Programming Project 3
(Due Monday, February 09, 2004 at 11:59pm + 24 hour free extension)

ANY HELP (EXCEPT FROM COURSE STAFF) IS PROHIBITED
Individuals found guilty of violating above policy will be referred to V&E committee

You may use VC++ 6.0 (no .NET allowed) environment for this project. In that case, you will need to submit the whole workspace and the project. Submissions must be made in the right folder on \badar\common. You are also free to use suraj machine for implementation in which case you will submit all .cc and .h files zipped as usual.

In this project, you will be implementing three solitaire-like card games. Since these games are related to each other in a close manner, you will use inheritance during your implementation. In fact, you will use two separate inheritance relationships in this project. One is between different piles of cards and the other is between various games. You can find information on Solitaire games at http://www.goodsol.com but you are not required to go to that web site for this project.

Card games follow different rules. There are so many of them that the games with slightly different rules may sometimes use the same name. You must implement the rules we define here. Do not come back to us saying that the rules we have specified are incorrect. If they seem incorrect to you, assume that this is some different planet where a tomato means a monkey and where monkeys mean grades!

Here is what is common to most Solitaire games, at least the ones we consider in this project. We are actually providing you the class description that should make your job a lot easier. Some starter code is also being provided.

The Base Classes

Card:
A Card is one of the 52 units used to play card games. There are 4 suits (Hearts, Clubs, Spades, and Diamonds) and 13 ranks (A/1, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K) in each suit. From a programming perspective, this is a class which should hold all the data which can distinguish a Card from another. It should also provide methods “comparing” itself to another Card. This comparison is not necessarily the equality or inequality comparison. Other comparisons should also be possible, for example if two Cards have same suit or same rank. A Card may also provide a toString() method which simply returns a C++ string to facilitate printing this Card. A Card should only be printed when it is facing up. So, perhaps the toString() method should return a string “down” or something like that if the Card is not up.

To make life simple, you should use ints as the ranks. Use 11 for J, 12 for Q and 13 for K. Your Card class will also provide a method to return the rank of a specific Card. Similarly, you might need a method to return the suit of a Card.

---

1 This project has been derived from a previous programming project which was given to students taking a programming course offered by computer science department at Stanford University.
**Deck:**
A Deck is a collection of all 52 cards. The constructor of Deck should initialize the Deck to hold all 52 cards. Since the number of Cards in a Deck is fixed, we could use a static array to hold the Cards of a Deck. Alternately, a vector can also be used to hold the Cards in this Deck. The Deck should be able to shuffle itself when requested and should also provide a getNextCard() method which is self explanatory.

**Pile:**
A Pile is also a collection of Cards but the number of Cards in a Pile may vary during the play. A Game will contain several Piles. Different Games will have different restrictions on various Piles they may contain. It would not always be possible to add or remove a Card from a Pile. This will depend upon the current state of Piles and the type of Game.

Since there can be many types of Piles, this class should behave as the base class from which other Piles are able to derive. Perhaps, the best way to implement a Pile is to encapsulate a deque/vector of Cards in it (we will prefer deques!). In fact, the Cards are still the property of Deck and a Pile should only hold references to them. Since an STL container can only hold real variables (and not the references), it is best to store pointers to Cards in a Pile. Each Pile may only contain a maximum number of Cards which should be passed as an int to the Pile constructor.

During the Games we implement, we will be moving Cards around one by one from one Pile to another. Therefore, Piles should implement a method which tells whether or not we can remove the top card from the Pile. If the Pile is not empty, this is usually true unless some Pile overrides this behavior and removal is prohibited, either in general or to a specific destination. Similarly, a Pile should be able to tell whether it can receive the top Card from some other Pile.

One important property of a Pile is that it could either be circular or non-circular. The former means that the Cards wrap around. For example, some Piles will require starting from a particular rank but others will not. Other methods a Pile should implement include, but are not limited to, getting the top Card, adding a Card, removing a Card, and printing the Pile.

Another important property of a Pile is that it may be fanned or not. When the Pile is not fanned, you could only see the top Card of the file and the other Cards underneath are not visible. Of course, you couldn’t see a Card that is not facing up.

It would probably be a good idea to make the printPile() method public and the printTopCard() method protected. This is because a specific Pile (perhaps a subclass of Pile) will decide itself whether it is fanned or not.

**Game:**
The Game class implements a card game which can be played with one standard deck of cards. Initial dealing of cards (distributing on various piles) is the property of a specific Game which will be derived from this class. Perhaps, a dealCards() method may be defined in this class as pure virtual.

As we said, a Game will be a sequence of moves of a single Card from one Pile to another. Once you move a Card, the Game should redraw itself, thus requiring a
redrawBoard() method. This is the best place to have an interface with the graphics objects. You will write the redrawBoard() method but don’t worry about the graphics interface unless you want to build a graphics version of the Game. Even if you are making a graphics version, we suggest you finish the project with console output and then build a graphics interface.

At the end of each move, the Game should be asked if the player has won the Game. For all the Games we are implementing, there is only one way of winning the game: when the player is able to move ALL Cards to four foundation piles; these Piles will derive from the Pile base class. Thus, the goal in each Game we implement is to move all the Cards to the foundation piles.

The Game class should also implement the main interactive loop where input is taken from the user. This is the second place where graphics objects will interface. We will provide you starter code for using the console input but feel free to modify it in any way that pleases your aesthetic sense.

Other Helper Classes:
Perhaps you would want to have one SimpleInput class or something similar to that and one class which acts as the entry point into the program. We will provide starter code for the entry point in a file called solitaire.cpp

The Derived Classes
In addition to the base classes described above, you will have an inheritance hierarchy for the two classes: Game and Pile.

Classes derived from the Game class will each represent a unique Game and the classes derived from the Pile class will each represent a Pile with some special properties.

FoundationPile
As we already mentioned, there are four foundation piles in each Game and they may be encapsulated in the Game base class. The goal of each Game we write is to fill up the four FoundationPiles, each carrying 13 Cards. The FoundationPiles derive from the Pile class and may or may not be circular. If the FoundationPiles are non-circular, they must start from a specific Card and build up on that Card (usually the A of each suit). If they are circular, they may start at any Card location and building up on top of that. For example, {4 3 2 A K Q J 10 9 8 7 6 5} is a valid circular FoundationPile.

The Games we have chosen restrict you to start the FoundationPiles all with the same Card even if they are circular. In the above example, if a Game allows circular FoundationPiles, all of them must start with 5. This can easy be handled by having a data member in the FoundationPile which holds a static baseCard.

---

2 Definition from http://www.goodsol.com/pgshelp/foundation.htm: Piles of cards where the object is to build upon a base card, usually up in a sequence. In most games, the foundations are built up by suit, although this can vary. Also, once cards are moved to foundations, usually they cannot be moved again. Most solitaire games are won when all the cards have been moved to the foundations.

3 Play a card one rank higher and of the same suit on a card. For example, play a 5H on a 4H.

4 The convention we follow is that the right most card on screen is at the bottom of a pile. That is, the cards are placed in a pile on top of the cards to their right on the console.
The FoundationPile should be able to tell if it can accept top Card from another pile. Remember, you can't move Cards from any Pile to any Pile; there are restrictions. If the FoundationPile is not empty, you must build up the Pile with Card of same suit. If, however, the Pile is empty, you will need to check whether the baseCard allows you to place a Card onto the FoundationPile. FoundationPiles do not allow removal of Cards nor are they fanned. This means that you can only see the top Card and once a Card moves to a FoundationPile, it is not possible to remove that Card.

You would want to provide a couple of helper methods in this class for printing and adding Cards.

**TableauPile**

These Piles can hold Cards on a temporary basis for moving them around. Since different Games have different rules regarding what happens in a Tableau Pile when it becomes empty or what kind of Cards can be played there, you will not write this class as a base class; in fact, you don’t even need to write this class at all. Instead you will have Game-dependent tableau Piles. The description of each of those Piles is given later.

**StockPile**

You will implement it as a class derived from Pile. This is the Pile which contains all remaining Cards which you can use during the play. That is, Cards remaining to be placed onto various Piles are kept by StockPile. This Pile never accepts a Card and it does not allow removal of cards except when the destination is a DiscardPile. Remember, the Games we are implementing will require Cards may be removed from the StockPile only if the destination is a DiscardPile.

All Cards in StockPile are faced down and there is no need to make it fanned. Take a look at the sample run of the program how this Pile should print itself.

**DiscardPile** (or WastePile)

This Pile can only accept Cards from the StockPile and keeps the un-played Cards. This Pile has all Cards faced up but is not fanned. This means that only the top Card is visible.

**The Games**

We will implement three Games and each of them will derive from the Game base class. Each Game will have four FoundationPiles which may or may not be circular. If

---

5 Definition from [http://www.goodsol.com/pgshelp/tableau.htm](http://www.goodsol.com/pgshelp/tableau.htm): Piles of cards where building (or packing) is usually permitted. Cards may be packed on the exposed cards of each pile, according to rules that vary from game to game. Usually the game begins with a certain number of cards in each pile. When all the cards in a tableau pile have been played elsewhere, there is usually a rule about whether or what kind of cards can be moved to the empty space.

6 Definition from [http://www.goodsol.com/pgshelp/stock.htm](http://www.goodsol.com/pgshelp/stock.htm): A pile containing the remainder of the cards after all the rest of the piles have been dealt. Cards are usually dealt from the stock in some way, such as to a waste pile or to tableau piles.

7 Definition from [http://www.goodsol.com/pgshelp/waste.htm](http://www.goodsol.com/pgshelp/waste.htm): A pile where unplayable cards are stored. Usually the top card of the Waste is available for play on other piles. In some games the waste pile is turned over to become the Stock (this is called a redeal).
they are not circular, they will build up from A to K and if they are circular, they all must start with the same baseCard. The description of each Game is given below:

**Sixteens**  
This Game has the following Piles:

- Four FoundationPiles each of which can hold a maximum of 13 Cards, all belonging to same suit and building up from A to K. This Game requires that the four FoundationPiles are circular. That is, they may start from any card and build up in a circular fashion. However, the baseCard of all four of these Piles must be the same. These Piles are not fanned.

- Sixteen Tableau piles (call them SixteensTableauPiles) each of which starts with carrying 3 Cards each after the initial deal. During the play, each of these Piles can carry a maximum of 3 Cards. These Piles are fanned and all Cards are faced up. These Piles can accept Cards from any other Pile in the Game except from the FoundationPiles, as long as they build down alternating on color\(^8\) and do not exceed the 3-card limit. For example, either 4C or 4S can be played on 5H, etc. This Pile becomes unusable once it becomes empty. Even a K can not be played on an empty SixteensTableauPile. This Pile is also circular meaning that KD can be played on 1C/AC.

- Two SpecialTableauPiles which are exactly the same as the other Tableau Piles in this Game except that they are reusable even after they are empty. An empty SpecialTableauPile can accept any card. However, the restrictions remain the same as the regular tableau Piles used in this Game, i.e., no more than 3 Cards and building down alternating in color. You must derive this class from SixteensTableauPile class.

**Towers**  
This Game has the following Piles:

- Four FoundationPiles each of which can hold a maximum of 13 Cards, all belonging to same suit and building up from A to K. This Game requires that the four FoundationPiles are not circular. That is, they must start from A and end at K upon completion. These Piles are not fanned.

- Ten Tableau piles (call them TowersTableauPiles) each of which starts with carrying 5 Cards each after the initial deal. During the play, each of these can carry a maximum of 17 Cards (restrictions in Card movement will not let you place more than 17 Cards on any of these Piles). These Piles are fanned and all Cards are up. These Piles can accept Cards from any other Pile in the Game except from the FoundationPiles, as long as they build down on the same suit. For example, only 4H can be played on 5H, etc. Once a TowersTableauPile becomes empty, it can only accept a K of any suit. These Piles are non-circular meaning that KD can not be played on 1D/AD.

- Four special TowersPiles. After the initial deal any two of these contain one Card each. These Cards should be faced up. TowersPiles can accept any

---

\(^8\) Each card has a color in addition to its rank and suit. Diamonds and Hearts have RED color while Spades and Clubs have BLACK color. Thus, 26 cards are RED and 26 cards have BLACK color.
Card as long as it is coming from the TowersTableauPiles. These Piles can not carry more than a single card.

FortyThieves
This Game has the following Piles:

- Four FoundationPiles each of which can hold a maximum of 13 Cards, all belonging to same suit and building up from A to K. This Game requires that the four FoundationPiles are not circular. That is, they must start from A and end at K upon completion. These Piles are not fanned.
- Ten non-Circular Tableau piles (call them FortyThievesTableauPiles) each of which starts with carrying 2 Cards each after the initial deal. During the play, each of these can carry a maximum of 14 Cards (restrictions in Card movement will not let you place more than 14 Cards on any of these Piles). These Piles are fanned and all Cards are up. These Piles can accept Cards from any other Pile in the Game except from the FoundationPiles and StockPile as long as they build down on the same suit. For example, only 4H can be played on 5H, etc. Once a FortyThievesTableauPile becomes empty, it can accept any Card of any suit. The FortyThievesTableauPile can act as a base class from which we can derive the TowersTableauPile class due to similar kind of behavior.
- One StockPile which contains remaining 32 Cards, all faced down, and obviously not fanned.
- One DiscardPile which is empty to start with. As described previously, Cards from StockPile can only be moved to DiscardPile which can only accept cards from a StockPile. DiscardPile is not fanned and all cards are faced up.

What and Where to submit?
1. For VC++ 6.0 (no .NET allowed), submit the complete workspace in zipped format. For UNIX, Submit ONLY those .cc and .h files needed to run your code.
2. Zip all the files together in one zip file whose name should be <your_student_num>.zip where <your_student_num> is an 8-digit number. Be careful with this, our script might throw away zip files that do not follow this convention.
3. Do just one submission of your zip file by dragging and dropping it in the following folder over the network:
   \badar\Common\cs292\section1\project3
   OR
   \badar\Common\cs292\section2\project3
depending upon your section. **DO NOT submit in the wrong section.** Note that you will not be able to open the project3 folder. Just drag and drop your zip file in there.