**My Ship Sails, Gitlin's version**

My Ship Sails is a great kids card game. Read [this](http://www.family-games-treasurehouse.com/my_ship_sails.html) to understand the rules to play. The aim of the game is to collect 7 cards of the "same suit" you choose. You can sail out your ship if you have 7 cards of the same suit. The card numbers are unimportant.

You need to select a card to discard, and pass it to your neighboring player. Each player selects a discard at the same time. And passing is done in a clockwise order.

In this assignment, you are going to play with 4 others. You will be known as player 0 who passes your discarded card to player 1, and then player 1 passes one card to player 2, and so on. You will receive your new card from player 4.

As a game is started, a deck of 52 cards must be shuffled, and 7 cards are dealt to the players.  
One of the players (you) is a human being and selects  
a card to be discarded by entering on the keyboard a letter between 'a' and 'g'.  
The other 4 players are automatons and use some kind of  heuristic to pick a card to be discarded,  
most likely one from the suit that they have the smallest number of cards.

Not required: but you can add additional logic that if trying to reduce the number of that first suit is not working out, an automaton tries to reduce the quantity of a different suit(second least numerous) or if they "see" that they are being passed the same suit by their supplier several rounds in a row, the automaton gives in and tries to maximize the suit that it was being fed.

Note: If you do see that all players continue to pass the same suit over and over, the game will not end unless one of the players (probably you) chooses a different suit or you abort the program.

Shown here are two execution runs of the game.  
Before choosing a card to discard, you are always shown your 7-card hand.  
You, the interactive human, are referenced as player 0.  
The ? is only a cheat that shows your hand and those of the other players (4 automatons).

In this document you will see the word **MUST** a few times. **The use of the word MUST refers to a required program element.**

A discarded card and a replacement card**MUST** always appear in the same position.  
Below the human selected the SK in slot b  
and receives the discard from player 4 of S8.  
As you can see the S8 appears in spot b.

By comparing the ? display before players select their discards  
with the ? display after picking up a new card,  
along with seeing who passed what to the next player,  
you can verify that for all players the slot of the replacement card is the same as what was discarded.  
The ? command is not needed for play, but is a "cheat" and be used in your debugging.

A little further down, you will see that an entry other than '?' or a letter between 'a' and 'g'  
yields "Invalid choice" and a re-prompt of "cmd: ".

The game continues until one or more players  
have seven cards of the same suit.  
In this game, player 4 wins with seven diamonds.  
At the time of a win, you should display the hands of all players.

You **MUST** reproduce the format of all the output,  
and provide the same functionality that is shown.

> myShipSails  
Your hand:  
0:          H7    SK    C3    D1    C4    C2    D4  
             a     b     c     d     e     f     g  
cmd: ?  
0:          H7    SK    C3    D1    C4    C2    D4  
1:          D8    C6    SQ    SJ    CJ    DJ    HQ  
2:          CQ    C7    HJ    S9    HK    D3    H2  
3:          S6    CK    C5    H8    DQ   S10    H6  
4:         H10    D9    D7    H5    H1    D5    S8  
cmd: b  
         Player 0 gave 1 SK  
         Player 1 gave 2 HQ  
         Player 2 gave 3 S9  
         Player 3 gave 4 DQ  
         Player 4 gave 0 S8  
Your hand:  
0:          H7    S8    C3    D1    C4    C2    D4  
             a     b     c     d     e     f     g  
cmd: ?  
0:          H7    S8    C3    D1    C4    C2    D4  
1:          D8    C6    SQ    SJ    CJ    DJ    SK  
2:          CQ    C7    HJ    HQ    HK    D3    H2  
3:          S6    CK    C5    H8    S9   S10    H6  
4:         H10    D9    D7    H5    H1    D5    DQ  
cmd: h  
Invalid choice  
cmd: b  
         Player 0 gave 1 S8  
         Player 1 gave 2 D8  
         Player 2 gave 3 D3  
         Player 3 gave 4 H8  
         Player 4 gave 0 H10  
Your hand:  
0:          H7   H10    C3    D1    C4    C2    D4  
             a     b     c     d     e     f     g  
cmd: f  
         Player 0 gave 1 C2  
         Player 1 gave 2 DJ  
         Player 2 gave 3 D8  
         Player 3 gave 4 H6  
         Player 4 gave 0 H8  
Your hand:  
0:          H7   H10    C3    D1    C4    H8    D4  
             a     b     c     d     e     f     g  
cmd: g  
         Player 0 gave 1 D4  
         Player 1 gave 2 C6  
         Player 2 gave 3 DJ  
         Player 3 gave 4 D3  
         Player 4 gave 0 H6  
Your hand:  
0:          H7   H10    C3    D1    C4    H8    H6  
             a     b     c     d     e     f     g  
cmd: d  
         Player 0 gave 1 D1  
         Player 1 gave 2 D4  
         Player 2 gave 3 CQ  
         Player 3 gave 4 DJ  
         Player 4 gave 0 H5  
Your hand:  
0:          H7   H10    C3    H5    C4    H8    H6  
             a     b     c     d     e     f     g  
cmd: c  
         Player 0 gave 1 C3  
         Player 1 gave 2 D1  
         Player 2 gave 3 D4  
         Player 3 gave 4 D8  
         Player 4 gave 0 H1  
Player 4 is a winner!  
At the end of the game the players had:  
0:          H7   H10    H1    H5    C4    H8    H6  
1:          S8    C3    SQ    SJ    CJ    C2    SK  
2:          D1    C7    HJ    HQ    HK    C6    H2  
3:          S6    CK    C5    CQ    S9   S10    D4  
4:          D3    D9    D7    DJ    D8    D5    DQ

In this second game that follows two players win: the human and player 3.  
> myShipSails  
             a     b     c     d     e     f     g  
cmd: c  
         Player 0 gave 1 C4  
         Player 1 gave 2 C9  
         Player 2 gave 3 C3  
         Player 3 gave 4 HJ  
         Player 4 gave 0 H3  
Your hand:  
0:          H5    H2    H3    H6    H7    H1    C2  
             a     b     c     d     e     f     g  
cmd: g  
         Player 0 gave 1 C2  
         Player 1 gave 2 C4  
         Player 2 gave 3 C9  
         Player 3 gave 4 DK  
         Player 4 gave 0 HJ  
Player 0 is a winner!  
Player 3 is a winner!  
At the end of the game the players had:  
0:          H5    H2    H3    H6    H7    H1    HJ  
1:          D3    S7   D10    S5    SJ    S1    C2  
2:          S3    S4    SK    D6    D5    SQ    C4  
3:          CK    C5    C3    C9    C1   C10    C8  
4:         S10    S6    D1    DK    D9    D2    S2  
>

**This problem could be solved without subclassing,**  
**but I want you to use subclassing, dynamic binding, static and protected data.**

As for inheritance, Charles Darwin might complain but I want you to implement a Player class  
and then derive the subclass AutoPlayer.  
With inheritance,  
you never negate behaviors of ancestor classes.  
A subclass can do everything the  base class can do and even more or do the same thing differently than the base class.   
When did an AutoPlayer ever eat  or drive a car?  
Well, as for the second functionality, it is coming soon.

In order to exercise dynamic binding, you **MUST** have the following declared in main():  
 vector<Player \*> players;  
The address of the human player will be first added to players, followed by the addresses of the four AutoPlayer objects.  
If methods in a base class are marked virtual,  
and derived classes override one of those methods,  
when a derived class object is referenced through a base class pointer, the function of the derived class will be invoked.

Player **MUST** have the methods:  
virtual void chooseDiscard(int);  
void replenish( );

Feel free to add additional methods that you need.

The human Player will use the keyboard to select a card to discard. But an AutoPlayer has no need for interactive input and will have its own version of  
chooseDiscard (thus overriding its parent).  
Though the AutoPlayer will use the same replenish() method inherited from its parent class.

Protected data in a higher class becomes private in all classes that are subclasses.  
You MIGHT find it helpful to have in Player protected members like:  
 int selfIndex;      //index within player vector, starting with 0 for the human  
 int takeFrom;       //player supplying new Card  
 vector<Card> hand;  //holds 7 cards

 Card discard;       //card being discarded  
 int discardIndex;   //position within hand of card being discarded

Static data does not belong to any one instance of a class. Static data belongs to the whole class.  
And, static data can be referenced through any class object or through the name of the class.  
If static data is also made protected, the static data is also available to any of the classes descendants.

So the collection of protected data in Player **MUST** have:  
 static  Card discardList[ ];  
   
That will make the discardList accessible to both the Player object and the AutoPlayer objects.  
This array should have the following use.  
discardList[0] should hold the card that Player discarded and will be picked up by the first AutoPlayer to replenish  its hand.  
discardList[1] should hold the card that first AutoPlayer discarded and will be picked up by the second AutoPlayer, etc.  
discardList[4], the last member of array, should hold the card that the fourth AutoPlayer discarded and will be picked up (given to) the human Player.

Some hints:  
A card should have a suit and face value. The could be implemented using a struct. The suits should be 'S' for Spade, 'H' for Heart, 'D' for Diamond and 'C' for Club.  The value should come be one from 1 through 10, J for Jack, Q for Queen and K for King. If you know that there are 52 cards in a deck and four suits, and use modula arithmetic, i.e. employing the % operator, you should be able to map the numbers 0 to 51 to all the cards in a deck.  
Once established the deck should be shuffled,  
and seven cards should be dealt to the five players.

After the players have been dealt their cards, play progresses until one or more of them holds seven cards of the same suit. With each round,  
the i-th player should place it's discard in discardList[i]. This should be followed by showing the cards discarded by all. Then every player should replenish its hand with card from its supplier.  If you had takeFrom  as a protected member of Player, you could use it to find the card to replenish your hand.

Some design choices:  
The interactive Player has to respond to a command of 'a' through 'g' and '?'.  
The latter caused some difficulty.  
Responding to the '?'  requires accessing all the hands of the players, therefore access to:  
 vector<Player \*> players;  
This vector needs to be set up in main, but also MIGHT be needed by member functions.  
   
This is could be handled a few different ways:  
1. Least desirable: have the vector be a global variable. This should be frowned upon.

2. Pass the vector as a const vector to any member function that requires it.

3. Most desirable: Limit the access to the vector to only be within main.  How do you do so, if the human Player requires the entry of a command?  
Well within main stay in a loop reading invalid commands the '?' and invalid letter choices.  
Only break out of the loop when a choice from 'a' to 'g' is selected, and use the choice to set an index that is passed to Player::chooseDiscard.   
A letter choice of 'a' will result in an index of 0,  
'b'  would yield an index of 1, etc.  
And a 'g' will translate  to an index of 6.  
The index would be used to select a card from a player's seven card array of cards making up a hand.

AutoPlayer::chooseDiscard does not require an interactive choice and could be passed a -1 that is not used. Program logic will used to select the index of the card being discarded.

chooseDiscard will deposit a discarded card in a specific index of discardList[ ]. The human Player deposits a card at index 0. The four AutoPlayer-s deposit cards using indices 1 through 4.

By knowing a player's supplier and the index within its hand of an earlier discard, the replenish function can find the new card to place in the same position of your hand.