CS 31 Worksheet 7

This worksheet is entirely **optional**, and meant for extra practice. Some problems will be more challenging than others and are designed to have you apply your knowledge beyond the examples presented in lecture, discussion or projects. All exams will be done on paper, so it is in your best interest to practice these problems by hand and not rely on a compiler.

Concepts: Review

1. You've received a ransom note in the mail with the message pieced together by cut-out letters, and it is your job to discover which magazine the letters originated from. Implement the following function to determine whether the message is recreatable from a given magazine page.

   ```
   bool isFromMagazine(string magazine, string message)
   ```

   Examples:

   ```
   magazine = "HAPPY THANKSGIVING 2017 - SPEND AND SAVE $ FOR
   GIFTS"
   message = "GIVE $7012 TODAY"

   isFromMagazine(magazine, message) // returns true

   magazine = "Have a wonderful holiday";
   message = "HAVE A WONDERFUL HOLIDAY";

   isFromMagazine(magazine, message) // returns false; the
   function is case-sensitive
   ```

2. Write a function with the following header:

   ```
   void oscillatingSort(int* arr, int len)
   ```

   Given an array *arr* of unique integers and its length *len*, rearrange the contents of *arr* so that the values alternate between increasing and decreasing.

   ```
   int a[6] = {1, 2, 3, 4, 5, 6};
   oscillatingSort(a, 6);
   // a could now be {1, 6, 2, 5, 3, 4}
   ```

3. Write a function, `swapPtrs`, that takes in two integer pointers, a and b, and swaps their addresses (not whatever values at the addresses they point to). You should know that memory addresses for variables are compiler assigned, so pa will likely not be 0xffff830 as shown below, but the example is just to illustrate how the two pointers passed into the function would exchange address values):

```
int a = 10;
int b = 20;
int* p1 = &a;
int* p2 = &b;
cout << p1 << endl; // prints address of variable a, say 0xffff830
cout << p2 << endl; // prints address of variable b, say 0xffff834

swapPtrs(p1, p2);
cout << p1 << endl;   // prints 0xffff834 (address of variable b)
cout << p2 << endl;  // prints 0xffff830 (address of variable a)
```

4. You work for a company, Stripify, that offers a special discount to students. The company needs to ask for each student customer's edu email address to verify their eligibility for the discount. Write a function, `isValidEdu()`, that takes in a C-string and returns whether that C-string is a valid email address. For simplicity, a valid edu email address has the form [userName]@[domain].edu, where [userName] and [domain] must be non-empty, and must consist of alphanumeric characters.

```
isValidEdu("TACAN") == false
isValidEdu("@glideslope")==false
isValidEdu("adf@ndb.edu") == true
isValidEdu("twin777@.ETOPS.com") == false
isValidEdu("ILS.DME@RNAV.edu") == false
```

5. Implement a `Netflix` class which holds `Show` objects in a "watching queue".

```
class Netflix {
      Public:
            Netflix(int capacity);
            void watch(string name);
            bool add(string name);
            void cleanUp();
            ~Netflix();
      private:
            int num_shows; // number of shows in queue
```

```
            Show* queue[num_shows];
            int m_capacity;
            // Hint: you can use this function in cleanUp()
            void remove_from_queue(int index) {
                  delete m_shows[index];
                  for (int i = index; i < num_shows - 1; i++) {
                        m_shows[i] = m_shows[i+1];
                  }
                  num_shows--;
            }
};

class Show {
      public:
            Show(string name);
            void watch();
            bool isWatched();
            string getName();
      private:
            string name;
            bool is_watched;
}
```

Implement all of the functions highlighted in blue.

1. Netflix(int capacity) -- declare a Netflix object with a maximum capacity for number of shows in queue.
2. void watch(string name) -- tells the Netflix object that you want to watch a particular show (as a result when cleanUp is called, the show you watched should be removed from the queue)
3. bool add(string name) -- add a new show to your queue. If the addition is successful (queue is not full), return True, else return False.
4. void cleanUp() -- clean up the queue and remove all shows that have been watched. Update the number of shows to reflect this change
5. ~Netflix() -- destructor, make sure you remember to use delete everything you have created on the heap!
6. Show(string name) -- declare a Show object with a name
7. void watch() -- updates the Show object from unwatched to watched. All shows are initially “unwatched”
8. bool isWatched() -- returns True if show has been watched, False otherwise

9. `string getName() -- getter for Show's name`

Sample use case:

```
int main() {
      Netflix n(3);
      n.add("Stranger Things"); // returns True
      n.add("The Office"); // returns True
      n.add("Arrested Development"); // returns True
      n.add("Sherlock"); // returns False
      n.watch("The Office");
      n.cleanUp();
      n.add("Sherlock"); // returns True
}
```

6. Write a function with the following header:

```
int parseCorruptCommand(char* cmd, int* arr, int len, int*&
loc);
```

This function takes in a C-string *cmd*, an array of positive integers *arr*, an array length *len* and an integer *loc*.

A valid command comes in the form `"get*"`, where * can be replaced by any valid integer value. If given the command "get5", the function would return the 5th value stored in `arr` and set loc to the point to the location of that value within the array. If the * is an integer >= len or < 0, return -1 without modifying loc.
In the case of an invalid command, the function would return -1 without modifying loc. An invalid command is any command (e.g. "put", "foo", etc) that does not take the form of a valid command.

Your job is to parse the command, determine if it is valid or invalid and do the necessary corresponding actions. However, the original `cmd` was corrupted and filled with noise in the form of **non-alphanumeric** characters. For example, the valid command "get5" could become `"@$@$@g####e@()t & 5"` and the invalid command "foo" could become "`fo##o()`".

`parseCorruptCommand` should parse the command to determine its validity and then set the necessary values. It should treat a corrupted command like `"@$@$@g####e@()t & 5"` as if it were the uncorrupted command "get5".