PROJECT 2

< LOTERÍA >

Name: Andrew Guzman

Date: 12/17/2022

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 game comes with a set of 20 boards each with a randomly generated 4x4 grid of cards. There can be 2 to 20 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). This program includes winning only for a full pattern. Players use tokens such as dried beans, small rocks, or bottle caps to mark matching cards found on their board.

To start the game, the caller shuffles the deck and pulls cards from the deck one-by-one. The caller pulls a card and calls out the name on the card. The players then search for the matching card on their board. 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.

DESCRIPTION

My general approach for transforming project 1 to an object-oriented program started with identifying the structs that would be turned into classes such as Card, Deck, Player, etc. Then, I identified the functions in project 1 that operated on individual structs which would then become member functions in the classes created from the structs. My program required additional classes such as BrdCard and BrdDeck that were derived from the original Card and Deck classes. Inheritance and polymorphism allowed me to create new classes based off other classes with additional data and additional functions to operate on the data.

Version 1 Card class:

I started by using functions to read the names of the cards in from a file and save the names to memory. Then I created an array of Card objects using a double pointer since the Card class uses a constructor to create new instances of the Card class.

Version 2 Deck class:

I took the functions from v1 that read in the card names as well as the array of Cards and encapsulated them into the Deck class. I added member variables and functions to operate on the Deck class and include functionalities such as displaying, shuffling, picking a card from the deck, and checking if the deck is empty.

Version 3 BrdCard class:

After version 2, the next class I was originally going to move on to implementing was the Board class which would be represented by a 2D array of Card objects. Then I realized that the cards used for the board would be different than the cards used for the deck. So, I decided to create the BrdCard class to inherit from the Card class. The BrdCard class would have additional member variables to hold the values of the x and y index of the card in the 2D array and a flag to tell whether that card has been found in the board. Additionally, the class includes getter and setter functions.

Version 4 BrdDeck class:

Since I concluded that a card and a board card were different, that meant that I needed to create another class for a deck of board cards. The Deck and BrdDeck class were almost identical, the only difference between them was the types of Card and BrdCard used in each class. In version 9 I implement a class template with the Deck class which eliminates all the repetitive code in BrdDeck class.

Version 5 Board class:

The Board class includes a constructor that creates the Board using a Deck of BrdCard objects. The Board has variables to tell how many rows and cols a board will have, the number of the board, and the board which is represented by a 2D array of BrdCard objects. Member functions allow me to set and get member variables and display the board to a file and screen.

Version 6 Player class:

The Player class was a simple class that includes variables to store the name, number of tokens used by a player, and a pointer to a Board object. Member functions to set and get member variables as well as print the Player object's board.

Version 7 AbsPlayer class:

The game of Loteria has different "types" of players, players who use boards and one player who is the caller who doesn't use a board, shuffles the cards, and picks cards from the deck. I was going to implement the Caller class in this version, but I decided to implement an abstract class AbsPlayer. This required that I go backwards for a bit to create this class to serve as a base class to inherit from for the Player class. The AbsPlayer class contains pure virtual functions that would then be defined in the Player class.

One way I can expand on this concept and create other "types of players" would be to create players who search their boards differently. The reason for this is that when Loteria is played in real life, you have players who behave differently when they are searching for matches on their board. This program includes players who find matches on the board by receiving the card name and searching for it on their board. In reality, there can also be players who use the card numbers and more advanced players who search their board by completely ignoring the card name and number. The advanced players instead listen to the riddles that the caller says and try to solve the riddles for the corresponding name to search for on their board.

Version 8 Caller class:

I implemented the Caller class which contained a deck of cards that the Caller uses for the game as well as the name of the Caller player. The Caller included functions to display the deck, display the card, shuffle the deck, pick a card, and check if the deck is empty. The Caller and the Deck have a "has-a" relationship(aggregation), which allowed me to avoid rewriting that functionality that was originally provided through the Deck class.

Version 9:

I implemented a class template for the Deck class which would allow me to go back and remove the repetitive code between the Deck and BrdDeck class. The Deck class template would generate code depending on the type used whether it was Card or BrdCard. The BrdDeck class then inherits from Deck<BrdCard> and inherit the public member functions of the Deck<T> class. The BrdDeck class would then be reduced to only require a destructor.

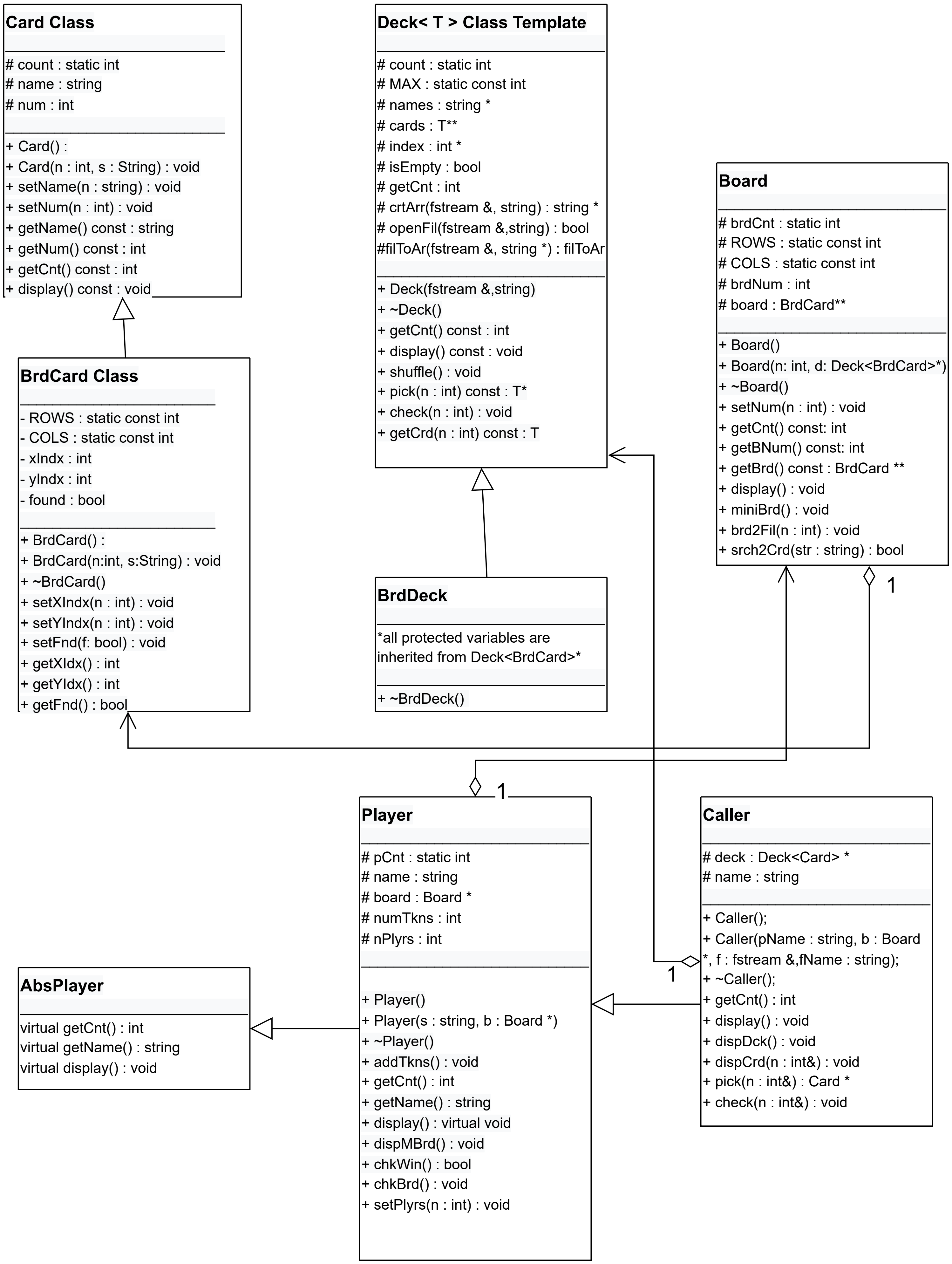
Version 10 and 11:

In version 10 I simply cleaned up my code and made sure that things were functioning properly. In version 11, I went ahead and implemented the game logic in main. The main from program 1 was very helpful in creating main for program 2, I only needed to make minor adjustments now that I was using classes vs structs. The final version includes variables such as an array of 20 Board objects (each game of Loteria comes with a set of 20 random boards), an array of Player objects, a Caller, and Deck used for the boards. Additionally, it includes variables such as a pointer to the current top card of the deck, the number of players, input, number of winners, number of rounds, a vector to hold the number of the boards that are used during a given game. I could expand on this program to encapsulate this data and functions into a LoteriaGame class.

CONCEPTS

Ch	Section	Topic	Location	Pts
13		Classes		
	1 to 3	Instance of a Class	main.cpp - line 51	4
	4	Private Data Members	BrdCard.h - line 13	4
	5	Specification vs. Implementation	Caller.h/Caller.cpp	4
	6	Inline	Card.h - line 21	4
	7, 8, 10	Constructors	BrdCard.cpp - line 12, 19	4
	9	Destructors	Board.cpp - line 46	4
	12	Arrays of Objects	main.cpp - line 48	4
	16	UML	included in write-up	4
14		More about Classes		
	1	Static	Card.h - line 15	5
	2	Friends	-	2
	4	Copy Constructors	-	5
	5	Operator Overloading	-	8
	7	Aggregation	Player.h - line 18	6
15		Inheritance		
	1	Protected members	Deck.h - line 14	6
	2 to 5	Base Class to Derived	BrdCard.h - line 12 (Card and BrdCard classes)	6
	6	Polymorphic associations	main.cpp - line 86 (Player and Caller classes)	6
	7	Abstract Classes	Player.h - line 14 (AbsPlayer class)	6
16		Advanced Classes		
	1	Exceptions	main.cpp - line 63	6
	2 to 4	Templates	Deck.h	6
	5	STL	main - line 42, 81	6
		Total		100

UML Class Diagrams


Card Class
# count : static int
# name : string
# num : int
+ Card() :
+ Card(n : int, s : String) : void
+ setName(n : string) : void
+ setNum(n : int) : void
+ getName() const : string
+ getNum() const : int
+ getCnt() const : int
+ display() const : void
Deck< T > Class Template
# count : static int
# MAX : static const int
# names : string *
# cards : T**
# index : int *
# isEmpty : bool
# getCnt : int
# crtArr(fstream &, string) : string *
# openFil(fstream &,string) : bool
#filToAr(fstream &, string *) : filToAr
+ Deck(fstream &,string)
+ ~Deck()
+ getCnt() const : int
+ display() const : void
+ shuffle() : void
+ pick(n : int) const : T*
+ check(n : int) : void
+ getCrd(n : int) const : T
Board
# brdCnt : static int
# ROWS : static const int
# COLS : static const int
# brdNum : int
# board : BrdCard**
+ Board()
+ Board(n: int, d: Deck<BrdCard>*)
+ ~Board()
+ setNum(n : int) : void
+ getCnt() const: int
+ getBNum() const: int
+ getBrd() const : BrdCard **
+ display() : void
+ miniBrd() : void
+ brd2Fil(n : int) : void
+ srch2Crd(str : string) : bool
BrdCard Class
- ROWS : static const int
- COLS : static const int
- xIndx : int
- yIndx : int
- found : bool
+ BrdCard() :
+ BrdCard(n:int, s:String) : void
+ ~BrdCard()
+ setXIndx(n : int) : void
+ setYIndx(n : int) : void
+ setFnd(f: bool) : void
+ getXIdx() : int
+ getYIdx() : int
+ getFnd() : bool
BrdDeck
all protected variables are inherited from Deck<BrdCard>
+ ~BrdDeck()
1
1
Player
# pCnt : static int
# name : string
# board : Board *
# numTkns : int
# nPlyrs : int
+ Player()
+ Player(s : string, b : Board *)
+ ~Player()
+ addTkns() : void
+ getCnt() : int
+ getName() : string
+ display() : virtual void
+ dispMBrd() : void
+ chkWin() : bool
+ chkBrd() : void
+ setPlyrs(n : int) : void
Caller
# deck : Deck<Card> *
# name : string
+ Caller();
+ Caller(pName : string, b : Board *, f : fstream &,fName : string);
+ ~Caller();
+ getCnt() : int
+ display() : void
+ dispDck() : void
+ dispCrd(n : int&) : void
+ pick(n : int&) : Card *
+ check(n : int&) : void
1
AbsPlayer
virtual getCnt() : int
virtual getName() : string
virtual display() : void