

COMP 3000 – Assignment 6
Due 11:59 pm – Friday, Dec 6, 2019
Submit Assign06.zip via Canvas

Program: Assign06

For Assignment 6, we will be using the deck class to create a second program which performs a magic trick. The files for the Deck class and Card struct, as well as Card's overloaded operators have been provided for you in the files Deck.h and Deck.cpp. To use the files, place them in the same folder as your Assign06.cpp file and use **#include "Deck.h"** in your program to access its class, struct, and overloaded operators.

Changes to Deck class:

Because the Deck class has been provided, you do not need to make changes to it. It is functionally similar to the Deck class from assignment 5, but the "private" members have been changed to "protected" to allow for overloading.

The Sim Sala Bim (or 21 cards) card trick:

We are using this trick because it always works so long as the magician follows the steps.

The steps are:

- The magician begins by dealing a deck of 21 cards and shuffling them.
- The 21 cards are dealt into three decks in the order: left, middle, right, left, middle, right, etc... until all three decks contain seven cards.
- The magician asks the participant to memorize one card from the three decks.
- Once the participant has selected their card, the magician stacks the three decks together, making sure that the deck containing the selected card is in the middle of the other two decks.
- **Without shuffling**, the magician repeats dealing the cards into three decks using the same method as last time.
- The magician asks the participant to identify which deck contains their card.
- As before, the magician stacks all three decks together, ensuring the deck containing the participant's card is in the middle.
- The magician repeats the process of dealing cards, having the participant select their card, and stacking the decks **at least two more times**.
- The magician pulls the 11th card in the deck of 21 (aka the 4th card of the middle deck), and shows it to the participant as "their card."

Gifs of the trick can be seen here: <https://www.wikihow.com/Do-a-21-Card-Card-Trick>

Because we need to stack the decks together after each round of dealing, we will use a linked list of decks to place our decks of 7 in order. However, to make a linked list, we need to add a pointer to our Deck class. So, we will create a class called **MagicDeck** that inherits the properties of **Deck**. **We have changed the private members of Deck to protected members to allow for inheritance.** Because the new class inherits Deck, it will have all the functions and members of Deck except its constructors.

Tip:

Because we are using a linked list, we will frequently be using pointers to point at a deck in our list of decks. **To use the member functions of a deck tied to a pointer, we must use this format:**

pointer->function() instead of variable.function()

e.g. headPointer->dealCard() instead of firstDeck.dealCard(), assuming headPointer points to firstDeck.

New child class MagicDeck:

First, we will create a new class called MagicDeck. **This new class should inherit the Deck class.** This will give it access to deck's members and functions.

A MagicDeck is a Deck that includes a pointer to a following MagicDeck to allow for linked list creation. Because constructors are not inherited, we will need to create constructors for the new class. We will also create a getter and setter for our pointer variable (called "link" in these instructions), and redefine outputDeck() so that it changes the header to "Deck #:", where # is an integer set to the function, instead of the user's name.

Private member:

MagicDeck* link: a pointer to the next magic deck.

Public members:

MagicDeck():	Create a deck of size 0 and with the pointer set to nullptr.
MagicDeck(int deckSize):	Create a deck of size deckSize with the pointer set to nullptr.
MagicDeck(MagicDeck* aDeck):	Create a deck of size 0 with a pointer to aDeck.
MagicDeck(int deckSize, MagicDeck* aDeck):	Create a deck of size deckSize with a pointer to aDeck.
void setLink(MagicDeck* theDeck):	Set the "link" pointer to theDeck.
MagicDeck* getLink():	Return the value of the "link" pointer.
void outputDeck(int deckPos):	Output the deck with the head "Deck #:" where # is the value of deckPos

Tips:

The * character in data types is used to denote a pointer, rather than a value. If you are getting errors about incorrect data types, make sure you are using the * correctly.

When creating a constructor for an inherited class, the constructor of the parent class must be called in the initialization section of the child class's constructor. For instance:

```
class Parent{
    Parent() {}
}

class Child : public Parent{
    Child(): Parent(), ... {} Note: ... represents setting Child's other variables
}
```

If the deck does not have a deck following it, the nullptr constant should be assigned to link

Much of the code from Deck's outputDeck() can be used in MagicDeck's outputDeck().

New data type:

The datatype "DeckPtr" should be created to represent a pointer to a MagicDeck. This is done using:
typedef MagicDeck* DeckPtr;

This will help make it clearer when we are using a pointer to a MagicDeck or not.

Make sure your typedef comes after you declare the MagicDeck class.

Magician Class:

A new class called Magician will take the role of the magician and perform the steps of the magic trick.

Private Variables:

DeckPtr head: a pointer that points to the first deck in our linked list of decks.

Public functions:

Magician():	Default constructor. Sets head to nullptr.
DeckPtr getHead():	Returns the value of the head pointer.
void greetUser():	Greets the user and asks them to memorize a card.
void startTrick():	creates a deck of the initial 21 cards, sets up the linked list, then calls dealDecks to deal out the 21 cards. Pseudocode below.
void dealDecks(DeckPtr firstDeck):	Deals all cards from linked list of one or more MagicDecks into three decks of equal size. Pseudocode on next page.
void outputDecks():	Outputs a list of one or more of MagicDecks using outputDeck().
int askDeck():	Asks the user which deck contains their card and returns an integer indicating the chosen deck (1, 2, or 3).
void stackDecks(int chosenDeck):	Orders the three decks so that the deck indicated in chosenDeck is in the middle of the linked list. Pseudocode on next page.
void findCard():	Deals the 4 th card of the middle deck and displays it to the user.
bool offerReplay():	Asks the user if they would like to rerun the program and returns true or false based on their response.

Tips:

Input in the program should be validated, as in the last program. Specifically, the user should only be able to input 1, 2, or 3, when asked which deck contains their card and only “yes” or “no” (in any case) when asked if they would like to rerun the program.

To traverse a linked list, set a variable equal to head, then when you want to move on to the next deck, set the same variable equal to *variable* -> `getLink()`. This sets the variable keeping track of the current deck to the deck that the current deck is pointing to.

If you want to traverse the entire list, use `(DeckPtr p = head; p != nullptr; p = p->getLink())`. This starts a DeckPtr at the head and changes it to the next MagicDeck when the loop iterates once, stopping the loop when the DeckPtr is null.

If you want to output a specific card in a deck, deal off the cards before the card you want first without storing them in a variable. Then, you can deal the card you want. For instance, to deal the 5th card, deal 4, then deal 1 more and store that one.

The code for offerReplay() for Magician will be similar to the code for offerReplay() for GameEngine.

startTrick() Pseudocode:

```

Create a pointer to an initial MagicDeck of 21 cards,
Shuffle the deck
Set head equal to a new MagicDeck.
Set the link of the head deck to a new MagicDeck.
Create a pointer and set it equal to the link you just set.
Repeat until all three decks are linked.
Set the link of the last deck to nullptr.
Call dealDecks() and send the pointer to the deck of 21 cards.

```

dealDecks():

To deal cards, we will need to keep track of which MagicDeck we are dealing to and what MagicDeck we are dealing from. The idea is to change the deck we are dealing to each time we deal a card, looping back to the beginning when we reach the end of our linked list, while only changing the deck we are dealing from after all original cards have been dealt from it and stopping once we reach the end of the list.

Pseudocode:

Create a DeckPtr and set it equal to head. This is the MagicDeck we are dealing to.
Get the size of one MagicDeck.

For each deck (iterate through list of MagicDecks using a DeckPtr; this DeckPtr tracks the MagicDeck we are dealing from)

 For each card in a deck (iterate through all cards using an int)

 Deal a card from the for loop's MagicDeck and add it to the other MagicDeck.

 If the link of the MagicDeck we are dealing to is nullptr (aka we are at the end of our list)

 Set the MagicDeck we are dealing to equal to head.

 Else

 Set the MagicDeck we are dealing to equal to the next deck using getLink().

Tips:

We must get the size of the MagicDeck before dealing or adding cards because we only want to remove the cards the deck held before adding to it. Otherwise, the size will change each time the loop runs.

Since we are assuming all three MagicDecks are the same size, we only need to find the size once.

=====

StackDecks(int chosenDeck) Pseudocode:

Create a DeckPtr and set it equal to head. This DeckPtr now points to the first MagicDeck.

Create a DeckPtr and set it equal to the pointer stored in the link of head. This DeckPtr now points to the 2nd MagicDeck.

Create a DeckPtr and set it equal to the pointer stored in the link of the 2nd MagicDeck. This DeckPtr now points to the 3rd MagicDeck.

If chosenDeck == 1,

 Set the pointer on the first MagicDeck to point to the third MagicDeck

 Set the pointer on the second MagicDeck to point to the first MagicDeck.

 Set the head to point to the second MagicDeck.

If chosenDeck == 3,

 Set the pointer on the third MagicDeck to point to the second MagicDeck.

 Set the pointer on the first MagicDeck to set to the third MagicDeck.

 Set the pointer of the second MagicDeck to nullptr.

Tips:

Because the second deck is already in the middle, we do not need to change the decks' order if the user chooses deck 2.

We need to change the head in the first case because the second MagicDeck is now on the top of the list and has become the first MagicDeck. We need to change the "head" variable specifically to make sure the change carries across all uses of the list.

If a new MagicDeck is in the last position, make sure you set its pointer to nullptr.

Main():

Create a variable to store if the user wants to run the program again or not.

Create an instance of Magician.

Greet the user.

While the user wants to rerun the program

 Start the trick

 Output the decks.

 Repeat 3x:

 Ask which deck holds the user's card and store the result in a variable.

 Stack the decks based on the user's response.

 Deal the decks, sending the head of your linked list of MagicDecks.

 Output the Decks.

Find the user's card.

Ask the user if they would like to run the program again.

=====

Sample Output (User's card is the ace of spades):

```
Want to see a magic trick?
Pick a card from one of the three following decks and memorize it.

Deck 1:
5 of spades
9 of spades
3 of diamonds
8 of clubs
6 of spades
9 of hearts
6 of diamonds

Deck 2:
2 of diamonds
5 of hearts
J of diamonds
3 of clubs
8 of hearts
4 of clubs
10 of spades

Deck 3:
A of spades
K of hearts
7 of clubs
Q of clubs
4 of hearts
2 of spades
7 of diamonds

What deck is the card in?
(Enter 1, 2, or 3):
```

```
What deck is the card in?
(Enter 1, 2, or 3): 1

Deck 1:
8 of clubs
Q of clubs
3 of clubs
8 of hearts
9 of hearts
5 of hearts
3 of diamonds

Deck 2:
2 of spades
4 of clubs
5 of spades
A of spades
2 of diamonds
6 of spades
4 of hearts

Deck 3:
9 of spades
K of hearts
7 of clubs
J of diamonds
6 of diamonds
7 of diamonds
10 of spades

is this your card?
A of spades

Try again? (yes/no)
no
```

Submit via Canvas:

Assign06.zip

Compressed Visual Studio Solution folder