

1 Introduction

Each exercise should be its own separate project.

Remember to follow the coding and documentation standards for the class listed on the MyClasses pages.

When you are ready to submit your work create a folder called Homework11 in that folder have separate folders for each project, one folder per project. Put all the code files needed for that project in its respective folder. Do not include the files that the IDE creates, I just want the code files. Zip the entire Homework11 folder up into a single zip file and submit it.

2 Exercises

1. In the last homework you created a struct called `IntList` that stored a size, capacity, and a pointer to an integer. The capacity is the amount of space in the array that is pointed to by list. The size is how many elements are currently in the list. So if your array pointed to by list has 25 cells and you are using just the first 13 of those cells then the size is 13 and the capacity is 25.

The class specification is given below. You will notice that the destroy function is not listed since the destructor will take care of that function. Create the implementation of this class.

```
class IntList {
private:
    int size = 0;
    int capacity = 0;
    int *list = nullptr;

public:
    IntList();
    ~IntList();

    void set(int, int);
    int get(int);
    int get_size();
    int get_capacity();
    void push_back(int);
    void push_front(int);
    int pop_back();
    int pop_front();
    void concat(const IntList&);
    void increase_capacity();
    void sort();
    void print();
};
```

- `set`: Will take two integers, the first is the position to place the element and the second is the element to put into the array. If the position is out of bounds the function should display an error that you are out of bounds and do nothing to the array.

- `get`: Will take a position and return the element in the array at that position. If the position is out of bounds the function should display an error that you are out of bounds and return 0.
- `get_size`: Will return the size of the list, that is, the number of elements in the list.
- `get_capacity`: Will return the capacity of the list, that is, the number of cells available for storage.
- `push_back`: This will insert the integer onto the back of the list. If the capacity is not large enough for the entry the capacity should be increased.
- `push_front`: This will insert the integer onto the front of the list. If the capacity is not large enough for the entry the capacity should be increased.
- `pop_back`: This will remove the integer from the back of the list and return it.
- `pop_front`: This will remove the integer from the front of the list and return it.
- `concat`: This will concatenate the two lists together and store the result in the first list parameter. The new capacity and size are to be the sum of the sizes of the two input lists. That is, they should be the smallest size that will accommodate the two lists.
- `increase_capacity`: This will increase the capacity of the list. If the list is empty, size 0, then the new capacity is to be 1. Otherwise the capacity should be doubled. When the capacity is increased the values in the array and the size should no be altered. So if the current capacity is 8 with a size of 3 storing the numbers 5, 2, and 9 then when the capacity is increased to 16 it should still have a size of 3 storing the numbers 5, 2, and 9.
- `sort`: This will sort the list from lowest to highest. Use either the bubble sort, selection sort, or insertion sort to do this. Do not use or even include the algorithm library.
- `print`: This will print the list on a single line with a space or two between the elements.

Once these functions are written the following main program will produce the following output.

```
#include <iostream>
#include "IntList.h"

using namespace std;

int main() {

    IntList list1, list2;

    list1.push_back(15);
    list1.push_back(-4);
    list1.push_back(3);
    list1.push_back(7);
    list1.print();
    cout << list1.get_size() << " " << list1.get_capacity() << endl;
```

```

list1.push_back(17);
list1.print();
cout << list1.get_size() << " " << list1.get_capacity() << endl;

for (int i = 1; i <= 10; i++)
    list2.push_back(i);

list2.print();
cout << list2.get_size() << " " << list2.get_capacity() << endl;

cout << list2.get(5) << endl;
cout << list2.get(15) << endl;

list2.set(5, 1234);
list2.set(15, -5);
list2.print();

list2.push_front(3);
list2.print();
cout << list2.get_size() << " " << list2.get_capacity() << endl;

cout << list2.pop_front() << endl;
cout << list2.pop_front() << endl;
cout << list2.pop_back() << endl;

list2.print();
cout << list2.get_size() << " " << list2.get_capacity() << endl;

cout << endl;
list1.print();
list2.print();

cout << endl;
list1.concat(list2);
list1.print();
cout << list1.get_size() << " " << list1.get_capacity() << endl;

list2.print();
cout << list2.get_size() << " " << list2.get_capacity() << endl;

cout << endl;
list1.push_back(101);
list1.print();
cout << list1.get_size() << " " << list1.get_capacity() << endl;

list1.sort();
list1.print();
cout << list1.get_size() << " " << list1.get_capacity() << endl;

return 0;
}

```

Output:

```

15 -4 3 7
4 4
15 -4 3 7 17
5 8
1 2 3 4 5 6 7 8 9 10
10 16
6
List bounds error, returning 0.

```

```

0
List bounds error.
1 2 3 4 5 1234 7 8 9 10
3 1 2 3 4 5 1234 7 8 9 10
11 16
3
1
10
2 3 4 5 1234 7 8 9
8 16

15 -4 3 7 17
2 3 4 5 1234 7 8 9

15 -4 3 7 17 2 3 4 5 1234 7 8 9
13 13
2 3 4 5 1234 7 8 9
8 16

15 -4 3 7 17 2 3 4 5 1234 7 8 9 101
14 26
-4 2 3 3 4 5 7 7 8 9 15 17 101 1234
14 26

```

2. This exercise is a simple use of the class you created in the previous exercise. Create a new project and copy the `IntList` class files over to the new project. Now write a main that uses `IntList` and its functions to roll a pair of dice the number of times the user selects. Store the sum of the two die for each individual roll into a `IntList` structure. Then go through the rolls and count all of the 2's, 3's, 4's, ..., 12's. These counts should be stored into another `IntList` object. Finally print out a list of the counts and a list of the probabilities for each roll. You may only use the `IntList` object and its functions to store the data, no arrays, no vectors, not any other type of data structure. A run of the program should look like the following.

```
Input the number of rolls: 1000000
```

```

Counts
=====
2: 27552
3: 55946
4: 83232
5: 111158
6: 138927
7: 166512
8: 138379
9: 111200
10: 83569
11: 55633
12: 27892

Probabilities
=====
2: 0.027552
3: 0.055946
4: 0.083232
5: 0.111158
6: 0.138927
7: 0.166512
8: 0.138379

```

```
9: 0.1112
10: 0.083569
11: 0.055633
12: 0.027892
```

3. In this exercise you will be creating card and deck class structures to simulate a standard poker deck of 52 cards. The specifications of the card and deck are below.

```
#ifndef CARD_H_
#define CARD_H_

#include <string>

using namespace std;

class Card {
private:
    int value;
    int suit;

public:
    Card();
    Card(int v, int s);
    ~Card();

    string toStringFace();
    string toStringSuit();

    bool equals(Card card2);
    string toString();
    string toString(bool space);
    bool greater(Card card2);
};

#endif /* CARD_H_ */
```

```
#ifndef DECK_H_
#define DECK_H_

#include <string>
#include "Card.h"

using namespace std;

class Deck {
private:
    Card deck[52];
    int top = 0;

public:
    Deck();
    ~Deck();

    void PrintDeck();
    void ShuffleDeck();
    Card dealCard();
    Card getCard(int i);
    void reset();
};

#endif /* DECK_H_ */
```

In the card class the data is to be stored as integer values for the value and the suit. The value is to be a number between 1 and 13. The Ace is 1, 2–10 are the numeric cards, 11 is a Jack, 12 a Queen, and 13 a King. The suits are numbered 1–4 in the order of D, C, H, S.

The functions are as follows.

- `Card()`: Default constructor will set the value and suit both to 0, that is, a card that does not exist.
- `Card(int v, int s)`: Constructor that sets value to `v` and suit to `s`.
- `string toStringFace()`: Returns a string of the face value of the card, use A, J, Q, and K for the face cards.
- `string toStringSuit()`: Returns a string of the suit of the card, use D, C, H, and S.
- `bool equals(Card card2)`: Returns true if the two cards are identical, both value and suit are the same. Returns false otherwise.
- `string toString()`: Returns a string of the card in condensed form, for example, 2C, AS, JD, 10H. There should be no space between the value and the suit.
- `string toString(bool space)`: Returns a string of the card in condensed form, for example, 2C, AS, JD, 10H. If `space` is true then there should be one space between the value and the suit, for example, 2 C, A S, J D, 10 H.
- `bool greater(Card card2)`: Returns true if the card has a higher value than `card2`. The suit value does not matter.

The Deck class is to store an array of 52 cards and an integer value for the top of the deck. Some of you used this in the project and it is a quick way to do dealing without changing the deck contents. When you create a deck or after you shuffle you will reset the top to 0. Here is an example of how you can use the top. Say the beginning of our shuffled deck is as follows. So 2D is at index 0, QS index 1, 5H index 2, and so on

2D QS 5H 4C KD JD QH 6C 9C 9S KH 6H QD

Since the top is 0 then `deck[top]` is the card that is next to be dealt. So when you deal a card all you need to do is return the card at the top position then increment top. Now top is 1 and the next card to deal is `deck[top]` (the QS), a deal will return this card and increment top to 2 and hence another deal will return 5H and increment top to 3, and so on.

- `Deck()`: Constructor that creates the deck of cards.
- `void PrintDeck()`: Prints out the deck in a single line using no space between the value and suit for each card and one space between the cards.
- `void ShuffleDeck()`: Shuffles the deck of cards.
- `Card dealCard()`: Deals the next card off the top of the deck.

- Card `getCard(int i)`: Gets the card at index `i` in the deck.
- void `reset()`: Resets the top to 0;

Once these classes are constructed test them with the following program.

```
#include <iostream>

#include "Deck.h"
#include "Card.h"

using namespace std;

int main() {
    Deck deck1, deck2;

    deck1.PrintDeck();
    deck1.ShuffleDeck();
    deck1.PrintDeck();
    cout << endl;

    for (int i = 0; i < 10; i++)
        cout << deck1.dealCard().toString() << " - ";
    cout << endl;

    deck1.reset();

    for (int i = 0; i < 5; i++)
        cout << deck1.dealCard().toString() << " - ";
    cout << endl;

    for (int i = 3; i < 10; i++)
        cout << deck1.getCard(i).toString() << " - ";
    cout << endl;

    deck2 = deck1;

    cout << endl;
    deck1.PrintDeck();
    deck2.PrintDeck();

    cout << endl;
    deck1.ShuffleDeck();
    deck1.PrintDeck();
    deck2.PrintDeck();

    return 0;
}
```

The output should look something like the following. The deck prints will be on a single line and your ordering will be different.

```
AD 2D 3D 4D 5D 6D 7D 8D 9D 10D JD QD KD AC 2C 3C 4C 5C 6C 7C 8C 9C 10C JC QC KC
AH 2H 3H 4H 5H 6H 7H 8H 9H 10H JH QH KH AS 2S 3S 4S 5S 6S 7S 8S 9S 10S JS QS KS

6C 7S 5S 3C 4H QC 7C KS 5C AS 4S 2C JC 8D 4C KH KD 9H 9D 6H 6S QD 8C 10D 5H 3D
JS JH JD 6D 2D QS 7D 4D 3S 9C AD 10S 8H 8S QH 3H KC 9S 2H AH 10H 2S 5D 10C 7H AC

6C - 7S - 5S - 3C - 4H - QC - 7C - KS - 5C - AS -
6C - 7S - 5S - 3C - 4H -
3C - 4H - QC - 7C - KS - 5C - AS -

6C 7S 5S 3C 4H QC 7C KS 5C AS 4S 2C JC 8D 4C KH KD 9H 9D 6H 6S QD 8C 10D 5H 3D JS
```

```

JH JD 6D 2D QS 7D 4D 3S 9C AD 10S 8H 8S QH 3H KC 9S 2H AH 10H 2S 5D 10C 7H AC

6C 7S 5S 3C 4H QC 7C KS 5C AS 4S 2C JC 8D 4C KH KD 9H 9D 6H 6S QD 8C 10D 5H 3D JS
JH JD 6D 2D QS 7D 4D 3S 9C AD 10S 8H 8S QH 3H KC 9S 2H AH 10H 2S 5D 10C 7H AC

KS KD 8C 2D 2C QC 9C 10H KC AH 9H 7D 10D AD 6C 7C KH JS 2S JD 4S 7H 7S 3S 3D 8S JH
6S 5H JC 8D 5S 3C QH 9D 6H 10S 5C QS 3H 4C 2H AC 4H 9S 5D 8H QD AS 10C 4D 6D

6C 7S 5S 3C 4H QC 7C KS 5C AS 4S 2C JC 8D 4C KH KD 9H 9D 6H 6S QD 8C 10D 5H 3D JS
JH JD 6D 2D QS 7D 4D 3S 9C AD 10S 8H 8S QH 3H KC 9S 2H AH 10H 2S 5D 10C 7H AC

```

4. This exercise uses the card and deck classes you just created. Create a new project and copy all of the card and deck files to the new project.

A *derangement* is when you have a list of objects and then mix them up so that no object is in the same position as it started. For example, if we have the list 1 2 3 4 5, then the list 2 5 4 1 3 is a derangement of the original list but 2 1 3 5 4 is not since 3 is in the same position as where it started.

The same can be done with larger lists and is a common game using a deck of cards. Take a new deck of cards out of its wrapper and then bet someone that no matter how many times they shuffle the deck there will be at least one card in the same position as it was in the originally manufactured deck. The question is, would you take that bet? Most people do since they feel that if they shuffle the deck enough times then the cards will all be mixed up.

In these printouts, the top row is the original order of the deck and the second row is the shuffled deck. Looking at the first run it is clear that this is not a derangement.

```

AH 2H 3H 4H 5H 6H 7H 8H 9H 10H JH QH KH AD 2D 3D 4D 5D 6D 7D 8D 9D 10D JD QD KD
AS 6S 5C 9S 4H 6H 8H 10C 2H 9D 7C AC 8S 3C 6D 4C 5H QD 2D 5S QC 3S 7H 2C AD KD

AC 2C 3C 4C 5C 6C 7C 8C 9C 10C JC QC KC AS 2S 3S 4S 5S 6S 7S 8S 9S 10S JS QS KS
JD 10S AH JH 5D 4S QH 2S 8D JS 7S KS KC 9H 8C 3D 10D 6C KH 9C QS 4D 10H JC 7D 3H

```

On the other hand, the following is a derangement,

```

AH 2H 3H 4H 5H 6H 7H 8H 9H 10H JH QH KH AD 2D 3D 4D 5D 6D 7D 8D 9D 10D JD QD KD
10C 5S 3C 7H KH 2S 10D 5D JC 10S AS 2H QC 10H JS 4S 7S 5H 4D KD AC 3D KS 8D 9S 4C

AC 2C 3C 4C 5C 6C 7C 8C 9C 10C JC QC KC AS 2S 3S 4S 5S 6S 7S 8S 9S 10S JS QS KS
KC 4H 3S 9H 9D 2C 6D AD JH 8C JD 8H QD 6C 2D 6H AH 7D 3H 9C QH 6S 8S QS 5C 7C

```

Write a program that will simulate the shuffling many times and determine if the shuffle produces a derangement. We will count the number of derangements and then find the probability of a shuffled deck being a derangement. This will answer the question of whether or not you should take that bet. This program is not long if you use the functions in the card and deck classes efficiently. The program should take the number of trials to be done from the user. You will have two decks of cards, one you will shuffle on each trial and the other you will leave alone as the original deck order. Then go through the two decks card by card, if there is a match of a pair of cards then the shuffle

is not a derangement and if there are no matches then the shuffle is a derangement. Keep a count of the number of derangements found in the number of trials the user wanted. Then calculate the probability of a derangement by dividing the number of trials into the number derangements found. Also calculate one over the probability.

Answer the following questions, put the answers in the comments section of the main.

- (a) What is the approximate probability of a shuffled deck of cards being a derangement?
- (b) What is the approximate value of one over the probability of a shuffled deck of cards being a derangement?
- (c) Does the number for one over the probability look close to a very famous number you have seen before? If so, what is the famous number?