Project

CSI2372A – Fall 2020

University of Ottawa

/26

Du Online on Friday December 4 at midnight.

(In teams of two and only one submission by group is required)

Card Game:

In this project, you are asked to program a card game in C++ language. The game is played with a deck of cards with comical illustrations of eight different types of beans (some more scarce than others), which two players place cards in chains, trade the cards and sell the chains in order to raise money.

The player obtains cards of all different types *randomly* from the deck, and so must engage in trade with the other player to be successful.

The cards have 8 different faces corresponding to different types of Beans (see Table below). The goal of the game is to chain-up the cards of same bean to gain coins. The player with the most coins at the end wins. The chains for the cards are formed by each player for all to see and there are a maximum of either two or three chains at any point per player. Each chain can only be formed with a single type of Bean.

Each player is dealt a hand of five cards to start and the remaining cards form a draw deck. The rule is that cards in a player's hand need to be kept sorted. Cards will be placed on a discard pile during the game.

Cards in the hand are kept hidden. Cards in trading areas and chains are visible to all players. The discard pile is face up, but only the top card is visible.

The game proceeds with the players taking turns. During their turn, each player does the following:

1. If the other player has left cards in the trade area from the previous turn (in Step 5), the player may add these cards to his/her chains or discard them.
2. The player then plays the topmost card from his/her hand. The card must be added to a chain with the same beans. If the player has currently no such chain on the table, an old chain on the table will have to be tied and sold, and a new chain must be started. If the chain has not reached a sufficient length, a player may receive 0 coins.
3. The player has the option to repeat Step 2.

4. Then, the player has the option of discarding one arbitrary card (in any order) from his/her hand on the discard pile face up.
5. The player draws three cards from the draw deck and places them in the trade area. Next, the player draws cards from the discard pile as long as the card matches one of the beans in the trade area. Once the top card does not match or the pile is empty, the player can either chain the cards or leave them in the trade area for the next player. As in Step 2, if the player has currently no such chain matching the bean of the card, an old chain on the table will have to be tied and sold, and then a new chain is started.
6. The turn ends with the player drawing always two cards from the deck and placing them at the back of his/her hand.

Whenever a player ties a chain and sells it (Steps 2,3 or 5), the player receives coins in trade for the chain.

A player can purchase the right to form a third chain for three coins. No more than three chains can be formed simultaneously by a player during the game.

The game ends when the deck becomes empty.

Chains of different length and different beans have different value as shown below:

		Value of Chains			
Beans (Card Name)	**Total in Deck**	1 Coin	2 Coins	3 Coins	4 Coins
Blue	20	4	6	8	10
Chili	18	3	6	8	9
Stink	16	3	5	7	8
Green	14	3	5	6	7
soy	12	2	4	6	7
black	10	2	4	5	6
Red	8	2	3	4	5
garden	6	-	2	3	-

C++ program description

Each component of the game is represented by its corresponding class:

`Card`, `Deck`, `DiscardPile`, `Chain`, `Table`, `TradeArea`, `Coins`, `Hand`, `Players`.

The classes `Deck`, `DiscardPile`, `Hand` and `Chain` are all containers holding cards.

- `Deck` will initially hold all the cards which will have to be shuffled to produce a randomized order, then players' hands are dealt and during game play players draw cards from the `Deck`. There is no insertion into the `Deck`. `Deck` can therefore usefully extend a `std::vector`.
- `DiscardPile` must support insertion and removal but not at random locations but all at the end. Again a `std::vector` will work fine but here we can use simple aggregation.
- `Hand` is well mapped by a queue since players have to keep their hand in order and the first card drawn is the first card played. The only derivation form this pattern is if players discard a card from the middle in Step 4 in the above description of a player's turn. Therefore, we can use a `std::list` to remove at an arbitrary location efficiently with a `std::queue` adapter.
- `Chain` is also a container and will have to grow and shrink as the game progresses. Again insertion is only to one end of the chain and a `std::vector` is fine (see below).

  A template class will have to created for `Chain` being parametric in the type of card. In this project, we will instantiate `Chain` for the corresponding bean card.
- `Card` will be an abstract base class and the eight different bean cards will be derived from it (inheritance). All containers will hold cards through the base type. However, standard containers do not work well with polymorphic types because they hold copies (slicing will occur).
- `TradeArea` class will have to hold cards openly and support random access insertion and removal.
- `CardFactory` will generate all cards, and so, we will explore the factory pattern. A factory ensures that there is only a single unit in the program that is responsible to create and delete cards. Other parts of the program will only use pointers to access the cards. Note that means, we will delete the copy constructor and assignment operator in `Card`.

You will be using:

- *exceptions*: with downcasts to distinguish between different bean cards, and also of `IllegalType` if a player attempts to place a card illegally into a chain (a chain that holds different beans).
- *standard algorithms*: such as `std::shuffle` at the beginning to ensure the cards in the deck are in a random order.
- *stream insertion* and *extraction operator* to save and reload the game from file (for pause functionality).

C++ program Implementation

You need to decide on class variables and the private and protected interface of the classes. Your mark will depend on a reasonable design and documentation in the code. Use *const* as much as possible, you can make any function or operator *const* as you see fit, even if it is not indicated in the prototype below. You can add others constructor and destructor to a class if needed.

`Card` class and its derived classes (4 marks)

Create the bean card and its derived classes:

- `Card` is the base class `and` will be abstract
- `Blue, Chili, Stink, Green, soy, black, Red` and `garden` are derived classes (from `Card`) and will have to be concrete classes.
  A bean card can be printed to console with its first character of its name.

`Card` class will have the following pure virtual functions:

- `virtual int getCardsPerCoin(int coins)` will implement in the derived classes the above table for how many cards are necessary to receive the corresponding number of coins.
- `virtual string getName()` returns the full name of the card (e.g., Blue).
- `virtual void print(ostream& out)` inserts the first character for the card into the output stream supplied as argument. If the first character is the same for two cards, use uppercase for the first one and lowercase for the second one (For example B for Blue and b for Black).
- and a global stream insertion operator for printing any objects of such a class which implements the *"Virtual Friend Function Idiom"* with the class hierarchy.

`Chain` class (2 marks)

The template Chain will have to be instantiated in the program by the concrete derived card classes, e.g., `Chain<Red>`. Note that in this example `Chain` will hold the `Red` cards by pointer in a `std::vector<Red*>`. So, you will need an abstract chain interface (`Chain_Base`) to be able to reference chains of any type from the `Player` class.

`Chain` will have the following functions:

- `Chain(istream&, const CardFactory*) is` a constructor which accepts an `istream` and reconstructs the chain from file when a game is resumed.
- `Chain<T>& operator+=(Card*)` adds a card to the `Chain`. If the run-time type does not match the template type of the chain and exception of type `IllegalType` needs to be raised.
- `int sell()` counts the number cards in the current chain and returns the number coins according to the function `Card::getCardsPerCoin`.
- and the insertion operator (friend) to print `Chain` on an `std::ostream`. The hand should print a start column with the full name of the bean, for example with four cards:

```
Red     R R R R
```

`Deck` class (2 marks)

`Deck` is simple derived class from `std::vector`.

`Deck` will have the following functions:

- `Deck(istream&, const CardFactory*)` is a constructor which accepts an `istream` and reconstructs the deck from file.
- `Card* draw()` returns and removes the top card from the deck.
- and the insertion operator (friend) to insert all the cards in the deck to an `std::ostream`.

`DiscardPile` class (2 marks)

The `DiscardPile` class holds cards in a `std::vector` and is similar to `Deck`.

`DiscardPile` will have the following functions:

- `DiscardPile(istream&, const CardFactory*)` is a constructor which accepts an `istream` and reconstructs the `DiscardPile` from file.
- `DiscardPile& operator+=(Card*)` discards the card to the pile.
- `Card* pickUp()` returns and removes the top card from the discard pile.
- `Card* top()` returns but does not remove the top card from the discard pile.
- `void print(std::ostream&)` to insert all the cards in the `DiscardPile` to an `std::ostream`.
- and the insertion operator (friend) to insert only the top card of the discard pile to an `std::ostream`.

`TradeArea` class (3 marks)

The `TradeArea` holds cards in a `std::list,` and will have the following functions:

- `TradeArea(istream&, const CardFactory*)` is a constructor which accepts an `istream` and reconstruct the `TradeArea` from file.
- `TradeArea& operator+=(Card*)` adds the card to the trade area but it does not check if it is legal to place the card.
- `bool legal(Card*)` returns true if the card can be legally added to the TradeArea, i.e., a card of the same bean is already in the TradeArea.
- `Card* trade(string)` removes a card of the corresponding bean name from the trade area.
- `int numCards()` returns the number of cards currently in the trade area.
- and the insertion operator (friend) to insert all the cards of the trade area to an `std::ostream`.

`Hand` class (2 marks)

`Hand` class will have the following functions:

- `Hand(istream&, const CardFactory*)` is a constructor which accepts an `istream` and reconstruct the `Hand` from file.
- `Hand& operator+=(Card*)` adds the card to the rear of the hand.
- `Card* play()` returns and removes the top card from the player's hand.
- `Card* top()` returns but does not remove the top card from the player's hand.
- `Card* operator[](int)` returns and removes the `Card` at a given index.
- and the insertion operator (friend) to print `Hand` on an `std::ostream`. The hand should print all the cards in order.

`Player` class (3 marks)

`Player` class will have the following functions:

- `Player(std::string&)` is a constructor that creates a Player with a given name.
- `Player(istream&, const CardFactory*)` is a constructor which accepts an `istream` and reconstruct the `Player` from file.
- `std:string getName()` get the name of the player.
- `int getNumCoins()` get the number of coins currently held by the player.
- `Player& operator+=(int)` add a number of coins
- `int getMaxNumChains()` returns either 2 or 3.
- `int getNumChains()` returns the number of non-zero chains
- `Chain& operator[](int i)` returns the chain at position i.
- `void buyThirdChain()` adds an empty third chain to the player for three coins. The functions reduces the coin count for the player by two. If the player does not have enough coins then an exception `NotEnoughCoins` is thrown.
- `void printHand(std::ostream&, bool)` prints the top card of the player's hand (with argument false) or all of the player's hand (with argument true) to the supplied ostream.
- and the insertion operator (friend) to print a `Player` to an `std::ostream`. The player's name, the number of coins in the player's possession and each of the chains (2 or 3, some possibly empty) should be printed. The player printout may look as follows:

```
Dave    3 coins
Red     R R R R
Blue    B
```

`Table` class (2 marks)

`Table` will manage all the game components. It will hold two objects of type `Player`, the `Deck` and the `DiscardPile`, as well as the `TradeArea`.
`Table` class will have the following functions:

- `Table(istream&, const CardFactory*)` is a constructor which accepts an `istream` and reconstruct the `Table` from file.
- `bool win(std::string&)` returns true when a player has won. The name of the player is returned by reference (in the argument). The winning condition is that all cards from the deck must have been picked up and then the player with the most coins wins.
- `void printHand(bool)` prints the top card of the player's hand (with argument false) or all of the player's hand (with argument true).
- and the insertion operator (friend) to print a `Table` to an `std::ostream`. The two players, the discard pile, the trading area should be printed. This is the top level print out. Note that a complete output with all cards for the pause functionality is printed with a separate function.

`CardFactory` class (2 marks)

The card factory serves as a factory for all the bean cards.
`CardFactory` class will have the following functions:

- constructor in which all the cards need to be created in the numbers needed for the game (see the above table).
- `static CardFactory* getFactory()` returns a pointer to the only instance of `CardFactory`.
- `Deck getDeck()` returns a deck with all 104 bean cards. Note that the 104 bean cards are always the same but their order in the deck needs to be different every time. Use `std::shuffle` to achieve this.

Also, ensure that no copies can be made of `CardFactory` and that there is at most one `CardFactory` object in your program.

Pseudo Code (4 marks for *main* loop)

The pseudo-code of the *main* loop (game loop) is as follows.
Setup:

- Input the names of 2 players. Initialize the Deck and draw 5 cards for the Hand of each Player; or
- Load paused game from file.

```
While there are still cards on the Deck
    if pause save game to file and exit
    For each Player
        Display Table
        Player draws top card from Deck
        If TradeArea is not empty
            Add bean cards from the TradeArea to chains or discard them.
        Play topmost card from Hand.
        If chain is ended, cards for chain are removed and player receives coin(s).
        If player decides to
            Play the now topmost card from Hand.
        If chain is ended, cards for chain are removed and player receives coin(s).
        If player decides to
            Show the player's full hand and player selects an arbitrary card
            Discard the arbitrary card from the player's hand and place it on the discard pile.
        Draw three cards from the deck and place cards in the trade area
        while top card of discard pile matches an existing card in the trade area
            draw the top-most card from the discard pile and place it in the trade area
        end
        for all cards in the trade area
            if player wants to chain the card
                chain the card.
            else
                card remains in trade area for the next player.
        end
        Draw two cards from Deck and add the cards to the player's hand (at the back).
    end
end
```