**Project 1**

Title

**Drewbage Cribbage Game V.1**

Course

**CSC-5**

Section

**40651**

Due Date

**Feb 6, 2014**

Author

**Drew Allensworth**

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

1. **Introduction**

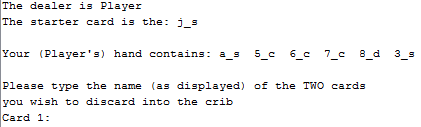
At the outset of this project, I knew that one of the most difficult games for me to implement with any modicum of success would be a card game. The difficulty, as I saw it, would be in attempting to accurately model the behavior of having cards in one’s hand and then using them. That is, to assign cards 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.

1. **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. More options will be added later but the current options are “Type 1 to Play Game” and pressing any key to quit the program.

At the outset of the game, the dealer is determined and both the computer and the Player are dealt six cards. The Player should notice that the “Starting Card” is displayed. This will come in handy during Version 2 of Drewbage when hand scoring will be implemented.

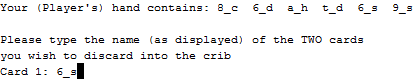


The first action the Player will undertake is discarding two cards into the “crib”. The “crib” hand will be used during Version 2. However, rest assured that although you as the Player are not using it yet, the crib has been filled properly and contains the cards given to it. As it stands right now, the game that the Player is playing is the portion of cribbage called The Play. This occurs after the dealer is announced, the starting card is determined, and two cards from each player have been given to the crib.

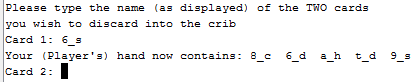
At this point, I would like to take the time to detail some of the codes used in the program:

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.

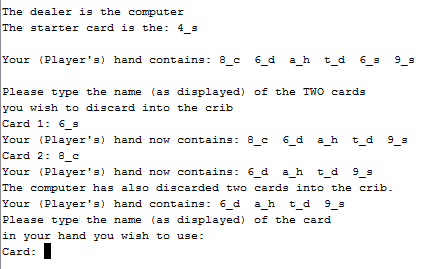
|  |  |
| --- | --- |
| \_ underscore character | “of” |
| ***Ranks*** | |
| a | Ace |
| 2 | two |
| 3 | three |
| 4 | four |
| 5 | five |
| 6 | six |
| 7 | seven |
| 8 | eight |
| 9 | nine |
| t | ten |
| j | Jack |
| q | Queen |
| k | King |
| ***Suits*** | |
| c | clubs |
| d | diamonds |
| h | hearts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

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

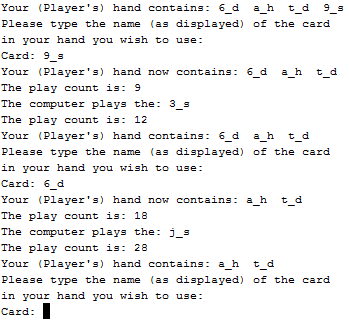
Let’s enter that “6\_s” (which is the “six of spades”)



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. The following image gives more information, showing that the dealer is the computer. 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.



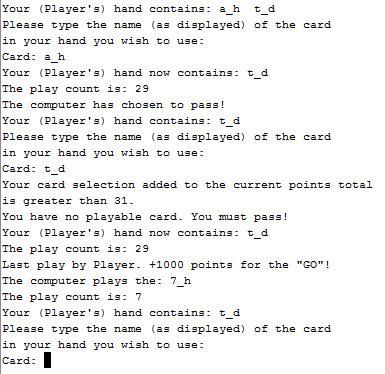
Two turns later:

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

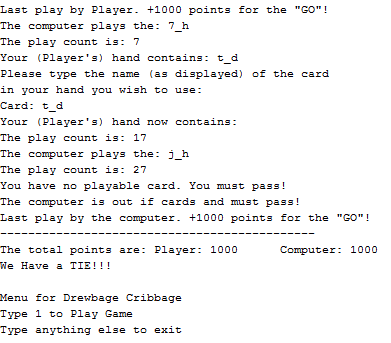
You play the nine of spades. The card is removed from your hand. The count is 9. The computer plays the three of spades and the count is now 9 + 3 = 12 This then continues until the score reaches as close to 31 as possible without going over 31.

|  |  |
| --- | --- |
| **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:

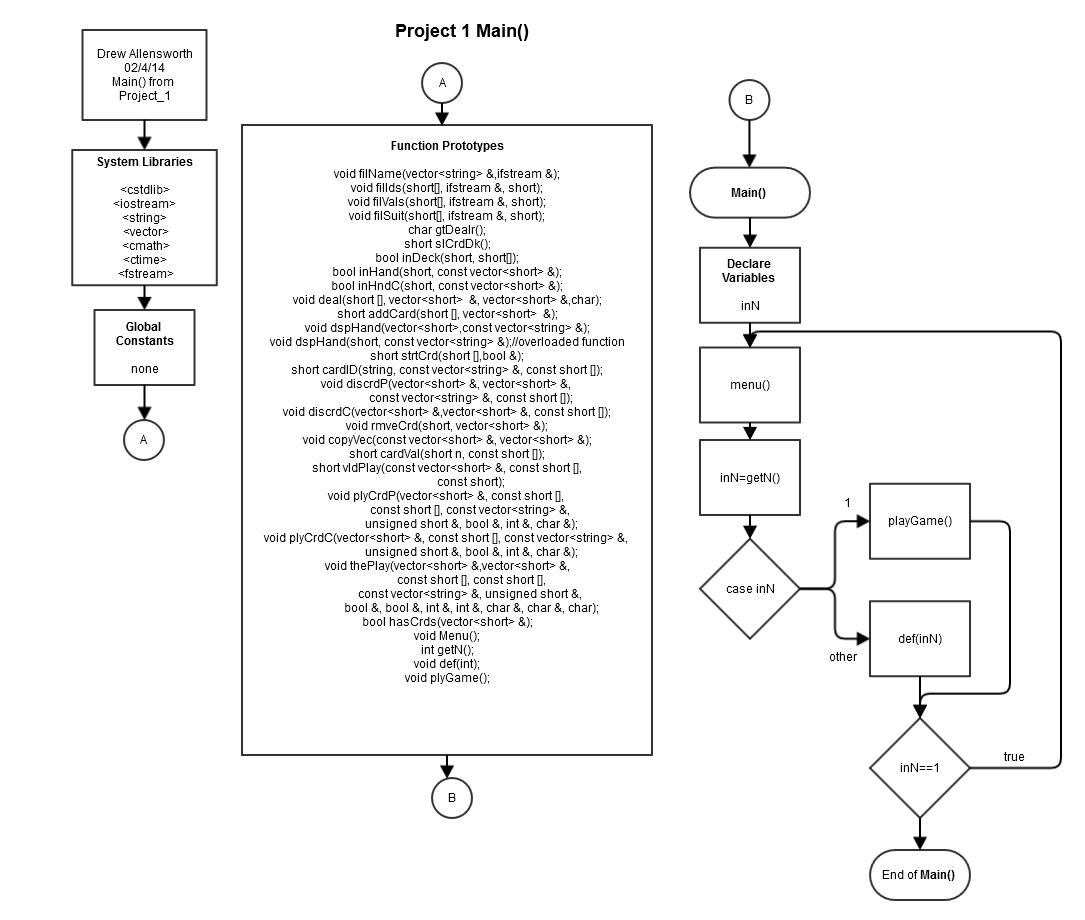


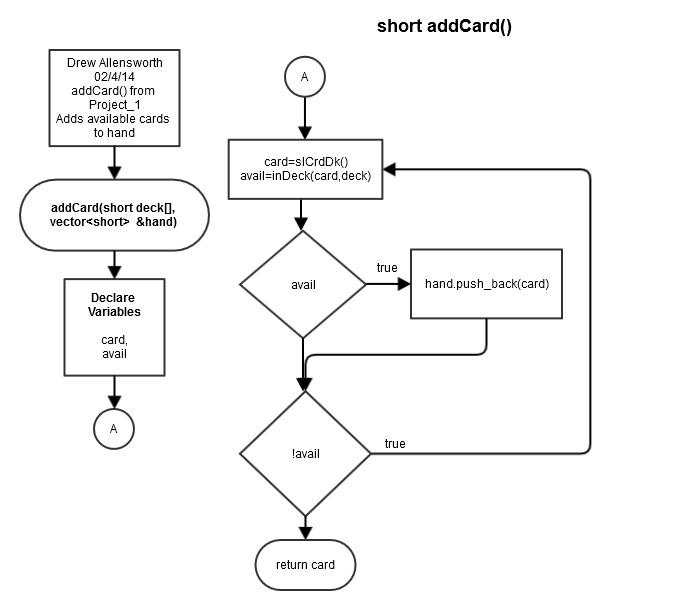
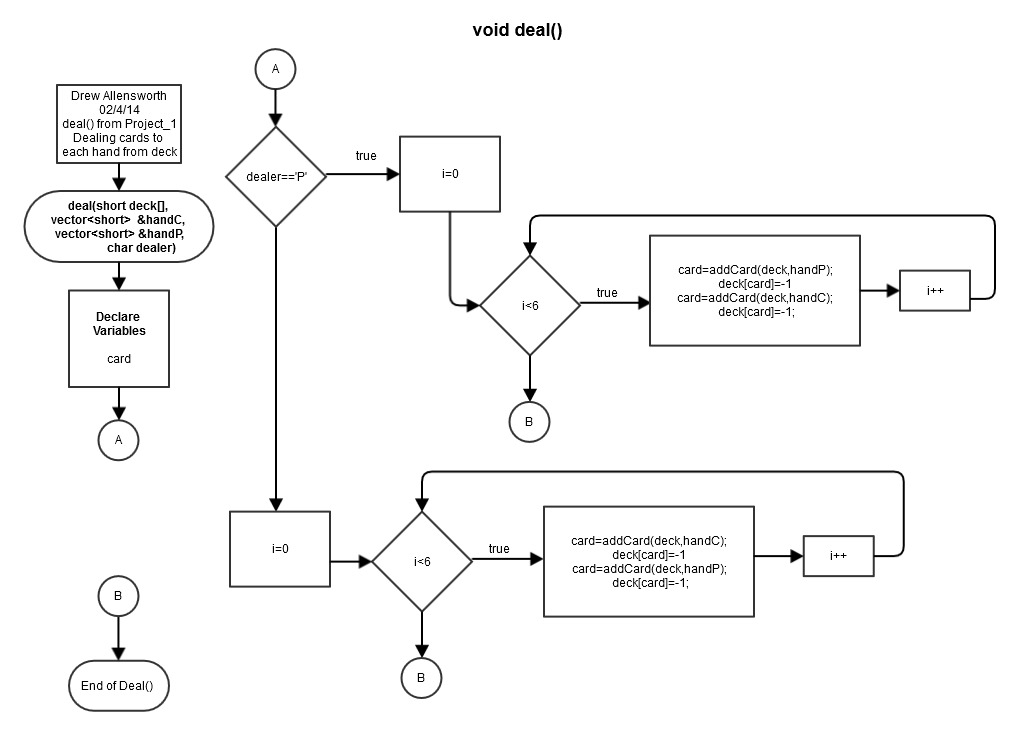
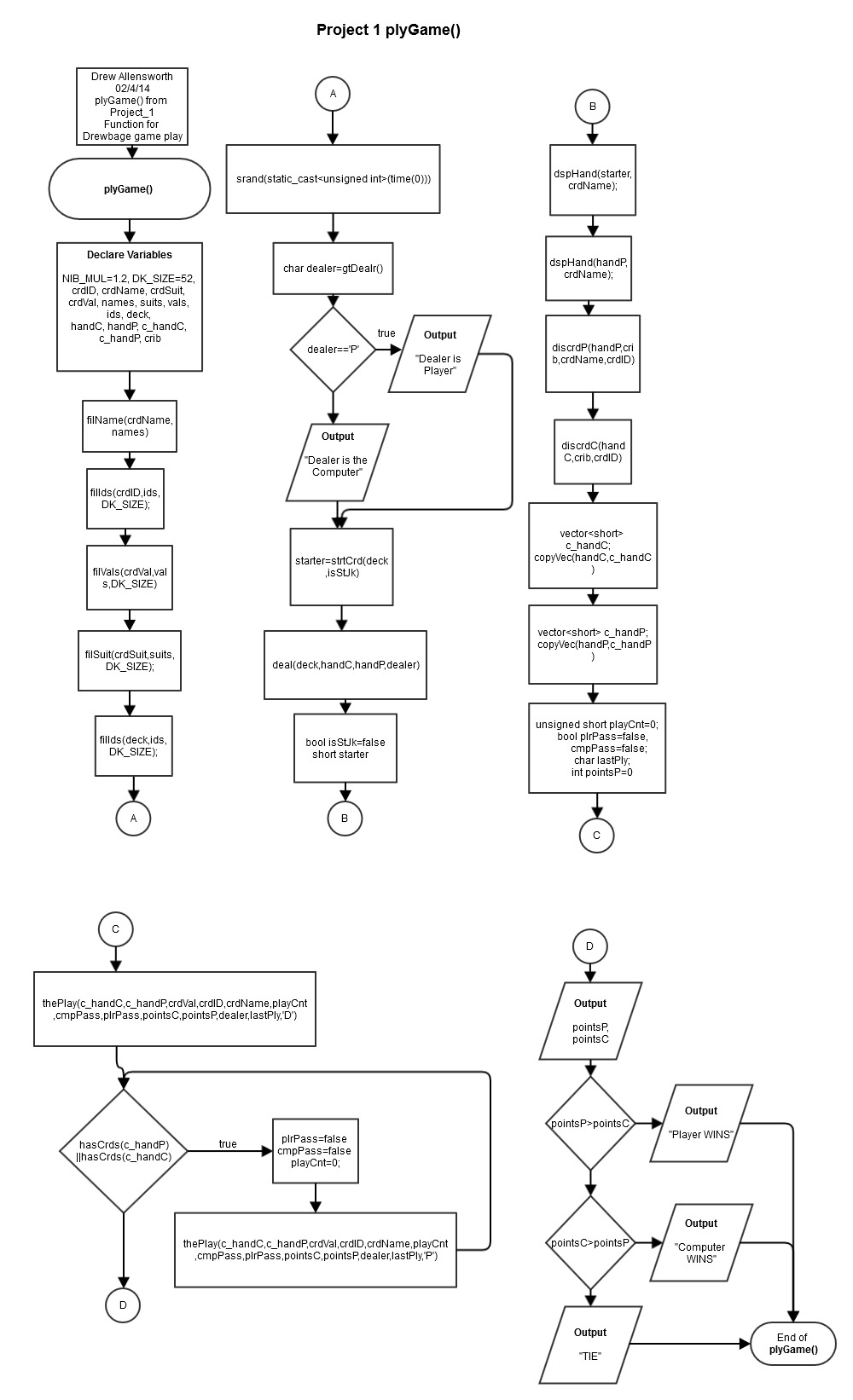
If you follow the count, you’ll notice that at a certain point the program will ask you to enter a card that will send the count over 31. Don’t worry, give it the card and the program will now that not only is that card not playable, but you don’t have a playable card in your hand for that given count. 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, points will be tallied and a winner will be announced. 

Finally, the following is the scoring breakdown. 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 |

1. **Flowcharts**



1. **Functions**

|  |  |
| --- | --- |
| **Function Descriptions and Comments** | |
| **Function Name** | **Description** |
| void banner(string) | Parameters: string for the filename  Displays banner character image from a file |
| 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 banner(char dealer, string) | Parameters: bool dealer from plyGame(); string for the filename  !OVERLOADED FUNCTION! Character dealer is used to indicate who gets nibs. Displays a banner from a file. Displays a message indicating that the bonus will be applied to the dealer |
| 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. |

|  |  |  |  |
| --- | --- | --- | --- |
| **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() | unsigned short | stCrdID | 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 |

1. **Concepts Used-References Gaddis Chapters**

|  |  |
| --- | --- |
| 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 |
| Chapter 3 | cin object  type conversion  type casting  mathematical expressions |
| 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 |
| Chapter 5 | while loop  do-while loop  for loop  nested loops  using files for data  breaking a loop |
| Chapter 6 | defining and calling functions  function prototypes  sending data into a function  passing data by value  return statement  reference variables  overloading functions |
| Chapter 7 | array initialization  parallel arrays  processing array contents  arrays as function arguments  STL vector |
| Chapter 8 | searching arrays  searching vectors |

1. **Pseudocode and General Logical Layout of the Program**

***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~~
4. **Program listing**

/\*

\* File: main.cpp

\* Author: Drew Allensworth

\*

\* Created on February 3, 2014, 4:05 PM

\* Purpose: Program for CSC-5 Project 1

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

\* 1 player against the computer.

\*

\*/

//System Libraries

#include <cstdlib>

#include <iostream>

#include <string>

#include <vector>

#include <cmath>//for rand()

#include <ctime>//for srand()

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

using namespace std;

//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();

//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(){

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)));

//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

//display the starter card

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

dspHand(starter,crdName);

cout<<endl<<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;

//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;

//'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');

}

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

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

<<pointsC<<endl;

//determine who wins

(pointsP>pointsC)?cout<<"Player WINS!!!":

(pointsC>pointsP)?cout<<"Computer WINS!!!":

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

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

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 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: ";

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{//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 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();

}