1. **Program Statement**

This program uses arrays and linked lists to taken in input from user and organize it in an array and linked lists alphabetically with a counter of the number of times a particular letter occurs in the words.

1. **Requirements**
   1. **Assumptions**
      1. User inputs words when asked for words without any numbers inside
      2. User cannot type more than 100 characters of a word
      3. User types the word all lowercase with no spaces in the middle
   2. **Specifications**
      1. 2 arrays to store 2 words, and 3rd array to store the combination of two words
         1. Array 1 stores user’s first input of a word with letter in alphabetic order and updating occurrences
         2. Array 2 stores user’s second input of a word with letter in alphabetic order and updating occurrences
         3. Array 3 combines first and second words along with ordering them alphabetically and updating occurrences of each letter
      2. 2 lists to store 2 words using linked lists, and 3rd list to store the combination of two words
         1. List 1 stores user’s first input of a word with letter in alphabetic order and updating occurrences
         2. List 2 sores user’s second input of a word with letter in alphabetic order and updating occurrences
      3. Update occurrences each time looping to get and store a letter
2. **Decomposition Diagram**

|  |  |  |
| --- | --- | --- |
| **Main** | | |
| **Input** | **Process** | **Output** |
| Word 1 | Store the input in variable for word 1. Call function to store this in an array and list |  |
| Word 2 | Store the input in variable for word 2. Call function to store this in an array and list |  |

1. **Test Strategy**
   1. **Valid Data**
   2. **Invalid Data**
2. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | A valid word is given as an input that is less than 100 characters |  |  |  |  |
| Valid | 2 | User inputs a word with that has only 1 letter |  |  |  |  |
| Valid | 3 | User inputs a word with ALL characters |  |  |  |  |

1. **Initial Algorithm**
   1. In struct *sortedListNode()*
      1. Make char variable for letter, int variable for occurrences, and pointer variable for next to store location for linked lists.
   2. In main()
      1. Make variables
         1. String variables for word 1 and word 2
      2. Make arrays
         1. 1st and 2nd array holds 100 structs created above
         2. 3rd array holds 200 structs created above as it will hold combined array 1 and array 2
      3. Make 3 lists with pointers
      4. Ask user for 1st and 2nd words and store them in their variable
      5. Call functions to load these words in an array
      6. Call function to combine these words and store in 3rd array that is passed in
      7. Call function to print each array in an organized manner
      8. Call function to print each linked list in an organized manner
   3. In function to load arrays
      1. Set a Boolean variable contain to false initially
      2. Set a counter to 0
      3. For loop that loops each letter from ‘a’ to ‘a’
         1. For loop that loops from number 0 until the end of the word
            1. If the letter at current position is found

Set contain to true

Update occurrences for this letter

* + - 1. If contain = true
         1. Value at current array location is set to the current letter
      2. Set contain back to false
    1. For loop that loops through every index of the array
       1. If occurrence of any index is 0, then set that position’s letter to NULL
    2. Return count;
  1. In function to combine arrays
     1. Make two temporary empty string variables to store two words
     2. For loop that loops from 0 until a letter in the array is NULL
        1. For loop that loops from 0 until the total number of occurrences
           1. Add the letter at that position to the 1st string variable
     3. For loop that loops from 0 until a letter in the 2nd array is NULL
        1. For loop that loops from 0 until the total number of occurrences
           1. Add the letter at that position to the 2nd string variable
     4. Combine the two strings into one string and pass it in the function to load to an array
     5. Return count
  2. In function that prints array
     1. For loop that loops from 0 until the count variable passed in
        1. Print out the letter and occurrence of it at current location
  3. In *fromString()* function
     1. Make 3 pointer instances of struct
     2. Set a Boolean variable contain to false initially
     3. Initially set temp’s occurrence to 0 and letter to null
     4. Set curr to temp and head to curr
     5. For loop that goes through each letter in the alphabets
        1. For loop that loops from 0 until the end of the word
           1. If word is found, set the letter at current position of temp to that letter, update occurrences, and set contain to true
        2. If contain is true
           1. Make a new node for temp
           2. Set temp’s next value for position to null, occurrences to 0 and letter to null
        3. Set contain back to false
     6. Return head
  4. In *printlist()* function
     1. Make a new temp and its node
     2. Set temp to the node passed in
     3. Loop until temp’s next value for position is null
        1. Print out current temp’s letter and occurrence
        2. Set temp to the position value next

1. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | A valid word is given as an input that is less than 100 characters | “hello” | E – 1  H – 1  L – 2  0 – 1 |  |  |
| Valid | 2 | User inputs a word with that has only 1 letter | “hhhhh” | H – 5 |  |  |
| Valid | 3 | User inputs a word with ALL characters | “abcdefghijklmnopqrstuvwxyz” | Each alphabet is printed with occurrence 1 for each |  |  |

1. **Code**

// CIS200\_Lab7.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <iostream>

#include <string>

using namespace std;

struct sortedListNode {

char letter;

int occurrences;

sortedListNode \* next;

};

int load(sortedListNode array[], string word);

int combine(sortedListNode array1[], sortedListNode array2[], sortedListNode array3[]);

void print(sortedListNode array[], int count);

void printlist(sortedListNode \* list1);

sortedListNode \* fromString(string word1);

int main()

{

string word1, word2;

//arrays

sortedListNode array1[100];

sortedListNode array2[100];

sortedListNode array3[200];

sortedListNode \* list1;

sortedListNode \* list2;

sortedListNode \* list3;

for (int i = 0; i < 100; i++)

{

array1[i].letter = NULL;

array1[i].occurrences = 0;

array2[i].letter = NULL;

array2[i].occurrences = 0;

array3[i].letter = NULL;

array3[i].occurrences = 0;

}

cout << "Enter first word: ";

cin >> word1; //hello

cout << "Enter second word: ";

cin >> word2;

int count1 = load(array1, word1);

int count2 = load(array2, word2);

int count3 = combine(array1, array2, array3);

cout << endl << "Array 1: " << endl;

cout << "Letter\tOccurrences" << endl;

print(array1, count1);

cout << endl << "Array 2: " << endl;

cout << "Letter\tOccurrences" << endl;

print(array2, count2);

cout << endl << "Combined Array: " << endl;

cout << "Letter\tOccurrences" << endl;

print(array3, count3);

//Linked lists

list1 = fromString(word1);

list2 = fromString(word2);

cout << endl << "Linked List 1: " << endl;

cout << "Letter\tOccurrences" << endl;

printlist(list1);

cout << endl << "Linked List 2: " << endl;

cout << "Letter\tOccurrences" << endl;

printlist(list2);

system("pause");

return 0;

}

int load(sortedListNode array[], string word)

{

bool contain = false;

int count = 0;

for (char i = 'a'; i <= 'z'; i++)

{

for (int x = 0; x < word.length(); x++)

{

if (word[x] == i) {

contain = true;

array[count].occurrences++;

}

}

if (contain) {

array[count].letter = i;

count++;

}

contain = false;

}

for (int i = 0; i < 100; i++)

{

if (array[i].occurrences == 0)

{

array[i].letter = NULL;

}

}

return count;

}

int combine(sortedListNode array1[], sortedListNode array2[], sortedListNode array3[])

{

string temp = "";

string temp2 = "";

for (int i = 0; array1[i].letter != NULL; i++)

{

for (int x = 0; x < array1[i].occurrences; x++) {

temp += array1[i].letter;

}

}

for (int i = 0; array2[i].letter != NULL; i++)

{

for (int x = 0; x < array2[i].occurrences; x++) {

temp2 += array2[i].letter;

}

}

string finaltemp = temp + temp2;

int count = load(array3, finaltemp);

return count;

}

void print(sortedListNode array[], int count)

{

for (int i = 0; i < count; i++)

{

cout << array[i].letter << "\t" << array[i].occurrences << endl;

}

}

sortedListNode \* fromString(string word1)

{

sortedListNode \* head = NULL;

sortedListNode \* curr = NULL;

sortedListNode \* temp = new sortedListNode;

temp->next = NULL;

bool contain = false;

temp->occurrences = 0;

curr = temp;

head = curr;

for (char x = 'a'; x <= 'z'; x++) {

for (int i = 0; i < word1.length(); i++)

{

if (word1[i] == x) {

temp->letter = word1[i];

temp->occurrences++;

contain = true;

}

}

if (contain) {

temp = new sortedListNode;

temp->next = NULL;

temp->occurrences = 0;

curr->next = temp;

curr = temp;

}

contain = false;

}

return head;

}

void printlist(sortedListNode \* list1)

{

sortedListNode \* temp = new sortedListNode;

temp = list1;

while (temp->next != NULL)

{

cout << temp->letter << "\t" << temp->occurrences << endl;

temp = temp->next;

}

}

1. **Updated Algorithm**
   1. In struct *sortedListNode()*
      1. Make char variable for letter, int variable for occurrences, and pointer variable for next to store location for linked lists.
   2. In main()
      1. Make variables
         1. String variables for word 1 and word 2
      2. Make arrays
         1. 1st and 2nd array holds 100 structs created above
         2. 3rd array holds 200 structs created above as it will hold combined array 1 and array 2
      3. Make 3 lists with pointers
      4. Ask user for 1st and 2nd words and store them in their variable
      5. Call functions to load these words in an array
      6. Call function to combine these words and store in 3rd array that is passed in
      7. Call function to print each array in an organized manner
      8. Call function to print each linked list in an organized manner
   3. In function to load arrays
      1. Set a Boolean variable contain to false initially
      2. Set a counter to 0
      3. For loop that loops each letter from ‘a’ to ‘a’
         1. For loop that loops from number 0 until the end of the word
            1. If the letter at current position is found

Set contain to true

Update occurrences for this letter

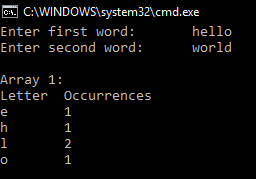
* + - 1. If contain = true
         1. Value at current array location is set to the current letter
      2. Set contain back to false
    1. For loop that loops through every index of the array
       1. If occurrence of any index is 0, then set that position’s letter to NULL
    2. Return count;
  1. In function to combine arrays
     1. Make two temporary empty string variables to store two words
     2. For loop that loops from 0 until a letter in the array is NULL
        1. For loop that loops from 0 until the total number of occurrences
           1. Add the letter at that position to the 1st string variable
     3. For loop that loops from 0 until a letter in the 2nd array is NULL
        1. For loop that loops from 0 until the total number of occurrences
           1. Add the letter at that position to the 2nd string variable
     4. Combine the two strings into one string and pass it in the function to load to an array
     5. Return count
  2. In function that prints array
     1. For loop that loops from 0 until the count variable passed in
        1. Print out the letter and occurrence of it at current location
  3. In *fromString()* function
     1. Make 3 pointer instances of struct
        1. Set head and curr to null
        2. Set temp to a new node
     2. Set temp’s next value for location to null
     3. Set a Boolean variable contain to false initially
     4. Initially set temp’s occurrence to 0 and letter to null
     5. Set curr to temp and head to curr
     6. For loop that goes through each letter in the alphabets
        1. For loop that loops from 0 until the end of the word
           1. If word is found, set the letter at current position of temp to that letter, update occurrences, and set contain to true
        2. If contain is true
           1. Make a new node for temp
           2. Set temp’s next value for position to null, occurrences to 0 and letter to null
           3. Set curr’s next value for position to temp’s location
           4. Set curr to temp
        3. Set contain back to false
     7. Return head
  4. In *printlist()* function
     1. Make a new temp and its node
     2. Set temp to the node passed in
     3. Loop until temp’s next value for position is null
        1. Print out current temp’s letter and occurrence
  5. Set temp to the position value next

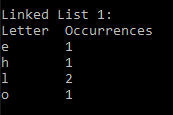
1. **Test Plan Version 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | A valid word is given as an input that is less than 100 characters | “hello” | E – 1  H – 1  L – 2  0 – 1 | E – 1  H – 1  L – 2  0 – 1 | Pass |
| Valid | 2 | User inputs a word with that has only 1 letter | “hhhhh” | H – 5 | H – 5 | Pass |
| Valid | 3 | User inputs a word with ALL characters | “abcdefghijklmnopqrstuvwxyz” | Each alphabet is printed with occurrence 1 for each | Each alphabet is printed with occurrence 1 for each | Pass |

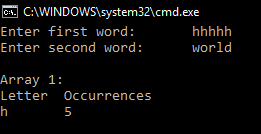
1. **Screenshots**

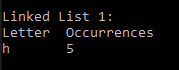
Valid Test Case 1:



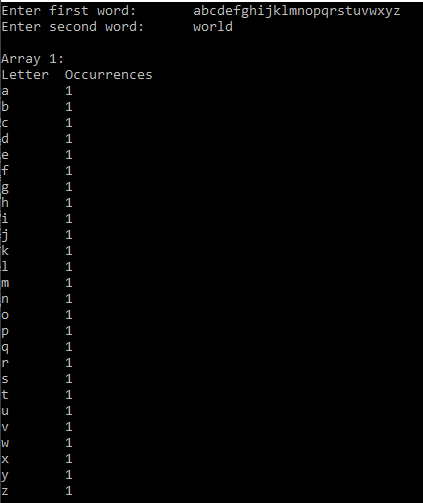


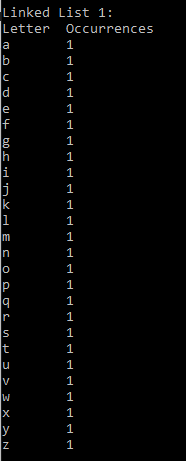
Valid Test Case 2:





Valid Test Case 3:





1. **Status**

Most of the program works correctly with assumptions in mind. Only thing that is not programmed is combining two lists using an operator overloading, which I had hard time doing under limited time.