PROJECT 1

< LOTERÍA >

Name: Andrew Guzman

Date: 11/06/2022

CSC-17A-48290

INTRODUCTION

Lotería is a traditional Mexican board game of chance, similar to American Bingo. It includes a deck of 54 cards that consist of an image, name, and number (the numbers listed on the cards are not usually used). The game comes with a set of 20 boards each with a randomly generated 4x4 grid of cards. There can be 2 to 10 players plus a "caller" who calls out the cards that are pulled from the shuffled deck. A winning pattern is set prior to starting the game (full board, row, column, corners, diagonals, 2x2 square, or 3x3 square). Players use tokens such as dried beans, small rocks, or bottle caps to mark matching cards found on their board. The winner is the first player to shout out Lotería right after completing the winning pattern.

To start the game, the caller shuffles the deck and pulls cards from the deck one-by-one. The caller pulls a card and instead of calling out the name on the card, they say a riddle that corresponds to the name. The players must decipher the riddle to get the name and then search for the matching card on their board. Some riddles are easier than others, as they contain the name in the riddle itself. As the game progresses and players find matches, players are checking their board against the winning pattern to see if they match.

The goal is to be the first one to match the winning pattern and shout Lotería. It's possible that there are zero, one, or multiple winners. If there are multiple winners, the remaining players reset their boards to start a new mini game with the same board. The caller reshuffles the deck and picks cards, the first player to match a corner wins the game.

I chose to create Lotería because I grew up playing it with my family so I'm very familiar with the rules of the game. It plays an essential part of Mexican culture and is relatively easy for all ages to play. Since the game is so popular, it was important that I recreate the game by replacing cards that normalize problematic stereotypes of historically marginalized people. Additionally, the game is in Spanish, so this project allows me to share the game with non-Spanish speakers (though some of the riddles get lost in translation and make no sense whatsoever). This may be something I can expand on in the future so people of different languages/cultures can create their own version of the game.

Summary

Language	files	blank	comment	code
C++	1	75	66	584

The program uses 6 structs, 23 major variables, and 25 functions. It meets the criteria for a first project because of its size and complexity - I used most of the concepts from chapters 9-12. This was a challenging project to keep up with because of its size. I took about a week to brainstorm/plan before I started writing the program which then took me about 8 days.

DESCRIPTION

I started by implementing one of each of the most basic elements of my game (1 image, 1 card, 1 player). I ensured that I was able to print the contents of the data stored in these structures before moving on. Then, I worked my way up to a deck of cards, a board, and a set of players. Next, I implemented more complex functionalities such as shuffling a deck of cards, generating random boards,

and printing the boards. Once I was the more complex pieces were working nicely, I was able to implement the logic of the game itself. I decided to create a computer vs computer game because it felt unfair to try to make it human vs. computer. I would want to expand on the program to create 20 random boards at the start of each game, allow users to decide how many computer players (2-10), and detect winners with a non-full winning pattern.

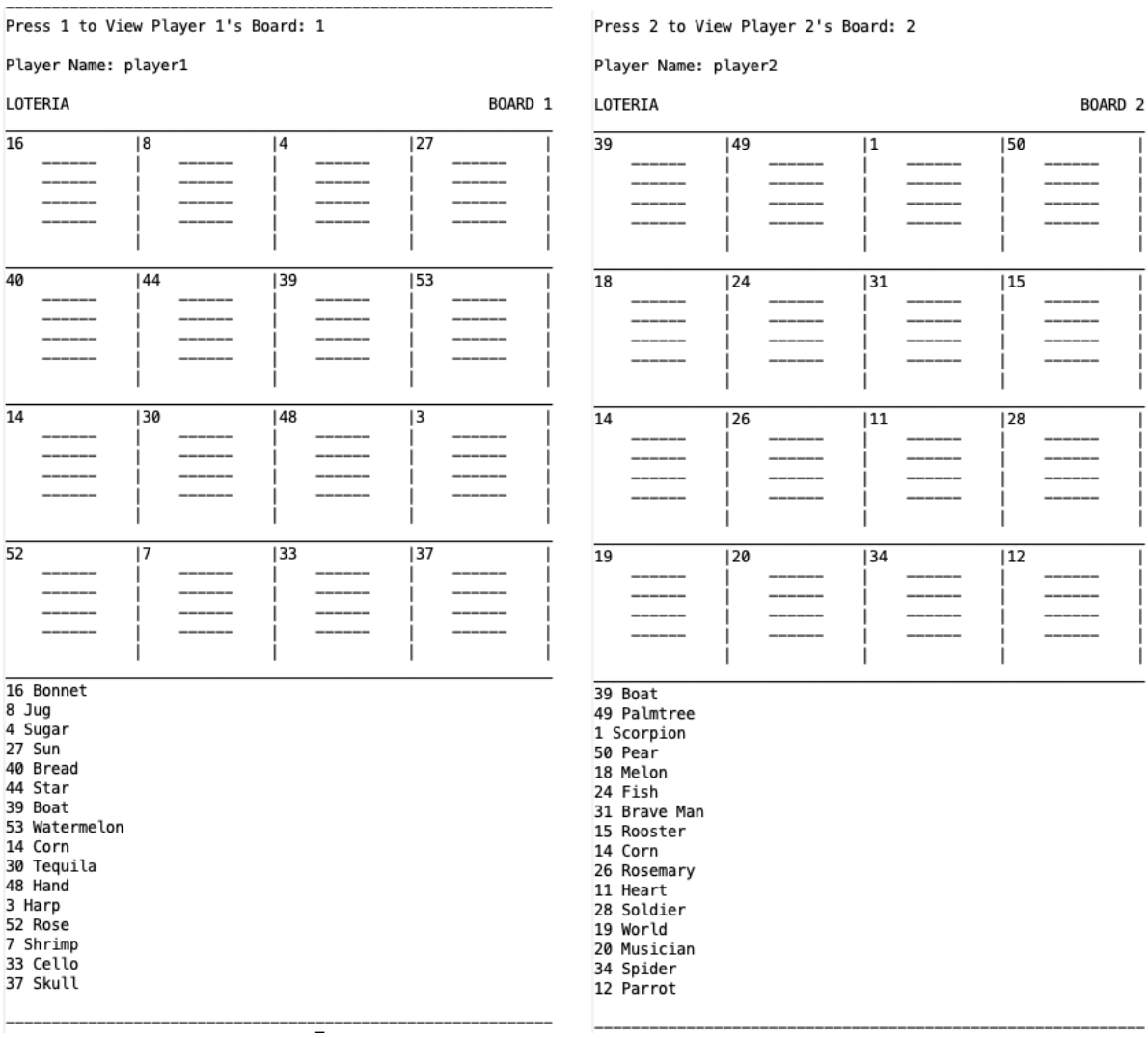
Sample Input/Output

```
LOTERÍA

This is a game of Lotería with 2 computer players and 1 caller.
You are the caller of this game.
------------------------------------------------------------
SET THE WINNING PATTERN
Options: #1=Full, #2=Row, #3=Column, #4=Diagonals

Enter Pattern #: 1
Players Must Match One of The Following Patterns to Win:
Full
| * || * || * || * |
| * || * || * || * |
| * || * || * || * |
| * || * || * || * |

------------------------------------------------------------
```

Enter 1: Pattern = Full

```
------------------------------------------------------------
SET THE NUMBER OF TOKENS
Tokens are Shared Among Players.
Enter Number of Tokens(16 to 32): 25
------------------------------------------------------------
Press 9 to Shuffle The Deck of Cards: 9
Deck Shuffled!
------------------------------------------------------------
```

Enter 9: Shuffle the deck



Enter 1: Player 1 gets random board

Enter 2: Player 2 gets random board

Enter 9: Pull cards from the deck

```
------------------------------------------------------------
ROUND 7:
Number of Tokens Remaining: 20

Press 9 to Pull a Card From the Deck: 9
_____________
  37
    ______
    ______
    ______
    ______
    ______
Skull
_____________

Riddle: As I Passed By The Cemetery I Found Myself A Skull

player1 Found a Match At 4, 4.


player1        3/16
|   ||   ||   ||   |
|   ||   ||   ||   |
| * ||   || * ||   |
|   ||   ||   || * |

player2        3/16
|   ||   ||   ||   |
|   ||   ||   ||   |
| * ||   || * ||   |
|   ||   || * ||   |

------------------------------------------------------------
```

Output: Either plyr find match

```
ROUND 10:
Number of Tokens Remaining: 17

Press 9 to Pull a Card From the Deck: 9
_______________
  39
    ______
    ______
    ______
    ______
    ______
Boat
_______________

Riddle: She Rows And Rows Sitting In Her Little Boat

player1 Found a Match At 2, 3.

player2 Found a Match At 1, 1.

player1        5/16
| * ||   ||   ||   |
|   ||   || * ||   |
| * ||   || * ||   |
|   ||   ||   || * |

player2        5/16
| * ||   ||   || * |
|   ||   ||   ||   |
| * ||   || * ||   |
|   ||   || * ||   |

------------------------------------------------------------
```

Output: Both plyrs find match

```
player1       11/16
| * ||   || * || * |
| * ||   || * ||   |
|   || * ||   || * |
| * || * || * || * |

player2       14/16
| * || * || * ||   |
| * || * || * || * |
| * || * || * || * |
| * || * ||   || * |

----------------------GAME OVER----------------------------
Saving player1 Game Data...
Displaying player1 Game Data...
Name: player1
Number of Matches Found This Round: 11

Saving player2 Game Data...
Displaying player2 Game Data...
Name: player2
Number of Matches Found This Round: 14

Displaying Game Data...
Pattern: Full
Number of Rounds: 44
Number of Tokens Remaining: 0
Number of Winners: 0
```

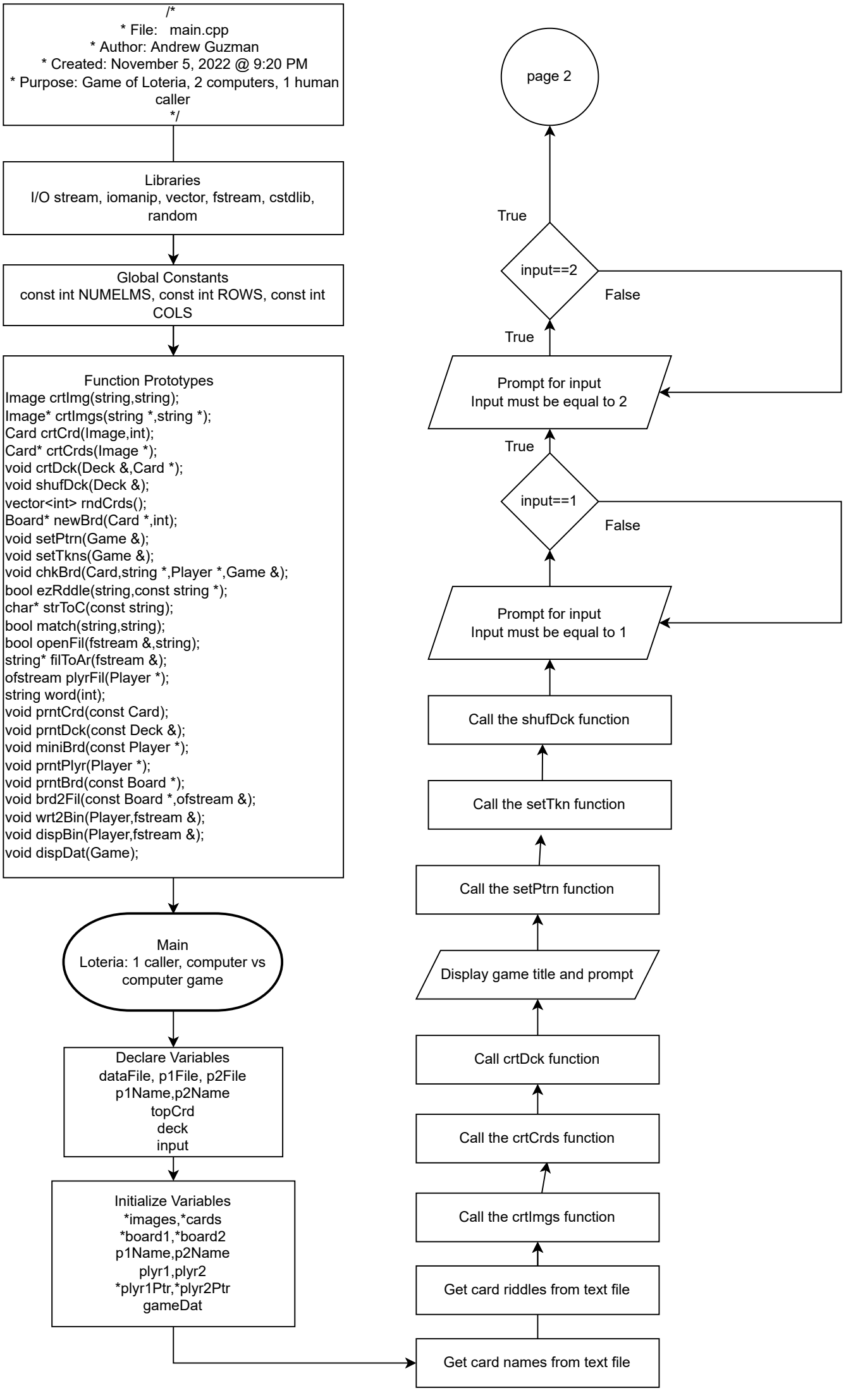
Output: Neither plyr wins

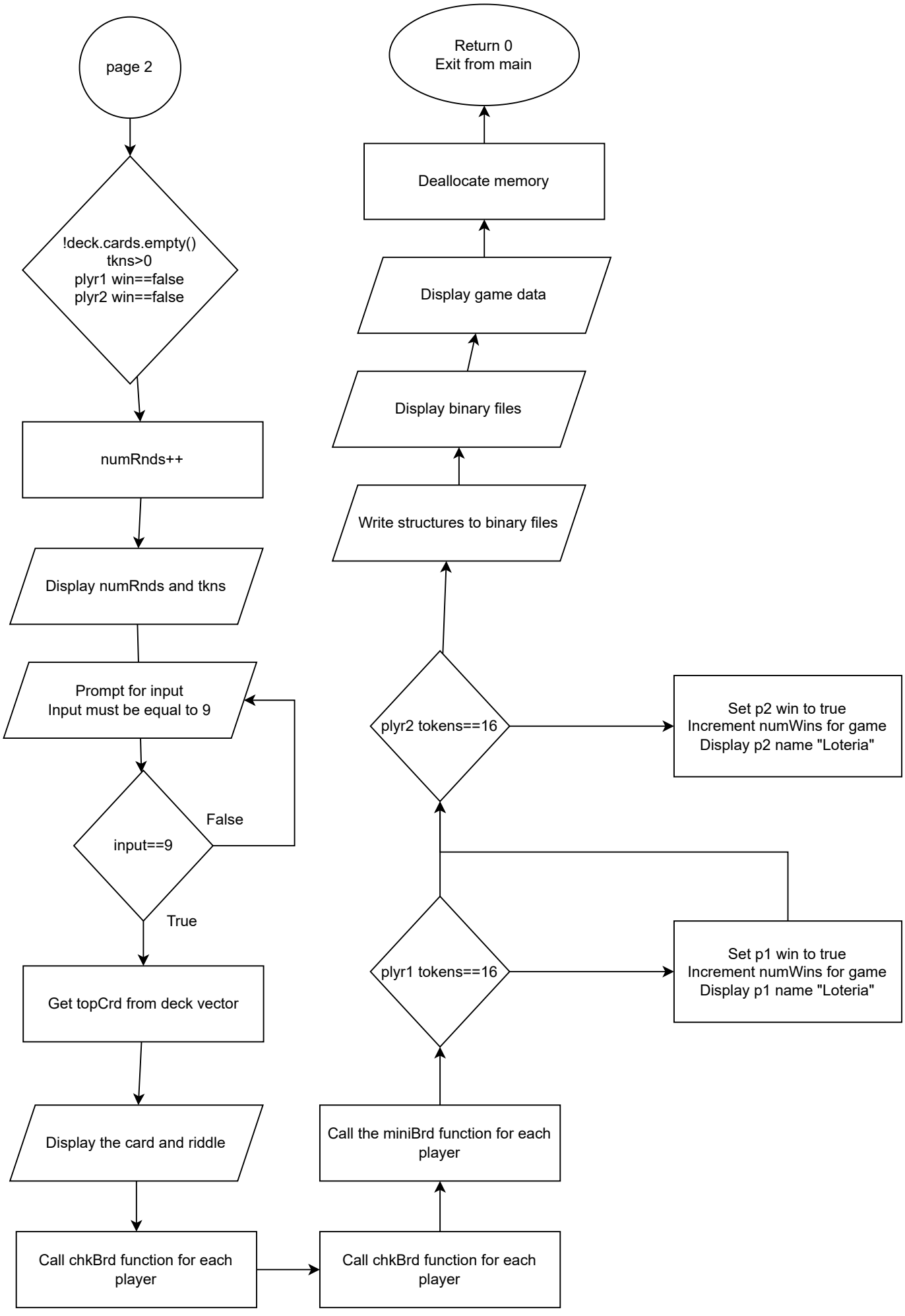
```
player1       16/16
| * || * || * || * |
| * || * || * || * |
| * || * || * || * |
| * || * || * || * |

player2       14/16
| * || * || * || * |
|   || * || * || * |
| * || * || * || * |
| * || * || * ||   |

player1: LOTERÍA!

----------------------GAME OVER----------------------------
Saving player1 Game Data...
Displaying player1 Game Data...
Name: player1
Number of Matches Found This Round: 16

Saving player2 Game Data...
Displaying player2 Game Data...
Name: player2
Number of Matches Found This Round: 14

Displaying Game Data...
Pattern: Full
Number of Rounds: 46
Number of Tokens Remaining: 2
Number of Winners: 1
```

Output: Either plyr wins

```
player1       16/16
| * || * || * || * |
| * || * || * || * |
| * || * || * || * |
| * || * || * || * |

player2       16/16
| * || * || * || * |
| * || * || * || * |
| * || * || * || * |
| * || * || * || * |

player1: LOTERÍA!

player2: LOTERÍA!

----------------------GAME OVER----------------------------
Saving player1 Game Data...
Displaying player1 Game Data...
Name: player1
Number of Matches Found This Round: 16

Saving player2 Game Data...
Displaying player2 Game Data...
Name: player2
Number of Matches Found This Round: 16

Displaying Game Data...
Pattern: Full
Number of Rounds: 53
Number of Tokens Remaining: 0
Number of Winners: 2
```

Output: Both plyrs win


/*
* File: main.cpp
* Author: Andrew Guzman
* Created: November 5, 2022 @ 9:20 PM
* Purpose: Game of Loteria, 2 computers, 1 human caller
*/
Libraries
I/O stream, iomanip, vector, fstream, cstdlib, random
Global Constants
const int NUMELMS, const int ROWS, const int COLS
Function Prototypes
Image crtImg(string,string);
Image* crtImgs(string *,string *);
Card crtCrd(Image,int);
Card* crtCrds(Image *);
void crtDck(Deck &,Card *);
void shufDck(Deck &);
vector<int> rndCrds();
Board* newBrd(Card *,int);
void setPtrn(Game &);
void setTkns(Game &);
void chkBrd(Card,string *,Player *,Game &);
bool ezRddle(string,const string *);
char* strToC(const string);
bool match(string,string);
bool openFil(fstream &,string);
string* filToAr(fstream &);
ofstream plyrFil(Player *);
string word(int);
void prntCrd(const Card);
void prntDck(const Deck &);
void miniBrd(const Player *);
void prntPlyr(Player *);
void prntBrd(const Board *);
void brd2Fil(const Board *,ofstream &);
void wrt2Bin(Player,fstream &);
void dispBin(Player,fstream &);
void dispDat(Game);
Main
Loteria: 1 caller, computer vs computer game
Declare Variables
dataFile, p1File, p2File
p1Name,p2Name
topCrd
deck
input
Initialize Variables
*images,*cards
*board1,*board2
p1Name,p2Name
plyr1,plyr2
*plyr1Ptr,*plyr2Ptr
gameDat
Get card names from text file
Get card riddles from text file
Call the crtImgs function
Call the crtCrds function
Call crtDck function
Display game title and prompt
Call the setPtrn function
Call the setTkn function
Call the shufDck function
Prompt for input
Input must be equal to 1
True
input==1
False
Prompt for input
Input must be equal to 2
True
input==2
False
True
page 2


page 2
!deck.cards.empty()
tkns>0
plyr1 win==false
plyr2 win==false
numRnds++
Display numRnds and tkns
Prompt for input
Input must be equal to 9
input==9
False
True
Get topCrd from deck vector
Display the card and riddle
Call chkBrd function for each player
Call chkBrd function for each player
Call the miniBrd function for each player
plyr1 tokens==16
Set p1 win to true
Increment numWins for game
Display p1 name "Loteria"
plyr2 tokens==16
Set p2 win to true
Increment numWins for game
Display p2 name "Loteria"
Write structures to binary files
Display binary files
Display game data
Deallocate memory
Return 0
Exit from main

Pseudocode

Get names from text file

Get riddles from text file

Create images using names and riddles

Create cards using the images

Create a deck of cards from cards

Set the winning pattern

Set the number of tokens

Shuffle the deck of cards

Create and display player boards

Game starts while the deck is not empty, tokens are available, and no players have won

Pull a card from the deck

Display the card and its riddle

Send the card info to the computer players to check their boards for

> Each player must take in the riddle, check if it's "easy" (the name is in the riddle) to get the name.
>
> Get the name of the card and search for a match on board.

If a player has a match, mark that element on the board as found. Decrease the number of tokens available by one. Increase number of matches found by player.

Display updated board pattern of players.

Remove the called card from the deck

Check if players have a full board.

> If a board matches the winning pattern, mark the player as a winner, increase number of winners in the game, and display the players name and "Loteria!"

Continue looping until the game ends (the deck is empty, 0 tokens are available, and 1+ players have won)

Variables

Structures

Type	Variable Name	Description
struct	Image	Image structure
string	riddle	Riddle that corresponds to an image
string	name	Name that corresponds to an image and printed on the card
struct	Card	Card structure
int	num	Number displayed on card
int	xIndex	X Index in the board
int	yIndex	Y Index in the board
bool	found	Flag to mark if card is found on board
Image	img	Image in card structure
struct	Deck	Deck structure
vector<Card>	cards	Vector to store deck of cards
int	maxSize	Maximum size of vector/deck of cards
struct	Board	Board structure
int	brdNum	Number printed on the board
Card [][]	board	2d array of cards makes up the board
struct	Player	Player structure
string	name	//Name of the player
bool	win	//Flag to set to true if player wins
int	plyrTkn	//Number of tokens used during game
Board*	board	//Pointer to a board in player structure
struct	Game	Game structure
int	tkns	Number of tokens available for game
int	pattern	Winning pattern set for game
int	numWins	Number of winners for given game
int	numRnds	Number of rounds in game

Major Variables

Type	Variable Name	Description	Location
const int	NUMELMS	# of elements(lines in files,names,riddles,images,cards/deck)	main()
const int	ROWS	Number of rows in board	main()
const int	COLS	Number of columns in board	main()
fstream	dataFile	Input files	main()
ofstream	p1File,p2File;	Player files to output to	main()
Image *	images	Array of images	main()
Card *	cards	Array of cards	main()
Card *	topCard	Card pulled from top of the deck	main()
Deck	deck	Deck of cards	main()

int	input	Input for user	main()
string *	riddles	String array of riddles used create images	main()
string *	names	String array of names used create images and cards	main()
Board *	board1,board2	Pointers to boards	main()
string	p1Name,p2Name;	Names of players	main()
Player	plyr1,plyr2;	Player structures	main()
Player *	plyr1Ptr,plyr2Ptr;	Pointers to player structures	main()
Game	gameDat	Game structure	main()

Concepts

CH	Section	Concept	Pts	Location	Comments
9		**Pointers/Memory Allocation**			
	1	Memory Addresses		114	
	2	Pointer Variables	5	111	
	3	Arrays/Pointers	5	107	
	4	Pointer Arithmetic			
	5	Pointer Initialization		114	
	6	Comparing			
	7	Function Parameters	5	67	
	8	Memory Allocation	5	267	
	9	Return Parameters	5	266	
	10	Smart Pointers			
10		**Char Arrays and Strings**			
	1	Testing			
	2	Case Conversion			
	3	C-Strings	10	474	
	4	Library Functions		477	strstr: search for a string in another string
	5	Conversion			
	6	Your own functions		492	strToC: c_str() returns const char *, needed char *
	7	Strings	10	251	
11		**Structured Data**			
	1	Abstract Data Types			
	2	Data		26	
	3	Access		709	
	4	Initialize		115	

	5	Arrays	5	252	
	6	Nested	5	36	
	7	Function Arguments	5	259	
	8	Function Return	5	244	
	9	Pointers	5	111	
	10	Unions ****			
	11	Enumeration	5	18, 348	
12		**Binary Files**			
	1	File Operations		503	
	2	Formatting	2	638	
	3	Function Parameters	2	512	
	4	Error Testing		506	
	5	Member Functions	2	516	
	6	Multiple Files	2	99	
	7	Binary Files	5	691	
	8	Records with Structures	5	682	
	9	Random Access Files	5		
	10	Input/Output Simultaneous	2		

TOTAL 100

REFERENCES

1. vector<int> rndCrds() : To generate random card numbers to create random boards.
   - https://cplusplus.com/reference/algorithm/shuffle/
   - https://en.cppreference.com/w/cpp/algorithm/random_shuffle
2. void shufDck(Deck &): To generate the random card numbers of the cards that would go on the boards (not the cards in the deck).
   - https://www.delftstack.com/howto/cpp/shuffle-vector-cpp/
3. bool match(string,string) :To convert strings to c-strings
   - https://cplusplus.com/reference/string/string/c_str/
4. bool match(string,string) : To check if a c-string contains another c-string.
   - https://en.cppreference.com/w/cpp/string/byte/strstr
   - Gaddis Textbook 10-6
5. void wrt2Bin(Player p,fstream &file): To write the contents of the structure to a binary file.
   - Gaddis Textbook 12-20
6. void dispBin(Player p,fstream &file): To display the contents of binary file.
   - Gaddis Textbook 12-21

PROGRAM

```
/*
* File:   main.cpp
* Author: Andrew Guzman
* Created: November 5, 2022 @ 9:20 PM
* Purpose: v10: Memory de-allocation & code clean up - no major changes. Final version.
*/

//System Libraries
#include <iostream>  //Input/output Library
#include <iomanip>
#include <vector>
#include <fstream>
#include <cstdlib>
#include <random>
using namespace std;

//User Libraries
enum BrdWin {NONE=0,FULL=1,ROW=2,COL=3,DIAGS=4};   //Winning board patterns

//Global Constants
const int NUMELMS=54;   //# of elements(lines in files,names,riddles,images,cards/deck)
const int ROWS=4;       //Number of rows in board
const int COLS=4;       //Number of columns in board

//Structure Definitions
struct Image {
   string riddle;  //Riddle that corresponds to an image
   string name;    //Name that corresponds to an image and printed on the card
};

struct Card {
   int num;       //Number displayed on card
   int xIndex;    //X Index in the board
   int yIndex;    //Y Index in the board
   bool found;    //Flag to mark if card is found on board
   Image img;     //Image in card structure
};

struct Deck {
   vector<Card> cards; //Deck of cards
   int maxSize;       //Maximum size of deck
};

struct Board {
   int brdNum;              //Number printed on the board
   Card board[ROWS][COLS];    //2d array of cards makes up the board
```

```
};

struct Player {
   string name;    //Name of the player
   bool win;       //Flag to set to true if player wins
   int plyrTkn;    //Number of tokens used during game
   Board* board;   //Pointer to a board in player structure
};

struct Game {
   int tkns;       //Number of tokens available for game
   int pattern;    //Winning pattern set for game
   int numWins;    //Number of winners for given game
   int numRnds;    //Number of rounds in game
};

//Function Prototypes
Image crtImg(string,string);   //Creates 1 image structure
Image* crtImgs(string *,string *); //Returns array of image structures
Card crtCrd(Image,int);        //Creates 1 card structure
Card* crtCrds(Image *);        //Returns array of card structures
void crtDck(Deck &,Card *);    //Creates a deck of cards using array of cards
void shufDck(Deck &);          //Shuffles deck of cards
vector<int> rndCrds();         //Generates 16 random card numbers for board
Board* newBrd(Card *,int);     //Returns pointer to a random board
void setPtrn(Game &);          //Sets winning pattern for the game
void setTkns(Game &);          //Sets the number of tokens for players to share
void chkBrd(Card,string *,Player *,Game &); //Checks board and updates player and game data
bool ezRddle(string,const string *); //Checks if a riddle is "easy"
char* strToC(const string);    //Returns pointer to char array
bool match(string,string);     //Searches for string in another string
bool openFil(fstream &,string); //Opens a file for input
string* filToAr(fstream &);    //Saves file contents to memory
ofstream plyrFil(Player *);    //Create text file to save player game data to
string word(int);              //Returns word for winning pattern set
void prntCrd(const Card);      //Prints 1 card structure
void prntDck(const Deck &);    //Print deck of cards
void miniBrd(const Player *);  //Prints player's mini boards on screen
void prntPlyr(Player *);       //Prints player game data to the screen
void prntBrd(const Board *);   //Prints 1 board to the screen
void brd2Fil(const Board *,ofstream &); //Prints a board to a file
void wrt2Bin(Player,fstream &);    //Outputs contents of player to binary files
void dispBin(Player,fstream &);    //Prints contents of binary file
void dispDat(Game);            //Prints game data

//Program Execution Begins Here!!!
int main(int argc, char** argv) {
   //Random number seed
```

```
srand(static_cast<unsigned int>(time(0)));

//Declare Variables
fstream dataFile;       //Input files
ofstream p1File,p2File; //Player files to output to
string p1Name,p2Name;   //Name of players
Card topCrd;            //Card pulled from top of the deck
Deck deck;             //Deck of cards
int input;           //Input for user

//Initial Variables
input=0;
string *riddles=nullptr,    ////String array of riddles used create images
       *names=nullptr;      //String array of names used create images and cards
Image *images=nullptr;  //Array of images
Card *cards=nullptr;    //Array of cards
Board *board1=nullptr,*board2=nullptr;      //Pointers to boards
p1Name="player1",p2Name="player2";
Player plyr1={p1Name,false},plyr2={p2Name,false};       //Initialize player structures
Player *plyr1Ptr=&plyr1,*plyr2Ptr=&plyr2;   //Pointers to player structures
Game gameDat={NONE,0,0,0};

//Get card names from cardNames.txt file and save to names array
if(openFil(dataFile,"cardNames.txt")) {
   names=filToAr(dataFile);
   dataFile.close();
} else {
   cout<<"File open error!"<<endl;
}

//Get card riddles from cardRiddles.txt file and save to riddles array
if(openFil(dataFile,"cardRiddles.txt")) {
   riddles=filToAr(dataFile);
   dataFile.close();
} else {
   cout<<"File open error!"<<endl;
}

//Create all images
images=crtImgs(names,riddles);

//Create all cards using the images
cards=crtCrds(images);

//Create deck of cards
crtDck(deck,cards);

 //Display game title
```

```
   cout<<"LOTERÍA\n"<<endl;

   //Display game prompt
   cout<<"This is a game of Lotería with 2 computer players and 1 caller.\n"
       "You are the caller of this game."<<endl;

//Game Begins
   //User sets the winning pattern for this
   setPtrn(gameDat);

   //User sets the number of tokens for player to use during game
   setTkns(gameDat);

   //User shuffles the deck of cards
   shufDck(deck);

   //Create and display player 1 board
   do {
      cout<<"Press 1 to View Player 1's Board: ";
      cin>>input;
   } while(input!=1);
   board1=newBrd(cards,1);        //Create a random board for player1
   plyr1Ptr->board=board1;        //Assign board to player1
   p1File=plyrFil(plyr1Ptr);      //Create new file to save player1 data to
   prntPlyr(plyr1Ptr);
   cout<<"------------------------------------------------------------"<<endl;

   //Create and display player 2 board
   do {
      cout<<"Press 2 to View Player 2's Board: ";
      cin>>input;
   } while(input!=2);
   board2=newBrd(cards,2);    //Create a random board for player2
   plyr2Ptr->board=board2;       //Assign board to player2
   plyrFil(plyr2Ptr);            //Create file to save player2 data to
   prntPlyr(plyr2Ptr);

   //Play game while deck is not empty, tokens still available,and no win
   while(!deck.cards.empty()&&gameDat.tkns>0&&plyr1Ptr->win==false&&plyr2Ptr->win==false) {
      gameDat.numRnds++;
      cout<<"------------------------------------------------------------"<<endl;
      cout<<"ROUND "<<gameDat.numRnds<<": "<<endl;
      cout<<"Number of Tokens Remaining: "<<gameDat.tkns<<endl<<endl;
      do {
         cout<<"Press 9 to Pull a Card From the Deck: ";
         cin>>input;
      }while(input!=9);
```

```
    if(input==9) {
        //Caller pulls a card from the deck
        topCrd=deck.cards.front();
        //Display pulled card and corresponding riddle on the screen
        prntCrd(topCrd);
        //Send the card info to the computer players to check their boards
        chkBrd(topCrd,names,plyr1Ptr,gameDat);
        chkBrd(topCrd,names,plyr2Ptr,gameDat);
        //Display mini board pattern on screen
        miniBrd(plyr1Ptr);
        miniBrd(plyr2Ptr);
        //Remove called card from deck
        deck.cards.erase(deck.cards.begin(),deck.cards.begin()+1);
        //Check if players have a full board
        if(plyr1Ptr->plyrTkn==16) {
            plyr1Ptr->win=true;
            gameDat.numWins++;
            cout<<plyr1Ptr->name<<": LOTERÍA!\n\n";
        }
        if(plyr2Ptr->plyrTkn==16) {
            plyr2Ptr->win=true;
            gameDat.numWins++;
            cout<<plyr2Ptr->name<<": LOTERÍA!\n\n";
        }
    }
}

cout<<"----------------------GAME OVER----------------------------"<<endl;
//Write to binary files and display contents from binary files
wrt2Bin(plyr1,dataFile);
wrt2Bin(plyr2,dataFile);
dispBin(plyr1,dataFile);
dispBin(plyr2,dataFile);
dispDat(gameDat);

//Clean up the dynamic stuff
delete []riddles;
riddles=nullptr;
delete []names;
names=nullptr;
delete []images;
images=nullptr;
delete []cards;
cards=nullptr;
delete board1;
board1=nullptr;
delete board2;
board1=nullptr;
```

```
   //Exit the code
   return 0;
}

Image crtImg(string name,string riddle) {
   Image img;  //Create image using name and riddle
   img.name=name;
   img.riddle=riddle;
   return img;
}

Image* crtImgs(string* names,string* riddles) {
   Image *imgs=new Image[NUMELMS]; //Dynamically allocate & fill array of images
   for(int i=0;i<NUMELMS;i++) {
      *(imgs+i)=crtImg(names[i],riddles[i]);
   }
   return imgs;
}

Card crtCrd(Image img,int num) {
   Card card;  //Create card using image and riddle
   card.img=img;
   card.num=num+1;
   return card;
}

Card* crtCrds(Image *imgs) {
   Card *cards=new Card[NUMELMS]; //Dynamically allocate & fill an array of cards
   for(int i=0;i<NUMELMS;i++) {
      *(cards+i)=crtCrd(*(imgs+i),i);
   }
   return cards;
}

void crtDck(Deck &d,Card *cards) {
   d.maxSize=NUMELMS;      //Max size of deck depends on number of elements
   for(int i=0;i<d.maxSize;i++) {
      Card card=*(cards+i);
      d.cards.push_back(card);    //Add card to vector
   }
}

void shufDck(Deck &d) {
   Deck shufDck;
   int rndIndx;
   int option;
```

```
   do {
      cout<<"Press 0 to Shuffle The Deck of Cards: ";
      cin>>option;
   } while(option!=0);
   cout<<"Deck Shuffled!"<<endl;
   cout<<"------------------------------------------------------------"<<endl;

   while(!d.cards.empty()) {
      srand(time(NULL));
      rndIndx=rand()%d.cards.size();  //generate random index
      shufDck.cards.push_back(d.cards[rndIndx]);  //push to card at random index
                                       //to shuffled deck
      d.cards.erase(d.cards.begin()+rndIndx);    //remove card from original deck
   }

   d=shufDck;     //deck is passed by reference
}

vector<int> rndCrds() {
   vector<int> crdNums(NUMELMS);   //vector to hold card numbers

   iota(crdNums.begin(),crdNums.end(),0);  //fill the range with values 0-54
   random_device rd;   //uniform random number generator
   mt19937 g(rd());    //needed for the shuffle function
   shuffle(crdNums.begin(),crdNums.end(),g);   //randomly rearrange the elements in the range

   return crdNums;
}

Board* newBrd(Card *cards,int num) {
   int index;
   Board *b=new Board;
   vector<int> v;     //vector of random card numbers

   index=0;
   b->brdNum=num;
   v=rndCrds();    //random card numbers to be used to make a random board
   for(int r=0;r<ROWS;r++) {
      for(int c=0;c<COLS;c++) {
         //De-reference pointer to get card to be placed in board
         b->board[r][c]=*(cards+v[index]);
         //Set the found flag of the card on the board to false
         b->board[r][c].found=false;
         index++;   //Index 0-15
      }
   }
   return b;
}
```

```
void setPtrn(Game &gameDat) {
   int option;
   cout<<"------------------------------------------------------------"<<endl;
   cout<<"SET THE WINNING PATTERN\n";
   cout<<"Options: #1=Full, #2=Row, #3=Column, #4=Diagonals\n\n";
   do{
      cout<<"Enter Pattern #: ";
      cin>>option;
   } while(option!=1&&option!=2&&option!=3&&option!=4);
   gameDat.pattern=option;
   cout<<"Players Must Match One of The Following Patterns to Win:\n";
   switch(option) {
      case FULL:
       cout<<word(FULL)<<endl;
       for(int r=0;r<ROWS;r++) {
          for(int c=0;c<COLS;c++) {
             cout<<"| * |";
          }
          cout<<endl;
       }
       break;
      case ROW:
       for(int i=0;i<4;i++) {
          cout<<word(ROW)<<i+1<<endl;
         for(int r=0;r<ROWS;r++) {
          for(int c=0;c<COLS;c++) {
             if(r==i) cout<<"| * |";
             else     cout<<"|   |";
          }
          cout<<endl;
         }
       }
       break;
      case COL:
       for(int i=0;i<4;i++) {
          cout<<word(COL)<<i+1<<endl;
         for(int r=0;r<ROWS;r++) {
          for(int c=0;c<COLS;c++) {
             if(c==i) cout<<"| * |";
             else     cout<<"|   |";
          }
          cout<<endl;
         }
       }
       break;
      case DIAGS:
       cout<<word(DIAGS)<<"1"<<endl;
```

```
        for(int i=0;i<4;i++) {
           for(int j=0;j<4;j++) {
              if(i==j) cout<<"| * |";
              else     cout<<"|   |";
           }
           cout<<endl;
        }
        cout<<endl;

        cout<<word(DIAGS)<<"2"<<endl;
        for(int i=0;i<4;i++) {
           for(int j=0;j<4;j++) {
              if(i+j==3){
                 cout<<"| * |";
              } else {
                 cout<<"|   |";
              }
           }
           cout<<endl;
        }
        break;
     }
   cout<<endl;
}

void setTkns(Game &gameDat) {
   int pattern=gameDat.pattern;
   int numTkns,
      minTk=2*ROWS,             //Amount tokens for non-full pattern
      maxTk=2*ROWS*COLS-ROWS,
      minTkFu=ROWS*COLS,        //Amount tokens for full pattern
      maxTkFu=2*ROWS*COLS;
   cout<<"------------------------------------------------------------"<<endl;
   cout<<"SET THE NUMBER OF TOKENS\nTokens are Shared Among Players.\n";
   do {
      if(pattern!=1) {
         cout<<"Enter Number of Tokens("<<minTk<<" to "<<maxTk<<"): ";
         cin>>numTkns;
      } else {
         cout<<"Enter Number of Tokens("<<minTkFu<<" to "<<maxTkFu<<"): ";
         cin>>numTkns;
      }

}while(((numTkns<minTk||numTkns>maxTk)&&pattern!=1)||((numTkns<minTkFu||numTkns>maxTkFu)
&&pattern==1));
   gameDat.tkns=numTkns;      //Set number of tokens for game data
   cout<<"------------------------------------------------------------"<<endl;
}
```

```
void chkBrd(Card card,string *names,Player *p,Game &gameDat) {
   bool ez=ezRddle(card.img.riddle,names);

   //If riddle is easy, use card name to search board
   if(ez) {
      for(int r=0;r<ROWS;r++) {
         for(int c=0;c<COLS;c++) {
            if(p->board->board[r][c].img.name==card.img.name) {
               cout<<p->name<<" Found a Match At "<<r+1<<", "<<c+1<<"."<<endl;
               p->board->board[r][c].found=true;   //set found to true
               p->plyrTkn++;   //Increment number of tokens used by player
               gameDat.tkns--;    //Decrement number of tokens available for game
            }
         }
      }
   //If riddle not easy, use riddle to search the board
   } else {
      for(int r=0;r<ROWS;r++) {
         for(int c=0;c<COLS;c++) {
            if(p->board->board[r][c].img.riddle==card.img.riddle) {
               cout<<p->name<<" Found a Match At "<<r+1<<", "<<c+1<<"."<<endl;
               p->board->board[r][c].found=true;   //set found to true
               p->plyrTkn++;   //Increment number of tokens used by player
               gameDat.tkns--;    //Decrement number of tokens available for game
            }
         }
      }
   }
   cout<<endl;
}

bool ezRddle(string riddle,const string *names) {
   string currStr;
   for(int i=0;i<NUMELMS;i++) {       //Go through names array
      currStr=(*(names+i)).substr(0,(*(names+i)).length()); //bug
      if(match(riddle,currStr)) { // Search the riddle for match of name
         return true;
      }
   }
   return false;      //If not match found, return true
}

bool match(string riddle,string name) {
   char *ridStr=strToC(riddle);       //return char * to cstr copy of riddle
   const char *nameStr=name.c_str();
   char *strPtr=nullptr; // To point to the found word in riddle
```

```
   strPtr=strstr(ridStr,nameStr); // Search the array for a matching substring
   if (strPtr!=nullptr) {  //Return true if name found in riddle
      return true;
   } else {
      return false;
   }
   //De-allocate memory
   delete []ridStr;
   ridStr=nullptr;
   delete []nameStr;
   ridStr=nullptr;
   delete []strPtr;
   ridStr=nullptr;
}

char* strToC(const string str) {    //c_str() returns const char *, needed char *
   char *cStr=new char[str.length()+1];

   for(int index=0;index<str.length();index++){
     cStr[index]=str[index];             //Copy string to cstring
   }
   cStr[str.length()]='\0';            //Append null character to cstring

   return cStr;
}

bool openFil(fstream &file,string name) {
   file.open(name,ios::in);    //Open file for input

   if(file.fail()) {
      return false;
   } else
      return true;
}

string* filToAr(fstream &file) {
   string *items=new string[NUMELMS];
   string line;
   for(int i=0;i<NUMELMS;i++) {
      getline(file,line,'\n');    //Get a line from text file
      *(items+i)=line;          //Add line to array of strings
   }
   return items;
}

ofstream plyrFil(Player* p) {
   ofstream file;
   string filName;
```

```
   filName=p->name+"Board.txt";
   // Open the file in output mode.
   file.open(filName,ios::out);
   file<<"Player Name: "<<p->name<<endl;
   brd2Fil(p->board,file);
   file.close();

   return file;
}

void prntCrd(const Card c) {
   //Print out top border of the card
   for(int i=0;i<14;i++) {
      cout<<"_";
   }
   //Print out starting where an image would be (represented by -)
   cout<<endl<<" "<<setw(14);
   cout<<setw(3)<<right<<c.num<<endl;
   for(int i=0;i<5;i++) {
      cout<<setw(4)<<left<<" ";
      cout<<"------"<<endl;
      if(i==5) cout<<endl;
   }
   //Print out the name
   for(int i=0;i<c.img.name.length();i++) {
      if(i>=0 && i<=c.img.name.length()) {
         cout<<c.img.name[i];
      }
   }cout<<endl;
   //Print out bottom border of the card
   for(int i=0;i<14;i++) {
      cout<<"_";
   }cout<<endl;
   cout<<endl<<"Riddle: "<<c.img.riddle<<endl<<endl;
}

void prntDck(const Deck& d) {
   cout<<"Printing shuffled deck..."<<endl;
   for(int i=0;i<NUMELMS;i++) {
      prntCrd(d.cards[i]);
   }cout<<endl;
   cout<<"Done printing shuffled deck.\n"<<endl;
}

void miniBrd(const Player *p) {
   cout<<setw(10)<<left;
   cout<<p->name<<" ";
```

```
   cout<<setw(5)<<right<<p->plyrTkn<<"/"<<ROWS*COLS<<endl;
   for(int r=0;r<ROWS;r++) {
      for(int c=0;c<COLS;c++) {
         if(p->board->board[r][c].found==true) {
            cout<<"| * |";
         }
         else {
            cout<<"|   |";
         }
      }
      cout<<endl;
   }
   cout<<endl;
}

void prntPlyr(Player *p) {
   cout<<"\nPlayer Name: "<<p->name<<endl;
   prntBrd(p->board);
}

void prntBrd(const Board *b) {
   int brdWdth=60;     //Set width to 60 variable for width

   cout<<endl;
   cout<<"LOTERIA"<<setw(brdWdth-8)<<"BOARD "<<b->brdNum<<endl;
   for(int j=0;j<4;j++) {
      //Print out top border of board
      for(int i=0;i<brdWdth;i++) cout<<"_";
      cout<<endl;
      //Print first row containing card numbers of 4 cards in top row
      for(int i=0;i<4;i++) {
         cout<<setw(brdWdth/4-1)<<left<<b->board[j][i].num
            <<setw(1)<<right<<"|";
      }
      cout<<endl;

      //Print out rows 2-7 containing stars in place of images
      for(int n=0;n<4;n++) {
         for(int i=0;i<4;i++) {
            cout<<setw(4)<<left<<" "
            <<setw(6)<<"------"
            <<setw(4)<<left<<" "
            <<"|";
         } cout<<endl;
      }

      //Print out blank row 9
      for(int i=0;i<4;i++) {
```

```
            cout<<setw(15)<<right<<"|";
        }
        cout<<endl;
    }
    //Print out bottom border of board
        for(int i=0;i<brdWdth;i++) cout<<"_";
        cout<<endl;
    //Print the card names and numbers underneath the board
        for(int r=0;r<ROWS;r++) {
            for(int c=0;c<COLS;c++) {
                cout<<b->board[r][c].num<<" ";
                cout<<b->board[r][c].img.name<<endl;
            }
        }
    cout<<endl;
}

void brd2Fil(const Board *b,ofstream &file) {
    int brdWdth=60;         //Set width to 60 variable for width
    file<<endl;
    file<<"LOTERIA"<<setw(brdWdth-8)<<"BOARD "<<b->brdNum<<endl;
    for(int j=0;j<4;j++) {
        //Print out top border of board
        for(int i=0;i<brdWdth;i++) file<<"_";
        file<<endl;
        //Print first row containing card numbers of 4 cards in top row
        for(int i=0;i<4;i++) {
            file<<setw(brdWdth/4-1)<<left<<b->board[j][i].num
                <<setw(1)<<right<<"|";
        }
        file<<endl;
        //Print out rows 2-7 containing stars in place of images
        for(int n=0;n<6;n++) {  //5 rows
            for(int i=0;i<4;i++) {
                file<<setw(4)<<left<<" "
                <<setw(6)<<"------"
                <<setw(4)<<left<<" "
                <<"|";
            } file<<endl;
        }
        //Print out blank row 9
        for(int i=0;i<4;i++) {
            file<<setw(15)<<right<<"|";
        }
        file<<endl;
    }
    //Print out bottom border of board
        for(int i=0;i<brdWdth;i++) file<<"_";
```

```
      file<<endl;

   //Goal is to print the name on the board
   //Temporarily print the card names and numbers underneath the board
      for(int r=0;r<ROWS;r++) {
         for(int c=0;c<COLS;c++) {
            file<<b->board[r][c].num<<" ";
            file<<b->board[r][c].img.name<<endl;
         }
      }
   file<<endl;
}

void wrt2Bin(Player p,fstream &file) {
   // Open the file for binary output.
   fstream plyrDat(p.name+".dat",ios::out|ios::binary);
   //Wite contents of structure to binary file
   cout<<"Saving "<<p.name<<" Game Data..."<<endl;
   plyrDat.write(reinterpret_cast<char *>(&p),sizeof(p));
   plyrDat.close();   // Close the file.
}

void dispBin(Player p,fstream &file) {
   // Open the file for binary input.
   fstream plyrDat(p.name+".dat",ios::in|ios::binary);
   // Now read and display the records.
   cout<<"Displaying "<<p.name<<" Game Data..."<<endl;
   plyrDat.read(reinterpret_cast<char *>(&p),sizeof(p));
   while(!plyrDat.eof()){
         cout<< "Name: ";
         cout<< p.name << endl;
         cout<< "Number of Matches Found This Round: ";
         cout<< p.plyrTkn << endl;
         plyrDat.read(reinterpret_cast<char *>(&p),
               sizeof(p));
         cout<<endl;
   }
   plyrDat.close();    // Close the file.
}

void dispDat(Game game) {
   cout<<"Displaying Game Data..."<<endl;
   cout<<"Pattern: "<<word(game.pattern)<<endl;
   cout<<"Number of Rounds: "<<game.numRnds<<endl;
   cout<<"Number of Tokens Remaining: "<<game.tkns<<endl;
   cout<<"Number of Winners: "<<game.numWins<<endl;
   cout<<endl;
}
```

```
string word(int num) {
   string word;
   if(num==1)      word="Full ";
   else if(num==2) word="Row ";
   else if(num==3) word="Column ";
   else if(num==4) word="Diagonal ";
   return word;
}
```