Lab 3 Discussion: Text class with char array buffer and int bufferSize as data members.

1. Useful includes,

```
#include <iostream>
#include <iomanip>
#include <cstring>
#include "Text.h"
```

2.Text Constructors

a. Parameterized constructor with initial values. See the method prototype below,

```
Text ( const char *charSeq = "" );
```

How to implement this constructor in Text.cpp?

```
/* Creates a string containing the
delimited sequence of characters
charSeq. Allocates enough memory for
this string.
Precondition:none
Postcondition: new Text object
constructed as either the empty string
or the sequence of characters in
charSeq */
```

```
Text:: Text ( const char *charSeq )
{
    bufferSize = strlen(charSeq) + 1;
    buffer = new char [ bufferSize ];
    strcpy(buffer,charSeq);
}
```

How could this constructor be used by the client (in main.cpp)?

```
Text t1;
Text t2("Hello World");
```

---

b. Copy constructor

In our simple Circle class we were allowed to do the following,

```
Circle c1(5, 3, 10);
Circle c2(c1); //copy c1 into c2

Circle foo();  //return a copy of a
               circle object
void bob(Circle c); //pass a copy of a
            circle object by value
```

This was because the data members of the Circle class were primitive data

types (float) that the compiler knows how to copy.

QUESTION: Can we do this in C?

```
int array1[] = {1,2,3,4};
int array2 = array1;
```

NO! The compiler copies only the address of array1 into array2. This is called a shallow copy where changes to either array effects them both.

When dynamic memory allocation is used for any data member in a class then several methods must be provided,
A Copy constructor and an assignment operator overload. These tell the compiler how to copy the object.

```
  /* Copy constructor, creates a copy of
  valueText.
  Precondition:none
  Postcondition: creates a Text object
  that is a copy of valueText */

Text:: Text ( const Text &valueText )
  : bufferSize(valueText.bufferSize)

{
    buffer = new char [bufferSize];
    strcpy(buffer,valueText.buffer);
}
```

3. Assignment operator – this is given in the Text_ADT.doc

```
/* Assigns other to a Text object.
Precondition:none
Postcondition: left side of = right
side

void Text:: operator = ( const Text&
other )
{
    int rlen = other.getLength();
    if ( rlen >= bufferSize )
// If other will not fit
    {
       delete [] buffer;
// Release buffer and
       bufferSize = rlen + 1;
//  allocate a new larger buffer
       buffer = new char[ bufferSize ];
    }
    strcpy(buffer,other.buffer);  //
Copy other
}
```

ShowStructure method is given in Text_ADT.doc – basic print of characters in an array in a nice format.

Because we dynamically allocate memory for a Text object, we are now responsible for telling the compiler how to deallocate memory for the object. This is done in a destructor method.

4. Class destructor – same name as class, preceded by ~, no return type. The destructor performs garbage collecton of an object. This method is never called by the client program. Instead the compiler creates the destructor call whenever the class object goes out of scope (its lifetime expires).

```
~Text ();

Text:: ~Text ()

// Frees the memory used by the Text
object buffer.

{
    delete [] buffer;
    bufferSize = 0;
}
```

5. The in lab exercise reads from an input file. For both Xcode and Clion, the input file must be in the same folder as the executable. In Xcode, open the Products folder in left pane of window. Right click on black icon (executable file) and select “Show in Finder”. This opens the directory and you can copy the input file into the directory. For CLion versions, I suggest you Google the location where CLion stores the executable. But in my experience,

All executable files from CLion should be located in the cmake-build-debug folder of your ClionProjects\ProjectName\ directory. For Windows users the executable should have the .exe extension.