**CSCI 1411 – Lab 10 – CStrings and Strings**

**Goals:**

* Understand how to build a CString
* Understand how to use some of the CString functions
* Understand how to use some of the string member functions

**Development Environment:** (all students must use Visual Studios)

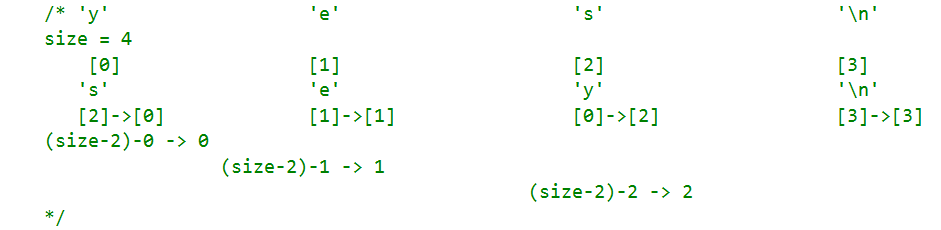
**Skills:** use of strings, cstrings, arrays, functions

**Reading**: Chap 10

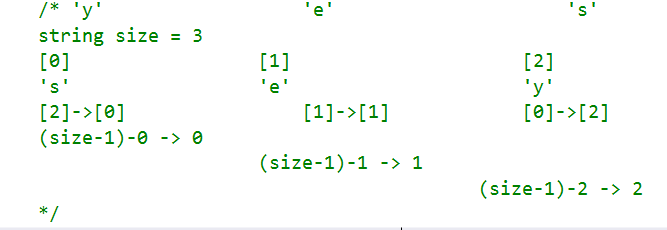
**Deliverables:** 1) This lab with 2 screen shots 2) lastnameFirstLab10.cpp

**Part I – Skills Practice (5 points)**

* Open a new project in Visual Studios. Call the project Lab10. Don’t forget to uncheck the Pre-compiled headers and Security Development Lifecycle checks. Then check the box for an empty project
* For this project we are reversing a CString and a string. A C String isn’t a data type itself, but just defines an array of characters with a ‘/n’ single character at the end. So if a CString has 5 characters, we must have an array of at least 6 characters. 5 characters plus a ‘n’. Then when printing in C, you would print until you get to a ‘\n’ which is a non-printable character.
* Remember that a char \* is a pointer to a character that can point to the start of a char array.
* In order to determine the algorithm needed to reverse characters in a CString we must first try to map out how we would manually move the characters within the array for a specific word, then try to generalize.
* Write down the math formula you would use for the 4th letter in a 4 letter word (based on the chart below)



* Similarly, we have to figure the algorithm for a string (which is a C++ object, not an array of characters). A string has its own “member functions” or functions associated with the object. When it gives size, it does not count the trailing ‘\n’. So if there are 3 characters, the size would be 3.
* Again, write the formula for the 4th letter in a 4 letter word (based on the chart below for strings.)



You will be creating 3 files called lab10a.cpp, functions.h and functions.cpp

* In functions.h.

**#ifndef FUNCTIONS\_H**

**#define FUNCTIONS\_H**

**using namespace std;**

**#include <string>**

**char \*backwardCString(char \*cstring, int size);**

**string backwardString(string stringObject);**

**void printCString(char\* cString, int size);**

**#endif**

* In functions.cpp

**#include <iostream>**

**#include "functions.h"**

**using namespace std;**

**char\* backwardCString(char \*cstring, int size)**

**{**

**char\* backCString = new char[size];**

**//remember that the backCString[size-1] needs to contain a '\n'.**

**//'\n' is just one character**

**backCString[size - 1] = '\n';**

**//now we want to reverse all of the letters before the '\n'**

**for (int i = 0; i < size; i++)**

**{**

**backCString[size-2 - i] = cstring[i];**

**}**

**return backCString;**

**}**

**string backwardString(string stringObject)**

**{**

**string backString = stringObject;**

**for (int i=0; i < stringObject.size(); i++)**

**backString[stringObject.size()-1 -i] = stringObject[i];**

**return backString;**

**}**

**void printCString(char\* cString, int size)**

**{**

**for (int i = 0; i < size; i++)**

**cout << cString[i];**

**cout << endl;**

**}**

* In Lab10a.cpp

**#include <iostream>**

**#include "functions.h"**

**using namespace std;**

**int main()**

**{**

**char\* cstring = "C string";**

**char\* reverse;**

**const int size = 9;**

**string mystring = "string", reverseString;**

**cout << "cstring:" << endl;**

**printCString(cstring, size);**

**reverse = backwardCString(cstring, size);**

**printCString(reverse, size);**

**cout << endl << "string:" << endl;**

**cout << mystring << endl;**

**reverseString = backwardString(mystring);**

**cout << reverseString << endl;**

**return 0;**

**}**

* **Run** it. Take a **screen shot** of the successful output and place it below. Replace this output with your output (and 2 different cars) For a Windows 10 screen shot: Alt key + PrtSc key. Then Ctrl + V to paste. For Mac: Shift + Command + 4. You will not credit unless you have a successful screen shot with Your name in the output.

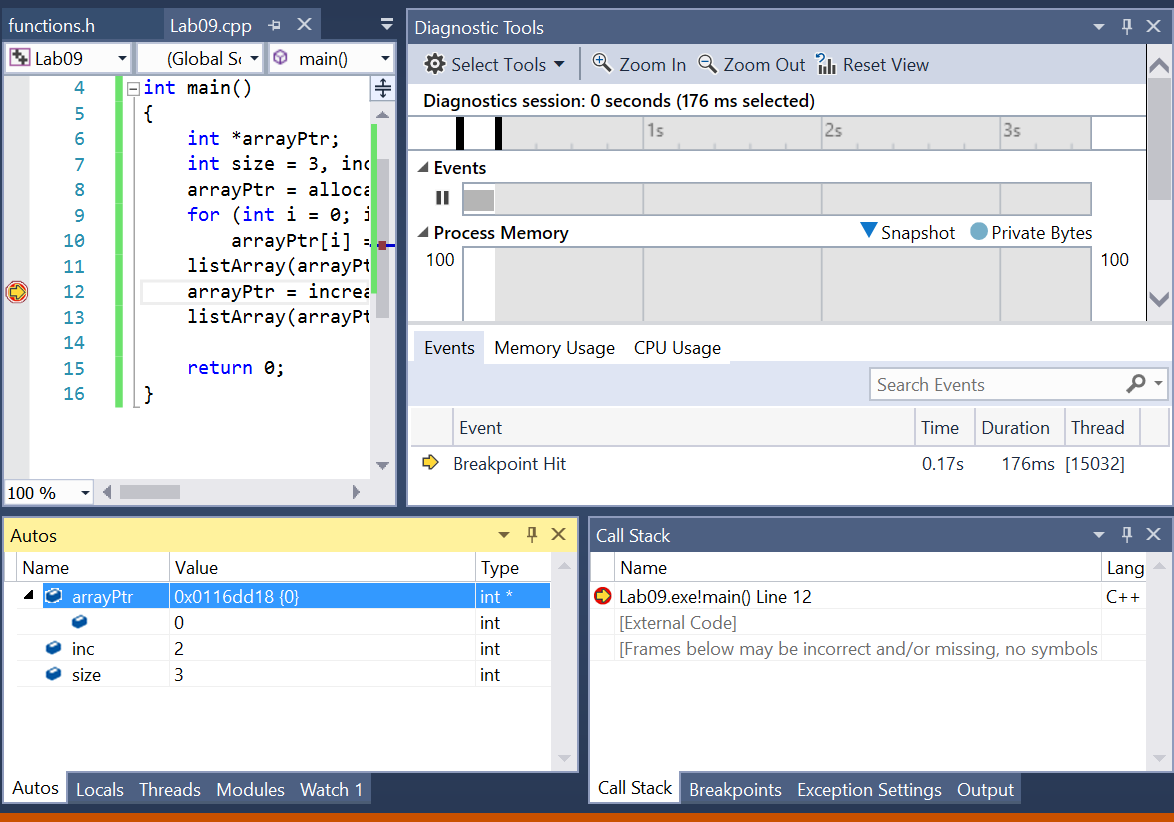
**Part II – Use of Visual Studios Debugger (5pts)**

For this section, we are going to use the Visual Studios Debugger. You should use the debugger whenever you are using pointers in order to ensure that you are not going out of bounds and are pointing to the correct variables.

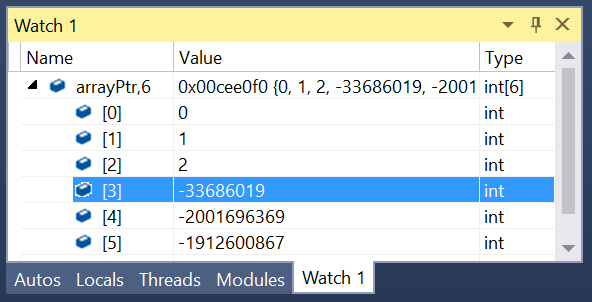
* Go to the Lab09a.cpp file
* Click on the number to the left of the line

**arrayPtr = increaseArray(arrayPtr,size, inc);**

* This will highlight the line
* Right Click. Go to Breakpoint-Insert Breakpoint
* Build->Re-Build Solution
* Debug->Start Debugging
* It will run the first part, and you will see a shell window open that starts printing the initial numbers. Now it is stopping at the line that you set a breakpoint.
* This should open window frames that look something like this



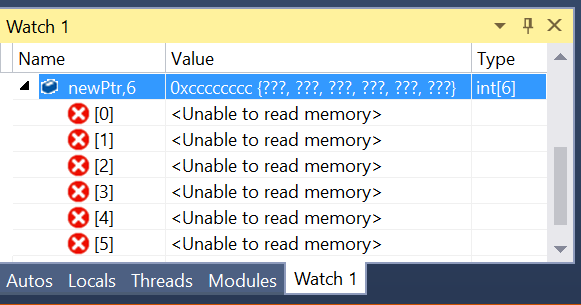
* Let’s focus on the Autos window. Note that inc and size have the value that you set at the beginning. ArrayPtr has a value starting with 0x which says that it is a hexademical number. This is a very long number (you’ll learn more in CSCI 1510) that denotes a starting memory address that the pointer “points to”. Note that the arrayPtr has an arrow before it, meaning you can expand it to see more information. In this case, the first value it points to is zero.
* Now what if we want to see the other values of arrayPtr?
* In the Autos tab, right click on arrayPtr and go to Add Watch. This places it in the Watch Window. Double click on the word arrayPtr, and change it to watch 6 positions. By changing arrayPtr to arrayPtr,6. It should look similar to this



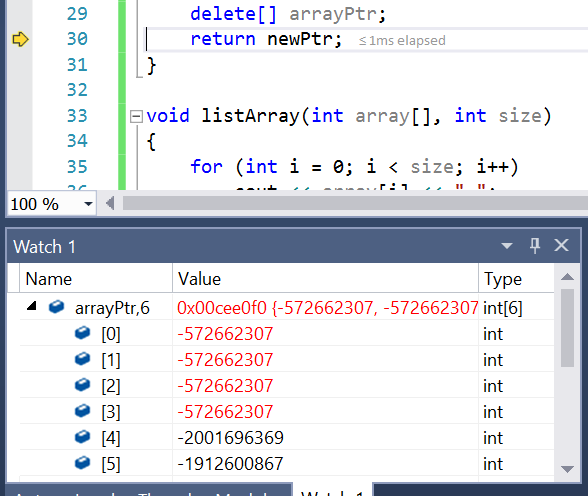
* Notice that you had set the values of the first 3 places inside Lab09a.cpp (at the for loop). You did not set the value for arrayPtr[3],[4], or [5], so that’s why it is whatever was in that memory position.
* Now we have a choice of going to the next statement, or stepping over the next statement. Hover your cursor over the symbols in the top right to see this (or they are also listed under Debug)



* First Step Into. That will take you into the increaseArray function
* Step over the **int \*newPtr;**
* Then you will see down in Autos that there is a newPtr entry. Right Click and Add Watch
* In the Watch window, double click on newPtr and change to newPtr, 6. Hit the enterkey



* **It is unable to read the memory, because if you notice at this point in the code we have not yet done the new, giving values.**
* **Step Over the newPtr = new int[currentSize + increaseBy];**
* Keep stepping over the for loop, noting that the value of newPtr[0], [1], etc. will change as you go through the loop.
* Take a **screen shot** after the values of newPtr [0], [1], [2] are filled. Place that below
* When you Step Over the **delete[] arrayPtr;** note that addresses go red, because this variable is no longer assigned in the program (because you removed the entire array from memory with delete[])



* Go to the stop icon and stop debugging. You can then click on the entire line again and Breakpoint->Delete Breakpoint.

It takes some practice to know how and when to use the debugger, but the above exercise should help. Please try this on your own in future programs, especially those dealing with pointers and memory addresses.