CIS 200 - Lab 0801

1. Problem Statement

Create a program which takes in two words, organizes the letters into two respective linked lists which are organized by letter order. Then merge them together and print out the final result.

2. Requirements

2.1 Assumptions

- Written for Windows 10 Only
- Not Compiled for Linux
- Testing *main*() was *mostly* provided
- User will only input lower case words

2.2 Specifications

- '+' Operator will be overloaded for this example
- User will fill two initial word linked lists
- Linked Lists will be organized then combined

3. Decomposition Diagram

- Program
    - Input
        - User Enters Two Words
    - Process
        - Fill and Sort Linked Lists
        - Combine Link Lists
    - Output
        - Print Sorted Linked Lists

4. Test Strategy

- Valid Data

5. Test Plan Version 1

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
Valid Data	1	Valid Word 1 Letters				
Valid Data	2	Valid Word 1 Occurrences				
Valid Data	3	Valid Word 2 Letters				
Valid Data	4	Valid Word 2 Occurrences				
Valid Data	5	Combined Above Letters				
Valid Data	6	Combined Above Occurrences				
Valid Data	7	Similar Words Letters				
Valid Data	8	Similar Words Occurrences				

6. Initial Algorithm

1. Create Structure
    a. Name: sortedListNode
    b. Contents:
        i. character letter
        ii. integer occur
        iii. pointer sortedListNode
2. Create Class
    a. Name: arraySortedList
    b. Public:
        i. sortedListNode head
3. Create Function
    a. Name: fromString
    b. Parameters: Reference arraySortedList list, string word

c. Return: VOID
d. Method:
e. Declare Movement Nodes of type sortedListNode
    i. prior
    ii. current
    iii. temp
f. For Each Letter In Word
    i. If the list is empty
        1. Add new node
    ii. If new first element
        1. Add as the first element and link remaining members
    iii. If equals the first node
        1. Add 1 to occurrences
    iv. Else
        1. Search list to find the element or add
    v. If Found
        1. Add 1 to occurrences
    vi. If Element Higher Is Found
        1. Add to list before the higher element
    vii. If End of List is Reached
        1. Not found, add as the last element

4. Create Function
    a. Name: printList
    b. Parameters: Reference arraySortedList list
    c. Return: VOID
    d. Method:
        i. Declare Current Marker
        ii. Set Current to Head of list
        iii. If current equals null
            1. Skip
        iv. Else
            1. While current not equal NULL
                a. Print Next Letter
        v. Set Current to Head of list
        vi. If current equals null
            1. Skip
        vii. Else
            1. While current not equal NULL
                a. Print Next Occurrence
5. Create Main
    a. Declare Holder Strings
        i. word1
        ii. word2

iii. word3

b. Declare Head of Linked Lists
   i. list1
   ii. list2
   iii. list3
c. Prompt User For Input
   i. Fill word1
   ii. Fill word2
d. fromString(list1, word1)
e. printList(list1)
f. fromString(list2, word2)
g. printList(list2)
h. Combine word1 and word2
i. fromString(list3, word3)
j. printList(list3)

7. Test Plan Version 2

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
Valid Data	1	Valid Word 1 Letters	testing	e, g, i, n, s, t		
Valid Data	2	Valid Word 1 Occurrences	testing	1, 1, 1, 1, 1, 2		
Valid Data	3	Valid Word 2 Letters	lasagna	a, g, l, n, s		
Valid Data	4	Valid Word 2 Occurrences	lasagna	3, 1, 1, 1, 1		
Valid Data	5	Combined Above Letters	testing, lasagna	a, e, g, i, l, n, s, t		
Valid Data	6	Combined Above Occurrences	testing, lasagna	3, 1, 2, 1, 1, 2, 2, 2		
Valid Data	7	Similar Words Letters	testing, islands	a, d, e, g, i, l, n, s, t		

Valid Data	8	Similar Words Occurrences	testing, islands	1, 1, 1, 1, 2, 1, 2, 3, 2		

8. Code

[source.cpp]

```
#include <iostream>
#include <string>

//Structure for Linked List
struct sortedListNode
{
	char letter;
	int occur;
	sortedListNode * next;
};

//Class for Linked List to be passed into functions
class arraySortedList
{
public:
	sortedListNode * head = NULL;
};

//Function Prototypes
void fromString(arraySortedList&, std::string);
void printList(arraySortedList&);

int main()
{
	//Declare Character Arrays for User Input
	std::string word1;
	std::string word2;
	std::string word3;

	//Declare Head of Linked Lists
	arraySortedList list1;
	arraySortedList list2;
	arraySortedList list3;

	//Prompt User for Input
	std::cout << "Please Enter The First Word: ";
	std::cin >> word1;
	std::cout << "Please Enter The Second Word: ";
	std::cin >> word2;
```

```
	//Compute and Print First Word Details
	std::cout << std::endl << "First Word Details " << std::endl;
	fromString(list1, word1);
	printList(list1);

	//Compute abd Print Second Word Details
	std::cout << std::endl << "Second Word Details " << std::endl;
	fromString(list2, word2);
	printList(list2);

	word3 = word1 + word2; //Operator Overloading*
                                                                *technically

	//Compute and Print Combined Word Details
	std::cout << std::endl << "Combined Word Details " << std::endl;
	fromString(list3, word3);
	printList(list3);

	system("pause");
	return 0;
}

//Description: Take in word and linked list, fill the linked list with the letters in
alphabetical order.
//Pre-Condition: Word should just include lower case letters
//Post-Condition: Sorted Link LIST!
void fromString(arraySortedList& list, std::string word)
{
	//Declare Movement Nodes
	sortedListNode * prior;
	sortedListNode * current;
	sortedListNode * temp;

	//BEGIN Corrected From Class
	for (int i = 0; i < word.length(); i++)
	{
		if (list.head == NULL)
		{
			temp = new sortedListNode;
			temp->letter = word.at(i);
			temp->occur = 1;
			temp->next = NULL;
			list.head = temp;
		}
		else if (list.head->letter > word.at(i)) {
			temp = new sortedListNode;
			temp->letter = word.at(i);
			temp->occur = 1;
```

```
                        temp->next = list.head;
                        list.head = temp;
                }
                else if (list.head->letter == word.at(i))
                {
                        list.head->occur++;
                }
                else
                {
                        prior = NULL;
                        current = list.head;

                        while (current != NULL && current->letter < word.at(i))
                        {
                                prior = current;
                                current = current->next;
                        }
                        if (current == NULL) {
                                temp = new sortedListNode;
                                temp->letter = word.at(i);
                                temp->occur = 1;
                                temp->next = NULL;
                                prior->next = temp;
                        }
                        else if (current->letter == word.at(i))
                        {
                                current->occur++;
                        }
                        else
                        {
                                temp = new sortedListNode;
                                temp->letter = word.at(i);
                                temp->occur = 1;
                                temp->next = current;
                                prior->next = temp;
                        }
                }
        }
        //END Corrected Code From Class
}

//Description: Print out linked list
//Pre-Condition: A filled linked list
//Post-Condition: A printed linked list
void printList(arraySortedList & list1)
{
        //Declare and Set Current Marker
        sortedListNode * current;
```

```
	current = list1.head;

	//If Good, Print all Characters
	if (current == NULL)
	{

	}
	else
	{
		std::cout << "List of Characters     : ";

		while (current != NULL)
		{
			std::cout << current->letter;
			current = current->next;
			std::cout << ", ";
		}
	}

	//Formatting
	std::cout << std::endl;

	//Reset Current Marker
	current = list1.head;

	//If Good, Print all Character Occurances
	if (current == NULL)
	{

	}
	else
	{
		std::cout << "Occurance of Characters: ";

		while (current != NULL)
		{
			std::cout << current->occur;
			current = current->next;
			std::cout << ", ";
		}
	}

	//Formatting
	std::cout << std::endl;
}
```

9. Updated Algorithm

1. Create Structure
    a. Name: sortedListNode
    b. Contents:
        i. character letter
        ii. integer occur
        iii. pointer sortedListNode
2. Create Class
    a. Name: arraySortedList
    b. Public:
        i. sortedListNode head
3. Create Function
    a. Name: fromString
    b. Parameters: Reference arraySortedList list, string word
    c. Return: VOID
    d. Method:
    e. Declare Movement Nodes of type sortedListNode
        i. prior
        ii. current
        iii. temp
    f. For Each Letter In Word
        i. If the list is empty
            1. Add new node
            2. Insert Letter
            3. Iterate Occurrence
            4. Move New Head
        ii. If new first element
            1. Add as the first element and link remaining members
        iii. If equals the first node
            1. Add 1 to occurrences
        iv. Else
            1. Set Prior to NULL
            2. Set Current to List Head
            3. While Current Isn't NULL and Current Letter < Word Letter
                a. Search list to find the element or add
        v. IF Current is NULL
            1. New Node
            2. Move new headw
        vi. Else If Found
            1. Add 1 to occurrences
        vii. If Element Higher Is Found

1. Add to list before the higher element

viii. If End of List is Reached

1. Not found, add as the last element

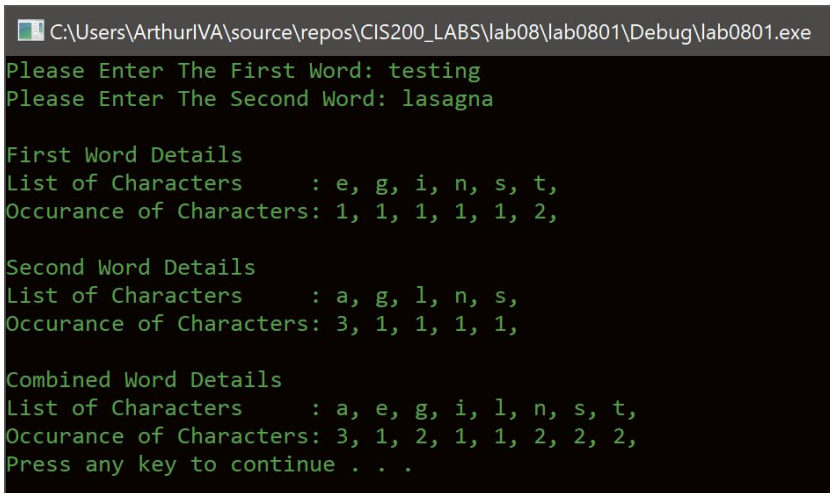
4. Create Function
    a. Name: printList
    b. Parameters: Reference arraySortedList list
    c. Return: VOID
    d. Method:
        i. Declare Current Marker
        ii. Set Current to Head of list
        iii. If current equals null
            1. Skip
        iv. Else
            1. While current not equal NULL
                a. Print Next Letter
        v. Set Current to Head of list
        vi. If current equals null
            1. Skip
        vii. Else
            1. While current not equal NULL
                a. Print Next Occurrence
5. Create Main
    a. Declare Holder Strings
        i. word1
        ii. word2
        iii. word3
    b. Declare Head of Linked Lists
        i. list1
        ii. list2
        iii. list3
    c. Prompt User For Input
        i. Fill word1
        ii. Fill word2
    d. fromString(list1, word1)
    e. printList(list1)
    f. fromString(list2, word2)
    g. printList(list2)
    h. Combine word1 and word2
    i. fromString(list3, word3)
    j. printList(list3)

10. Test Plan Version 3

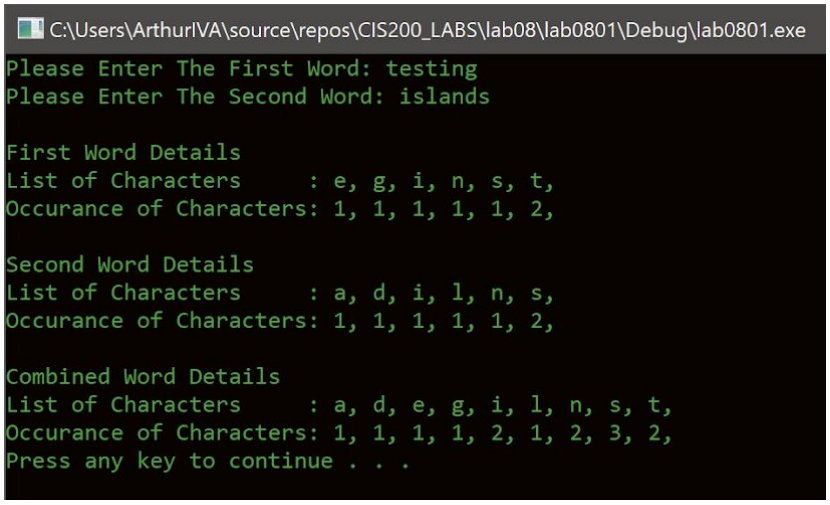
Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
Valid Data	1	Valid Word 1 Letters	testing	e, g, i, n, s, t	e, g, i, n, s, t	Pass
Valid Data	2	Valid Word 1 Occurrences	testing	1, 1, 1, 1, 1, 2	1, 1, 1, 1, 1, 2	Pass
Valid Data	3	Valid Word 2 Letters	lasagna	a, g, l, n, s	a, g, l, n, s	Pass
Valid Data	4	Valid Word 2 Occurrences	lasagna	3, 1, 1, 1, 1	3, 1, 1, 1, 1	Pass
Valid Data	5	Combined Above Letters	testing, lasagna	a, e, g, i, l, n, s, t	a, e, g, i, l, n, s, t	Pass
Valid Data	6	Combined Above Occurrences	testing, lasagna	3, 1, 2, 1, 1, 2, 2, 2	3, 1, 2, 1, 1, 2, 2, 2	Pass
Valid Data	7	Similar Words Letters	testing, islands	a, d, e, g, i, l, n, s, t	a, d, e, g, i, l, n, s, t	Pass
Valid Data	8	Similar Words Occurrences	testing, islands	1, 1, 1, 1, 2, 1, 2, 3, 2	1, 1, 1, 1, 2, 1, 2, 3, 2	Pass

11. Screenshots

Tests 1, 2, 3, 4, 5, 6



Tests 7, 8



12. Error Log

Error Type (Logic/Runtime)	Cause of Error	Solution to Error
Logic	Transferring Subroutine From Class into a Function gave tons of errors.	Make sure to pass by reference and include in a class, not just a struct.

13. Status

The program is working and performs all desired functions with linked lists.