(c\_handP,crdID,crdVal,crdName,playCnt,plrPass,

pointsP,lastPly);

}while(playCnt<31&&(plrPass==false||cmpPass==false));

}else{//dealer=='C'

do{

plyCrdP(c\_handP,crdID,crdVal,crdName,playCnt,plrPass,

pointsP,lastPly);

if(playCnt==31)break;

if(plrPass==true&&cmpPass==true)break;

plyCrdC(c\_handC,crdVal,crdName,playCnt,cmpPass,

pointsC,lastPly);

}while(playCnt<31&&(plrPass==false||cmpPass==false));

}

if(plrPass==true&&cmpPass==true){

playCnt=0;//reset the game count

if(lastPly=='P'){

pointsP+=1000;

cout<<"Last play by Player. +1000 points for the \"GO\"!"<<endl;

}else{//lastPly=='C'

pointsC+=1000;

cout<<"Last play by the computer. +1000 points for the \"GO\"!"

<<endl;

}

}

}

short vldPlay(const vector<short> &hand, const short crdVal[],

const short playCnt){

short val,id;

for(int i=0;i<hand.size();i++){

if(hand[i]>=0){

id=hand[i];

val=cardVal(id,crdVal);

if(playCnt+val<=31){

return hand[i];//return value of first card that's a valid play

}

}

}

return -1;//if no card in the hand is a valid play, return -1

}

void plyCrdC(vector<short> &c\_handC, const short crdVal[],

const vector<string> &name, unsigned short &playCnt,

bool &cmpPass, int &pointsC, char &lastPly){

if(hasCrds(c\_handC)){//checks if hand has cards

//gets the first valid playable card and return the ID

//If there is no playable card in the deck, returns -1

short card=vldPlay(c\_handC,crdVal,playCnt);

//perform operations if there is a playable card

if(card>=0){

short val=cardVal(card,crdVal);//gets the value of the card

playCnt+=val;//adds the value to the play count

//Show the Player the card the computer chose

cout<<"The computer plays the: ";

dspHand(card,name);

cout<<endl<<"The play count is: "<<playCnt<<endl;

if(playCnt==15){//assigns points if count == 15||31

pointsC+=2000;

cout<<"The computer scores 2000 points! It's total points are: "

<<pointsC<<endl;

}else if(playCnt==31){

pointsC+=4000;

cout<<"The computer scores 4000 points! It's total points are: "

<<pointsC<<endl;

}else;//assigns no points

lastPly='C';//sets the computer as the most recent player

rmveCrd(card,c\_handC);//removes the card from the hand

}else{

cmpPass=true;

cout<<"The computer has chosen to pass!"<<endl;

}

}else{//if hand does not have any cards

cmpPass=true;

cout<<"The computer is out if cards and must pass!"<<endl;

}

}

void plyCrdP(vector<short> &c\_handP, const short id[],

const short crdVal[], const vector<string> &name,

unsigned short &playCnt, bool &plrPass,

int &pointsP, char &lastPly){

if(hasCrds(c\_handP)){

cout<<"Your (Player's) hand contains: ";

dspHand(c\_handP,name);//displays the Player's hand for review

//gets the first valid playable card and return the ID

//If there is no playable card in the deck, returns -1

short card=vldPlay(c\_handP,crdVal,playCnt);

//perform operations if there is a playable card

if(card>=0){

cout<<"Please type the name (as displayed) of the card\n"

<<"in your hand you wish to use: "<<endl;

string crdName;//to hold the user's input

short crdID;//to hold the looked up cardID

short val;//to hold the card value

bool avail;//holds whether the card is found in the Player's hand

char pass='A';//holds 'P' to pass if user cannot play a card

do{

cout<<"Card: ";

cin>>crdName;

crdID=cardID(crdName,name,id);//checks if the card name has a card ID

while(crdID<0){//while card name is invalid

cout<<"That is not a valid card name\n"

<<"please enter a card name from your hand: ";

cout<<"Card: ";

cin>>crdName;

crdID=cardID(crdName,name,id);

}

avail=inHand(crdID,c\_handP);//checks if card ID is in plyrs hand

while(!avail||crdID<0){//while card ID is not in player's hand

cout<<"You do not have that card in your hand\n"

<<"please enter a card name from your hand: ";

cout<<"Card: ";

cin>>crdName;

crdID=cardID(crdName,name,id);

if(cardID>=0){

avail=inHand(crdID,c\_handP);

}

}

val=cardVal(crdID,crdVal);

if((playCnt+val)>31){

//checks hand for a playable card

short card=vldPlay(c\_handP,crdVal,playCnt);

cout<<"Your card selection added to the current points "

<<"total\n is greater than 31."<<endl;

if(card>=0){//if there is a playable card in the hand

cout<<"You have at least one playable card in your hand.\n"

<<"Please select another card."<<endl;

}else{

pass='P';//have the program pass for the player

//if no playable cards are in their hand

cout<<"You have no playable card. You must pass!!!"

<<endl;

plrPass=true;

}

}

}while(pass!='P'&&(playCnt+val)>31);

if(pass!='P'&&avail&&(playCnt+val)<=31){//defensive programming

playCnt = playCnt + val;//add card value to playCnt

rmveCrd(crdID,c\_handP);//remove card from Player's hand

lastPly='P';//Player is now the most recent one to play

if(playCnt==15){//assigns points if count == 15||31

pointsP+=2000;

cout<<"You score 2000 points! Your total point are: "

<<pointsP<<endl;

}else if(playCnt==31){

pointsP+=4000;

cout<<"You score 4000 points! Your total point are: "

<<pointsP<<endl;

}else;//assigns no points

}

// else cout<<"You have no playable card. You must pass!!!"<<endl;

// cout<<"Your (Player's) hand now contains: ";

// dspHand(c\_handP,name);//displays the Player's hand for review

cout<<"The play count is: "<<playCnt<<endl;

}else{

plrPass=true;

cout<<"You have no playable card. You must pass!"<<endl;

}

}else{//if hand does not have any cards

plrPass=true;

cout<<"You have no playable card. You must pass!"<<endl;

}

}

short cardVal(short n, const short vals[]){

return vals[n];

}

void cpyVec2(const vector<short> &a, vector<short> &b){

for(int i=0;i<a.size();i++){

if(a[i]>=0)b.push\_back(a[i]);//only copy non-negative values

}

}

void copyVec(const vector<short> &a, vector<short> &b){

for(int i=0;i<a.size();i++){

b.push\_back(a[i]);

}

}

void discrdC(vector<short> &handC,vector<short> &crib, const short id[]){

short sub, card;

bool avail;

for(int i=0;i<2;i++){

do{

sub=rand()%6;//assigns a random number for vector sub from 0-5

avail=inHndC(sub,handC);//true if card is available

if(avail){

card=handC[sub];//assigns value in handC at sub number to card

crib.push\_back(card);//puts that card ID into the crib

handC[sub]=-1;//removes the ID from the computer's hand

}

}while(!avail);//loop again if card is not in deck

}

}

void discrdP(vector<short> &handP, vector<short> &crib,

const vector<string> &name, const short id[]){

cout<<"Please type the name (as displayed) of the TWO cards\n"

<<"you wish to discard into the crib"<<endl;

string crdName;//to hold the user's input

short crdID;//to hold the looked up cardID

bool avail;//holds whether the card is found in the Player's hand

for(int i=0;i<2;i++){

cout<<"Card "<<i+1<<": ";

cin>>crdName;

crdID=cardID(crdName,name,id);//checks if the card name has a card ID

while(crdID<0){//while card name is invalid

cout<<"That is not a valid card name\n"

<<"please enter a card name from your hand: ";

cout<<"Card "<<i+1<<": ";

cin>>crdName;

crdID=cardID(crdName,name,id);

}

avail=inHand(crdID,handP);//checks if card ID is in players hand

while(!avail||crdID<0){//while card ID is not in player's hand

cout<<"You do not have that card in your hand\n"

<<"please enter a card name from your hand: ";

cout<<"Card "<<i+1<<": ";

cin>>crdName;

crdID=cardID(crdName,name,id);

if(cardID>=0){

avail=inHand(crdID,handP);

}

}

if(avail){//defensive programming

crib.push\_back(crdID);//push card into crib vector

rmveCrd(crdID,handP);//remove card from Player's hand

}else cout<<"ERROR: From dscrdP. This message should not be seen"

<<endl;

cout<<"Your (Player's) hand now contains: ";

dspHand(handP,name);//displays the Player's hand for review

}

}

void rmveCrd(short id, vector<short> &hand){

for(int i=0;i<hand.size();i++){

if(hand[i]==id){

hand[i]=-1;

break;

}

}

}

short cardID(string str, const vector<string> &name, const short id[]){

for(int i=0;i<name.size();i++){

if(name[i]==str){

return id[i];

}

}

return -1;

}

short strtCrd(short deck[],bool &isStJak){

short card;

bool avail;

do{

card=slCrdDk();//assigns a random number from 0-51

avail=inDeck(card,deck);//true if card is available

if(avail){

if(card>=40&&card<=43){//if card ID is equal to any suit Jack

isStJak=true;

}

deck[card]=-1;//remove card from deck

return card;

}

}while(!avail);//loop again if card is not in deck

}

void dspHand(short n, const vector<string> &name){

cout<<name[n];

}

void dspHand(vector<short> hand, const vector<string> &crdName){

for(int i=0;i<hand.size();i++){

//if the card ID for a given hand sub number is valid

//then output the corresponding name for that ID number

if(hand[i]>=0)cout<<crdName[hand[i]]<<" ";

}

cout<<endl;

}

void deal(short deck[], vector<short> &handC, vector<short> &handP,

char dealer){

short card;

if(dealer=='P'){//deal Player first

for(int i=0;i<6;i++){//assign six cards

card=addCard(deck,handP);//for Player

deck[card]=-1;//remove card from deck

card=addCard(deck,handC);//for computer

deck[card]=-1;//remove card from deck

}

}else{//dealer=='C'//deal computer first

for(int i=0;i<6;i++){//assign six cards

card=addCard(deck,handC);//for computer

deck[card]=-1;//remove card from deck

card=addCard(deck,handP);//for Player

deck[card]=-1;//remove card from deck

}

}

}

short addCard(short deck[], vector<short> &hand){

short card;

bool avail;

do{

card=slCrdDk();//assigns a random number from 0-51

avail=inDeck(card,deck);//true if card is available

if(avail){

hand.push\_back(card);

}

}while(!avail);//loop again if card is not in deck

return card;

}

bool inHand(short id, const vector<short> &hand){

for(int i=0;i<hand.size();i++){

if(hand[i]==id){

return true;

}

}

return false;

}

bool inHndC(short sub, const vector<short> &hand){

if (hand[sub]<0){

return false;

}else return true;

}

bool inDeck(short sub, short deck[]){

if (deck[sub]<0){

return false;

}else return true;

}

short slCrdDk(){

return rand()%52;

}

char gtDealr(){

short n=rand()%2;

if(n==0){

return 'C';

}else return 'P';

}

void filName(vector<string> &crdName,ifstream &names){

names.open("names.txt", ios::in);

string tmp;

int cnt=0;

while(cnt<52){

names>>tmp;

crdName.push\_back(tmp);

cnt++;

}

names.close();

}

void filIds(short array[], ifstream &ids, const short MAX){

ids.open("ids.txt", ios::in);

short tmp;

for(int cnt=0;cnt<MAX;cnt++){

ids>>tmp;

array[cnt]=tmp;

}

ids.close();

}

void filVals(short array[], ifstream &vals, const short MAX){

vals.open("vals.txt", ios::in);

short tmp;

for(int cnt=0;cnt<MAX;cnt++){

vals>>tmp;

array[cnt]=tmp;

}

vals.close();

}

void filSuit(short array[], ifstream &suits, const short MAX){

suits.open("suits.txt", ios::in);

short tmp;

for(int cnt=0;cnt<MAX;cnt++){

suits>>tmp;

array[cnt]=tmp;

}

suits.close();

}

//Start of Version 2 Functions

void theShow(const vector<short> &vecHndP,const vector<short> &vecHndC,

const vector<short> &vecCrib,const short starter,

const char dealer, int &pointsP, int &pointsC){

//Declare constant variables

const int DK\_SIZE=52;//The size of the deck

const int HND\_SZ5=5;//The size of a hand

const int HND\_SZ4=4;//The size if the crib hand

const int COLS\_2=2;//Constant to hold the column size for 2D array

const int COLS\_3=3;//Constant to hold the column size for 2D array

const int COLS\_4=4;//Constant to hold the column size for 2D array

const int ROWS\_10=10;//Constant to hold the row size for 2D array

const int ROWS\_5=5;//Constant to hold the row size for 2D array

//Declare constant arrays

const short sets\_2[ROWS\_10][COLS\_2]={{0,1},{0,2},//2D array to hold all

{0,3},{0,4},//possible combinations of two cards.

{1,2},{1,3},//Used for scoring hands

{1,4},{2,3},

{2,4},{3,4}};

const short sets\_3[ROWS\_10][COLS\_3]={{0,1,2},{0,1,3},//2D array to hold all {0,2,3},{1,2,3},

{0,1,4},{0,2,4},//possible card combinations of

{1,2,4},{0,3,4},//three cards. Used for scoring

{1,3,4},{2,3,4},//hands

{0,2,3},{1,2,3}};

const short sets\_4[ROWS\_5][COLS\_4]={{0,1,2,3},

{0,1,2,4},

{0,1,3,4},

{0,2,3,4},

{1,2,3,4}};

//Declare Variables

short upCard=starter;//sets Up Card

int pointsK=0;//K for "kitty" or, if you prefer "krib"

//Declare arrays

Card crdDeck[DK\_SIZE];

Card handP[HND\_SZ5];//hand with up card

Card handC[HND\_SZ5];//hand with up card

Card crib[HND\_SZ5];//hand with up card

Card sHndP[HND\_SZ4];//hand without up card

Card sHndC[HND\_SZ4];//hand without up card

Card sCrb[HND\_SZ4];//hand without up card

short arHandP[HND\_SZ5];//Holds the sorted card ids for player's hand

short arHandC[HND\_SZ5];//Holds the sorted card ids for computer's hand

short arCrib[HND\_SZ5];//Holds the sorted card ids for crib

short arSHndP[HND\_SZ4];//sorted card ids for player's hand without up card

short arSHndC[HND\_SZ4];//sorted card ids for computer's hand w/o up card

short arSCrb[HND\_SZ4];//sorted card ids for crib without up card

//Character arrays for holding filenames

char fnIds[]="./ids.txt";

char fnRanks[]="./ranks.txt";

char fnSuits[]="./suits.txt";

char fnNames[]="./names.txt";

char fnVals[]="./vals.txt";

//Fill the crdDeck array

flCrdDk(crdDeck,DK\_SIZE,fnIds,fnRanks,fnSuits,fnNames,fnVals);

//Convert hands with up card

cnvHand(crdDeck,handP,arHandP,vecHndP,upCard);

cnvHand(crdDeck,handC,arHandC,vecHndC,upCard);

cnvHand(crdDeck,crib,arCrib,vecCrib,upCard);

//convert hands without up card

cnvSHnd(crdDeck,sHndP,arSHndP,vecHndP);

cnvSHnd(crdDeck,sHndC,arSHndC,vecHndC);

cnvSHnd(crdDeck,sCrb,arCrib,vecCrib);

//fill the Up Card's members

Card cdUpCrd;

cnvCard(crdDeck,cdUpCrd,upCard);

//header

cout<<"Up Card: "<<cdUpCrd.name<<endl;

cout<<"---------------------------------------"<<endl;

//Score hands and output results

if(dealer=='C'){//score hands starting with player if computer is dealer

cout<<"Dealer is the Computer"<<endl;

cout<<"----------------------"<<endl;

cout<<"Player's Hand Score:"<<endl;

scrHand(sets\_2,sets\_3,sets\_4,handP,sHndP,cdUpCrd,pointsP);

cout<<endl<<"Computer's Hand Score:"<<endl;

scrHand(sets\_2,sets\_3,sets\_4,handC,sHndC,cdUpCrd,pointsC);

cout<<endl<<"Crib Score:"<<endl;

scrHand(sets\_2,sets\_3,sets\_4,crib,sCrb,cdUpCrd,pointsK);

//assign crib points to dealer

pointsC+=pointsK;

cout<<"Total points for the Computer\n"

<<"with crib score added: "<<pointsC<<endl;

}else{//score hands starting with Computer if Player is dealer

cout<<"Dealer is the Player"<<endl;

cout<<"----------------------"<<endl;

cout<<"Computer's Hand Score:"<<endl;

scrHand(sets\_2,sets\_3,sets\_4,handC,sHndC,cdUpCrd,pointsC);

cout<<endl<<"Player's Hand Score:"<<endl;

scrHand(sets\_2,sets\_3,sets\_4,handP,sHndP,cdUpCrd,pointsP);

cout<<endl<<"Crib Score:"<<endl;

scrHand(sets\_2,sets\_3,sets\_4,crib,sCrb,cdUpCrd,pointsK);

//assign crib points to dealer

pointsP+=pointsK;

cout<<"Total points for the Player\n"

<<"with crib score added: "<<pointsP<<endl;

}

}

void scrHand(const short combi2[10][2],const short combi3[10][3],

const short combi4[5][4],const Card lngHnd[],

const Card shrtHnd[],const Card cdUpCrd,int & points,

const int size){

//display the hand

cout<<"Hand with Up Card: ";

dsCdHnd(lngHnd,size);

//score pairs

scrPrs(combi2,lngHnd,points);

//score 15s

scr15s(combi2,combi3,combi4,lngHnd,points);

//score for a flush

scrFlsh(lngHnd,shrtHnd,points);

//score nibs

scrNbs(shrtHnd,cdUpCrd,points);

//score runs

scrRns(combi3,combi4,lngHnd,points);

//output points

cout<<"Points: "<<points<<endl;

}

void dsCdHnd(const Card hand[],const int size){

//displays all the contents of a hand

for(int i=0;i<size;i++){

cout<<hand[i].name<<" ";

}

cout<<endl;

}

void scrRns(const short combi3[10][3],const short combi4[5][4],

const Card hand[],int &points){

bool rnFound=false;//used as a flag value to skip tests if larger runs

//are found

//tests if all five cards are only one away from the next. The moment one

//card fails the tests move to the next test

if((hand[1].rank)-(hand[0].rank)==1){

if((hand[2].rank)-(hand[1].rank)==1){

if((hand[3].rank)-(hand[2].rank)==1){

if((hand[4].rank)-(hand[3].rank)==1){

//all cards passed the test

//output the card names and assign +5000 points

cout<<hand[0].name<<" & "<<hand[1].name<<" & "

<<hand[2].name<<" & "<<hand[3].name<<" & "

<<hand[4].name<<" = Run of five: +5000 points"

<<endl;

points+=5000;//give points

rnFound=true;//set that a run of five was found.

//Skip all other tests if true

}

}

}

}

if(!rnFound){

for(int i=0;i<5;i++){//iterate less than the size of the combi4 array

if((hand[combi4[i][1]].rank)-(hand[combi4[i][0]].rank)==1){

if((hand[combi4[i][2]].rank)-(hand[combi4[i][1]].rank)==1){

if((hand[combi4[i][3]].rank)-(hand[combi4[i][2]].rank)==1){

//all cards passed the test

//output the card names and assign +4000 points

cout<<hand[combi4[i][0]].name<<" & "

<<hand[combi4[i][1]].name<<" & "

<<hand[combi4[i][2]].name<<" & "

<<hand[combi4[i][3]].name

<<" = Run of four: +4000 points"

<<endl;

points+=4000;//give points

rnFound=true;//set that a run of five was found.

//Skip all other tests if true

}

}

}

}

}

if(!rnFound){

for(int i=0;i<10;i++){//iterate less than the size of the combi4 array

if((hand[combi3[i][1]].rank)-(hand[combi3[i][0]].rank)==1){

if((hand[combi3[i][2]].rank)-(hand[combi3[i][1]].rank)==1){

//all cards passed the test

//output the card names and assign +3000 points

cout<<hand[combi3[i][0]].name<<" & "

<<hand[combi3[i][1]].name<<" & "

<<hand[combi3[i][2]].name

<<" = Run of three: +3000 points"

<<endl;

points+=3000;//give points

rnFound=true;//set that a run of five was found.

//Skip all other tests if true

}

}

}

}

}

void cnvCard(const Card crdDeck[],Card &card,const short id){

//takes a card Id and fills a card structure element with the proper values

card.id=crdDeck[id].id;

card.rank=crdDeck[id].rank;

card.suit=crdDeck[id].suit;

card.name=crdDeck[id].name;

card.val=crdDeck[id].val;

}

void scrNbs(const Card shrtHnd[],const Card cdUpCrd,int &points,

const int size4){

if(cdUpCrd.rank==11){

for(int i=0;i<size4;i++){

if(shrtHnd[i].rank==cdUpCrd.rank){

cout<<shrtHnd[i].name<<" makes nibs: +1000 points"

<<" and +20% of total points!"<<endl;

points+=1000;

points=(points\*1.2)+0.05;

}

}

}

}

void scrFlsh(const Card hand[],const Card shrtHnd[],int &points,

const int size5){

int cnt=0;//count to hold how many card have the same suit

for(int i=0;i<size5;i++){

if(hand[0].suit==hand[i].suit)cnt++;//increments ctr for each match

}

if(cnt==5){

cout<<"Flush of all five cards: +5000 points"<<endl;

points+=5000;

}

//if all the cards in the hand except for the up card are the same suit

//then +4000 points

if(cnt==4){

int nCnt=0;

for(int i=0;i<4;i++){//4 = go through the hand w/o the up card

//increments nCtr for each suit match without including up card

if(shrtHnd[0].suit==shrtHnd[i].suit)nCnt++;

}

if(nCnt==4){

cout<<"Flush of all four hand cards: +4000 points"<<endl;

points+=4000;

}

}

}

void scr15s(const short combi2[10][2],const short combi3[10][3],

const short combi4[5][4],const Card hand[],int &points){

//Go through hand looking for combinations of two cards which add to 15

for(int i=0;i<10;i++){//goes through const combi2 2D array with 10 rows

//and two columns. Compares ten combinations of two cards.

//uses values in card combinations array to check hand for 15s

if((hand[combi2[i][0]].val)+(hand[combi2[i][1]].val)==15){

//if true, print out names and add 2000 point to points total

cout<<hand[combi2[i][0]].name<<" + "<<hand[combi2[i][1]].name

<<" = 15: +2000 points"<<endl;

points+=2000;

}

}

//Go through hand looking for combinations of three cards which add to 15

for(int i=0;i<10;i++){//goes through const combi3 2D array with 10 rows

//and three columns. Compares ten combinations of three cards.

//uses values in card combinations array to check hand for 15s

if((hand[combi3[i][0]].val)+(hand[combi3[i][1]].val)+

(hand[combi3[i][2]].val)==15){

//if true, print out names and add 2000 point to points total

cout<<hand[combi3[i][0]].name<<" + "<<hand[combi3[i][1]].name

<<" + "<<hand[combi3[i][2]].name<<" = 15: +2000 points"

<<endl;

points+=2000;

}

}

//Go through hand looking for combinations of four cards which add to 15

for(int i=0;i<5;i++){//goes through const combi3 2D array with 5 rows

//and four columns. Compares five combinations of four cards.

//uses values in card combinations array to check hand for 15s

if((hand[combi4[i][0]].val)+(hand[combi4[i][1]].val)+

(hand[combi4[i][2]].val)+(hand[combi4[i][3]].val)==15){

//if true, print out names and add 2000 point to points total

cout<<hand[combi4[i][0]].name<<" + "<<hand[combi4[i][1]].name

<<" + "<<hand[combi4[i][2]].name<<" + "

<<hand[combi4[i][3]].name<<" = 15: +2000 points"<<endl;

points+=2000;

}

}

//Trivial case: checks if all five cards add up to 15. If so, +2000 points

if((hand[0].val)+(hand[1].val)+(hand[2].val)+(hand[3].val)

+(hand[4].val)==15){

//if true, print out names and add 2000 point to points total

cout<<hand[0].name<<" + "<<hand[1].name<<" + "<<hand[2].name<<" + "

<<hand[3].name<<" + "<<hand[4].name<<" = 15: +2000 points"

<<endl;

}

}

void scrPrs(const short combi[10][2],const Card hand[],int &points){

for(int i=0;i<10;i++){//goes through const combi 2D array with 10 rows

//uses values in card combinations array to check hand for pairs

if(hand[combi[i][0]].rank==hand[combi[i][1]].rank){

cout<<hand[combi[i][0]].name<<" & "<<hand[combi[i][1]].name

<<" = A Pair: +2000 points"<<endl;

points+=2000;

}

}

}

//creates a "Small" 4 card hand without the up card

void cnvSHnd(const Card crdDeck[],Card crdSHnd[],short arHnd[],

const vector<short> &vcHnd,const int size4){

//Declare variables

short srtArry[size4];

//convert hands

cpyVcAr(vcHnd,arHnd);

//Copy hand array for sorting

cpyArry(arHnd,srtArry,size4);

//Sort the hand copy in ascending order

selSort(srtArry,size4);

//Copy and add values to members of card hand

for(int i=0;i<size4;i++){

crdSHnd[i].id=crdDeck[srtArry[i]].id;

crdSHnd[i].rank=crdDeck[srtArry[i]].rank;

crdSHnd[i].suit=crdDeck[srtArry[i]].suit;

crdSHnd[i].name=crdDeck[srtArry[i]].name;

crdSHnd[i].val=crdDeck[srtArry[i]].val;

}

}

//creates a five card hand with the up card added

void cnvHand(const Card crdDeck[],Card crdHand[],short arHnd[],

const vector<short> &vcHnd,const short upCard,const int size5){

//Declare variables

short srtArry[size5];

//convert hands

cpyVcAr(vcHnd,arHnd);

//add the Up card

addUpCd(arHnd,upCard,size5);

//Copy hand array for sorting

cpyArry(arHnd,srtArry,size5);

//Sort the hand copy in ascending order

selSort(srtArry,size5);

//Copy and add values to members of card hand

for(int i=0;i<size5;i++){

crdHand[i].id=crdDeck[srtArry[i]].id;

crdHand[i].rank=crdDeck[srtArry[i]].rank;

crdHand[i].suit=crdDeck[srtArry[i]].suit;

crdHand[i].name=crdDeck[srtArry[i]].name;

crdHand[i].val=crdDeck[srtArry[i]].val;

}

}

void cpyArry(short a[],short b[],const int size){

//copies one array to another

for(int i=0;i<size;i++){

b[i]=a[i];

}

}

void selSort(short a[], const int size){

//A selection sort for an array

int strtScn, minVal, minIndx;

for(strtScn=0;strtScn<size-1;strtScn++){

minIndx=strtScn;

minVal=a[strtScn];

for(int i=strtScn+1;i<size;i++){

if(a[i]<minVal){

minVal=a[i];

minIndx=i;

}

}

a[minIndx]=a[strtScn];

a[strtScn]=minVal;

}

}

void addUpCd(short hand[],short upCard,const int size){

hand[size-1]=upCard;//always add upCard to last position in hand array

}

void cpyVcAr(const vector<short> &vcHnd,short aHnd[],const int size){

for(int i=0;i<size;i++){

aHnd[i]=vcHnd[i];//copies value in vector[i] to array[i]

}

}

void banner(char fName[]){

//Declare variables

ifstream input;

string str;

//open file

input.open(fName);

for(int i=0;i<7;i++){

getline(input,str,'\n');

cout<<str<<endl;

}

//clean up

input.close();

}

void flCrdDk(Card crdDeck[],const int size,char fnIds[],char fnRanks[],

char fnSuits[],char fnNames[],char fnVals[]){

ifstream ids;

ifstream ranks;

ifstream suits;

ifstream names;

ifstream vals;

//Open the file stream objects using the character arrays for filenames

ids.open(fnIds, ios::in);

ranks.open(fnRanks, ios::in);

suits.open(fnSuits, ios::in);

names.open(fnNames, ios::in);

vals.open(fnVals, ios::in);

//Fill the array

for(int i=0;i<size;i++){

ids>>crdDeck[i].id;

ranks>>crdDeck[i].rank;

suits>>crdDeck[i].suit;

names>>crdDeck[i].name;

vals>>crdDeck[i].val;

}

//Clean up

ids.close();

ranks.close();

suits.close();

names.close();

vals.close();

}