Assignment 2
Feedback
Drawing of Data Structure

Ragged Array (or jagged)

Elements

0: 1234567891
1: 1342567819
2: 1731126719

0: 12345678
1: 512131711
2: 5121556719
3: 5211517111
4: 5321641911
5: 5512561091
6: 5761178171
7: 5761178171

Element Count:

0: 3
1: 0
2: 0
3: 6
4: 0
5: 1
6: 0
7: 0
8: 1
9: 0

Capacity:

0: 4
1: 0
2: 0
3: 8
4: 0
5: 2
6: 0
7: 0
8: 0
9: 0
Drawing of Data Structure

“In our drawing, we must label the components of our data structure and include some test data.”

- Missing test data or only 1 test data -> so cannot ascertain the sorted stated of arrays

- Snapshot of our data structure (no code)
  - Imagine we execute our WICPS and at one point we freeze execution and take a snapshot of the memory content
Cannot have 10 different variables (pointers) each pointing to an array or Patient objects
  ▪ Repeated code

More efficient to group these 10 different variables (pointers) into an array
  ▪ Eliminating the repeated code
  ▪ Less error-prone
  ▪ Better software development (programming) style.
In Assignment 2, we had to make use of dynamic memory:

- Allowed us to manipulate each section of our data structure individually
  - section -> containing Patient objects with care card number starting with the same digit

Lab 3 says that if an object of a class contains dynamically-allocated memory, we must provide a destructor and a copy constructor as part of the class when we implement this class:

- This meant that our List class needed a destructor and a copy constructor
So, in List.h, when it said:

```
// We must add at least 1 constructor
// and a destructor
```

what this meant was that our List class needed a destructor and a copy constructor even though our WICPS did not make use of it.

When designing/implementing a data collection ADT class, we offer “services” to client code:

- We want to make our class as useful as possible.
Example of Copy Constructor

```cpp
// Copy constructor
MyADT::MyADT(const MyADT& rhs) {
    //cout << "MyADT::copy constructor" << endl; // For testing purposes ... and curiosity
    for (int index = 0; index < MyADT::MAX_ALPHA; index++) {
        elements[index] = NULL;
        elementCount[index] = rhs.elementCount[index];
        capacity[index] = rhs.capacity[index];

        // Deep copy the elements
        if (rhs.elements[index] != NULL && rhs.elementCount[index] == 0) {
            elements[index] = new Profile[capacity[index]];
            for (int element = 0; element < elementCount[index]; element++) {
                elements[index][element] = rhs.elements[index][element];
            }
        }
    }
} // end copy constructor
```
Example of Destructor

```cpp
MyADT::~MyADT() {
    // cout << "MyADT::destructor" << endl; // For testing purposes ... and curiosity
    // Delete all dynamically allocated memory
    deleteAll();
} // end destructor

// Description: Remove all elements.
void MyADT::removeAll() {
    // cout << "MyADT::removeAll()" << endl; // For testing purposes
    // Reset the data collection as it was when it was initially constructed
    deleteAll();
} // end removeAll

// Description: Deletes all dynamically allocated memory.
void MyADT::deleteAll() {
    // cout << "MyADT::deleteAll()" << endl; // For testing purposes ... and curiosity
    // Delete each array of elements associated with a letter ...
    for (int index = 0; index < MyADT::MAX_ALPHA; index++) {
        if (elements[index] != NULL) {
            delete [] elements[index];
            elements[index] = NULL;
        }
        // and set its corresponding element count to 0 ...
        elementCount[index] = 0;
        // and its corresponding capacity to original capacity
        capacity[index] = MyADT::MAX_ELEMENTS;
    }
} // end deleteAll
```