Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2022

Homework 2 – Simple Word Processing with Linked Lists

Due: 23/3/2022, Wednesday, 21:00

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You can NOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism and homework trading will not be tolerated!

Introduction

In this homework, you are asked to implement a simple word processor that will read two words from two different files and store them in two linked lists character by character. Your program will also process these lists by reading certain commands such as for inserting, deleting characters and finding the level of similarity between two words. After each command, some outputs regarding the operation will be displayed. The program details will be explained in the subsequent sections.

The Data Structure to be used

In this homework, you **must** use linked list (regular one-way linked list) as your main data structure. The node struct of this list must have the following data members (if you want, you can add constructors to the struct).

```
struct node
{
      char ch;
      node * next;
};
```

Figure 1. The minimal data members that the node should have

In this struct, **`ch`** stores one character of a word, and **`next`** pointer points to the next node in the linked list.

You are not allowed to use arrays, vectors or any similar containers (including files) in this homework; all data must be stored and processed within the linked list structure. Moreover, you are not allowed to use string to store the data that you will read from the files; strings can only be used to read the file names and the commands/arguments from the keyboard.

The input data will be given to the program using text files. A screenshot of a sample input file named in1.txt and the final structure of the linked list after reading the letters from this file are shown in

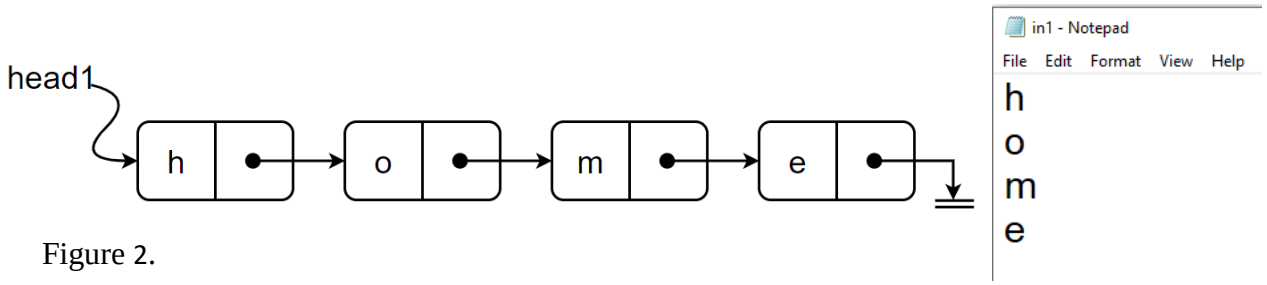


Figure 2.

Figure 2. A visualization of a linked list (pointed by `head1`) after reading the data from input file "in1.txt"(screenshot on the right) and storing all of the characters in the nodes

A screenshot of a sample input file named in1.txt and the final structure of the linked list after reading the letters from this file are shown in Figure 3.

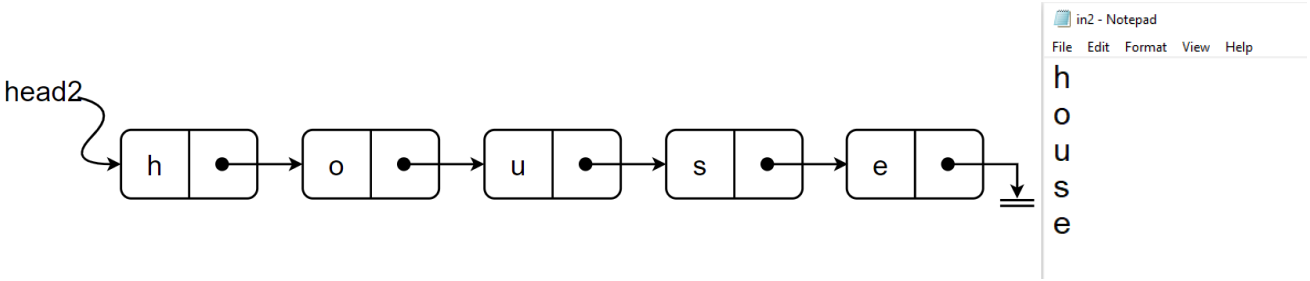


Figure 3 A visualization of a linked list (pointed by `head2`) after reading the data from input file "in2.txt"(screenshot on the right) and storing all of the characters in the nodes

Reading the Files and Creating the Linked Lists

Your program will start by asking the user to enter two file names, one by one. For each of these files, first you should check if it is opened successfully or not. If not opened, you have to read another file name.

Then, your program should read the contents of these two files and store each file's content in a separate linked list character by character as described above.

Each file has one word, which is lined up vertically and each line has only one character. This is an assumption and you do not have to make an input check for this. We also assume that:

- The file does not contain empty lines or whitespaces.
- The file contains only lowercase English letters (i.e., from *a* to *z*).
- The file may contain duplicate letters.
- The file is not empty and may contain any number of letters.

These are the assumptions and you do not need to make an input check for the content of the files (but there will be some input checks for the commands and arguments as detailed below).

In order to store the contents of two files, you need to have two different linked lists with different head pointers. For each of the files, you will read file's content line by line (i.e. letter by letter) and append each letter read to the end of the corresponding linked list. After that, your linked lists become ready to be processed with some commands as described below.

Operations/Commands to be Performed on the Linked Lists

After reading the data from the two files and storing them in two linked lists, first, you have to display the contents of the lists one by one. After that, your program will keep asking the user to enter a command followed by its arguments, if any. Basically, these commands are for *inserting* and *deleting* characters from the linked lists, finding *similarity* between two lists, and for *exiting* the program. Insertion and deletion commands also indicate the linked list number that this operation will be performed on (i.e., whether it is the linked list 1 or the linked list 2). Thus, we have a total of six commands that your program should support: **del1**, **del2**, **insert1**, **insert2**, **sim**, and **exit**.

- **del1** and **del2** commands are followed by one argument, which is the letter to be deleted from the corresponding list. This command will delete all of the nodes that contain the given letter in the argument from the specified linked list. For example, if the list contains `l l a m a` and if the argument is `l`, then the list will contain `a m a`. As another example, if another list contains `g o o g l e`, and if the argument was `g` then the list will contain `o o l e`.
  If the letter to be deleted does not exist in the list, then nothing will be deleted and a message that explains this has to be displayed. You have to display an output about the operation details and the content of the list after deletion. Please see the sample runs for example outputs. Other details about deletion are as follows.
  - **del1** will apply the deletion operation on the linked list 1.
  - **del2** will apply the deletion operation on the linked list 2.
  - The command and argument are separated by at least one whitespace and the argument is only one character. These are assumptions and you do not need to make input check for those.
  - The argument should be a lowercase letter and you have to do an input check for this.
- **insert1** and **insert2** commands are followed by two arguments, which have to be lowercase English letters. The first argument indicates the letter to be inserted after the first occurrence (just after first occurrence, not after all of the occurrences) of a node that contains the letter specified as the second argument. If the linked list does not contain a node with the second argument, then the program should insert the node that contains the first argument to the beginning of the linked list (i.e., it will become the *head* node). For example, if the list contains `a l b a l a` and if the arguments are `t` and `l`, then after insertion the list will contain `a l t b a l a`. As another example, suppose you insert once again to this updated list with arguments `b` and `s`; the list will contain

`b a l t b a l a` after insertion; here we inserted `b` to the beginning since there was no `s`. You have to display an output about the operation details and the content of the list after insertion. Please see the sample runs for example outputs. Other details about insertion are as follows.

  - **insert1** will apply the insertion operation on linked list 1.
  - **insert2** will apply the insertion operation on linked list 2.
  - The command and the arguments are separated by at least one whitespace and each argument is only one character. These are assumptions and you do not need to make input check for those.
  - The arguments should be lowercase letters and you have to do input check for this.

- **sim** does not take any argument. It is an abbreviation for *similarity*, and when the user enters this command the program will count and display the total number of matching letters in the same positions in both lists. For example, if the first list contains `a l k i` and the second list contains `e l t i k`, the similarity value is 2 since the letters `l` and `i` match in the same positions. This command should display this similarity value and also the current contents of the lists. Here be careful that the lists may have different sizes, so you have to stop checking once you reach `NULL` in one of them.
- **exit** command indicates that the program will end. At this point, the program will stop asking the user for more commands and clear both of the lists by properly deleting all the memory allocated for them.

If the user enters an invalid command and/or invalid argument(s), your program should give an error message and continue by reading new commands and arguments.

If the command is entered correctly, you may assume that the number of arguments is also correct (no input check needed); that means please assume that there are two arguments for insertion, one argument for deletion and no argument for similarity and exit. The only input check for the arguments is that they have to be lowercase letters (as mentioned above).

You have to ignore all of the extra characters that might appear after an invalid command in the input line so that the next command is read fresh. You can do so by using the command `cin.ignore(numeric_limits<streamsize>::max(),'\n');`

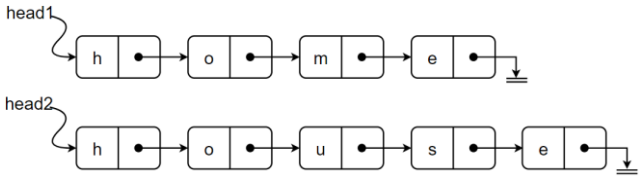
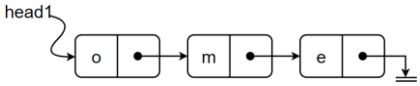
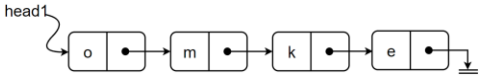
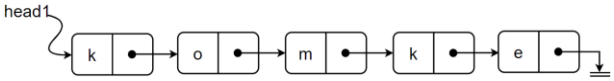
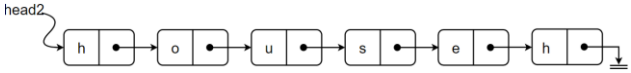
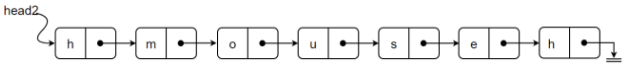
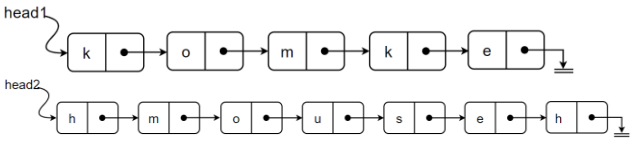
CAUTION

One of the lists or both of them may become empty during the execution of the program due to delete operations. Your program should keep processing commands in such a case as well. Please note that if one or both of the lists is/are empty, then similarity value should be zero, by definition of similarity.

Walk-through Example

A walk-through run of the program is shown in Table 1. The example given in the table assumes that data from in1.txt and in2.txt have been successfully read and stored in two different linked lists.

Table 1 A walk-through example

Command	Arguments (if any)	Effect on the corresponding linked list	Explanation
sim			**sim** command will display the similarity value (2 in this case) and the contents of the linked lists without changing them.
del1	h		Deleting **all** occurrences of letter 'h' from linked list 1
insert1	k m		Inserting a node with letter 'k' **after** the node with letter 'm' in linked list 1
insert1	k f		Inserting a node with letter 'k' at the beginning (i.e., to the head) of the linked list because no node with letter 'f' exists in linked list 1
insert2	h e		Inserting a node with letter 'h' **after** the node with letter 'e' in linked list 2
insert2	m h		Inserting a node with letter 'm' **after** the **first occurrence** of a node with letter 'h'
sim			**sim** command will display the similarity value (0 in this case) and the contents of the linked lists without changing them.
exit			The program will stop asking for further commands and will delete the two lists and terminate.

Sample Runs

Sample runs are given below, but these are not comprehensive, therefore you have to consider **all possible cases** to get full mark. Inputs are shown in **bold** and *italic*.

Sample Run 1:

```
Please enter the file 1 name: in1
Please enter the file 1 name: in1.txt
Please enter the file 2 name: in2.text
Please enter the file 2 name: in2.txt
List1: home
List2: house

Enter a command and its corresponding argument(s), if any: go sim
Invalid command.

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1: home
List2: house
There is/are 2 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: insert u o
Invalid command.

Enter a command and its corresponding argument(s), if any: insert 1 u o
Invalid command.

Enter a command and its corresponding argument(s), if any: insert1 u o
Inserting u after o in List1.
The node with u has been inserted after the first occurrence of a node with o value.
The current content of List1 is: houme

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1: houme
List2: house
There is/are 4 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: del2 E
Invalid arguments.

Enter a command and its corresponding argument(s), if any: del2 e
Deleting all occurrences of e in List2.
The current content of List2 is: hous

Enter a command and its corresponding argument(s), if any: del1 k
No deletion as the value k was not found in the list

Enter a command and its corresponding argument(s), if any: del1 m
Deleting all occurrences of m in List1.
The current content of List1 is: houe

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1: houe
List2: hous
There is/are 3 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: insert1 h u
```

File with name '***in1***' does not exist. So, the program asks again for a valid file name. Same for file 2 name, the program kept asking for a valid file name

go sim is considered as invalid command as the first word is not one of the valid commands

insert 1 u o is invalid command as the valid commands for insertion into the linked list 1 is ***insert1*** as a single word. Same for ***insert2*** with linked list 2

del2 E is invalid argument as ***E*** is uppercase

A message to show that no deletion has been done when the item to be deleted is not in the list

```
Inserting h after u in List1.
The node with h has been inserted after the first occurrence of a node with u value.
The current content of List1 is: houhe

Enter a command and its corresponding argument(s), if any: insert2 j o
Inserting j after o in List2.
The node with j has been inserted after the first occurrence of a node with o value.
The current content of List2 is: hojus

Enter a command and its corresponding argument(s), if any: insert1 b h
Inserting b after h in List1.
The node with b has been inserted after the first occurrence of a node with h value.
The current content of List1 is: hbouhe

Enter a command and its corresponding argument(s), if any: insert1 g e
Inserting g after e in List1.
The node with g has been inserted after the first occurrence of a node with e value.
The current content of List1 is: hbouheg

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1: hbouheg
List2: hojus
There is/are 2 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: exit
Clearing the two lists and saying Goodbye!
```

Sample Run 2:

```
Please enter the file 1 name: in11.txt
Please enter the file 2 name: in8.txt
List1: llama
List2: bubble
```

Displaying the contents of the lists one by one, after reading the data from the two files and storing them in two linked lists.

```
Enter a command and its corresponding argument(s), if any: del1 a
Deleting all occurrences of a in List1.
The current content of List1 is: llm

Enter a command and its corresponding argument(s), if any: del2 b
Deleting all occurrences of b in List2.
The current content of List2 is: ule

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1: llm
List2: ule
There is/are 1 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: del1 l
Deleting all occurrences of l in List1.
The current content of List1 is: m

Enter a command and its corresponding argument(s), if any: del1 m
Deleting all occurrences of m in List1.
The current content of List1 is:

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1:
```

```
List2: ule
There is/are 0 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: del2 u
Deleting all occurrences of u in List2.
The current content of List2 is: le

Enter a command and its corresponding argument(s), if any: del2 l
Deleting all occurrences of l in List2.
The current content of List2 is: e

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1:
List2: e
There is/are 0 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: del2 e
Deleting all occurrences of e in List2.
The current content of List2 is:

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1:
List2:
There is/are 0 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: insert1 a h
Inserting a after h in List1.
The node with h value does not exist. Therefore, Inserting a at the beginning of the list.
The current content of List1 is: a

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1: a
List2:
There is/are 0 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: del2 j
No deletion as the value j was not found in the list

Enter a command and its corresponding argument(s), if any: insert2 o p
Inserting o after p in List2.
The node with p value does not exist. Therefore, Inserting o at the beginning of the list.
The current content of List2 is: o

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1: a
List2: o
There is/are 0 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: exit
Clearing the two lists and saying Goodbye!
```

List 1 became empty after several **del1** operations

List 2 became empty after several **del2** operations

Inserting to list 1 after it became empty. Your program should support that case.

Inserting to list 2 after it became empty. Your program should support that case.

Sample Run 3:

```
Please enter the file 1 name: in5.txt
Please enter the file 2 name: in6.txt
List1: wow
List2: odd
```

```
Enter a command and its corresponding argument(s), if any: del1 w
Deleting all occurrences of w in List1.
The current content of List1 is: o

Enter a command and its corresponding argument(s), if any: del1 o
Deleting all occurrences of o in List1.
The current content of List1 is:

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1:
List2: odd
There is/are 0 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: del2 o
Deleting all occurrences of o in List2.
The current content of List2 is: dd

Enter a command and its corresponding argument(s), if any: del2 d
Deleting all occurrences of d in List2.
The current content of List2 is:

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1:
List2:
There is/are 0 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: insert1 m b
Inserting m after b in List1.
The node with b value does not exist. Therefore, Inserting m at the beginning of the list.
The current content of List1 is: m

Enter a command and its corresponding argument(s), if any: insert2 k r
Inserting k after r in List2.
The node with r value does not exist. Therefore, Inserting k at the beginning of the list.
The current content of List2 is: k

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1: m
List2: k
There is/are 0 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: insert2 m u
Inserting m after u in List2.
The node with u value does not exist. Therefore, Inserting m at the beginning of the list.
The current content of List2 is: mk

Enter a command and its corresponding argument(s), if any: sim
Displaying the contents of the two lists:
List1: m
List2: mk
There is/are 1 letter(s) matched in the same positions in both lists.

Enter a command and its corresponding argument(s), if any: exit
Clearing the two lists and saying Goodbye!
```

Similarity value between an empty and a non-empty list is zero.

Similarity value between two empty lists is zero.

Some Important Rules

In order to get a full credit, your programs must be efficient and well presented, presence of any redundant computation or unnecessary memory usage or bad indentation, or missing, irrelevant comments are going to decrease your grades. You also have to use understandable identifier names, informative introduction and prompts. Modularity is also important; you have to use functions wherever needed and appropriate.

Since you will use dynamic memory allocation in this homework, it is very crucial to properly manage the allocated area and return the deleted parts to the heap whenever appropriate. Inefficient use of memory may reduce your grade.

When we grade your homework, we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we may run your programs in *Release* mode and **we may test your programs with very large test cases**. Of course, your program should work in *Debug* mode as well.

Please do not use any non-ASCII characters (Turkish or other) in your code.

You are allowed to use sample codes shared with the class by the instructor and TAs. However, you cannot start with an existing .cpp or .h file directly and update it; you have start with an empty file. Only the necessary parts of the shared code files can be used and these parts must be clearly marked in your homework by putting comments like the following. Even if you take a piece of code and update it slightly, you have to put a similar marking (by adding "`and updated`" to the comments below.

```
/* Begin: code taken from ptrfunc.cpp */
```

…

```
/* End: code taken from ptrfunc.cpp */
```

What and where to submit (PLEASE READ, IMPORTANT) – Same as before

You should prepare (or at least test) your program using MS Visual Studio C++. We recommend using 2012 version; however, if you use another version provided by Sabancı University software repository, it is OK as long as you specify which version you use at the beginning of your program. We will use the standard C++ compiler and libraries of the abovementioned platform while testing your homework. It'd be a good idea to write your name and last name in the program (as a comment line of course). Using other platforms (Xcode, VSCode, etc.) is risky and may cause some incompatibility problems.

Submissions guidelines are below. Some parts of the grading process might be automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.

Name your cpp file that contains your main program using the following convention:

"SUCourseUserName_YourLastname_YourNames_HWnumber.cpp"

Your SUCourse user name is your SUNet user name which is used for checking sabanciuniv e-mails (not the numeric one). Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

Cago_Ozbugsizkodyazaroglu_Caglayan_hw2.cpp

Actually, it does not matter whether you use uppercase of lowercase letters in the file names.

In some homework assignments, you may need to have more than one .cpp or .h files to submit. In this case, add informative phrases after the hw number. However, do not add any other character or phrase to the file names. Sometimes, you may want to use some user defined libraries (such as strutils of Tapestry); in such cases, you have to provide the necessary .cpp and .h files of them as well. If you use standard C++ libraries, you do not need to provide extra files for them.

In some homework assignments, you may need to have more than one .cpp or .h files to submit. In this case, use the same filename format but add informative phrases after the hw number (e.g. Cago_Ozbugsizkodyazaroglu_Caglayan_hw2_myfuncs.cpp or Cago_Ozbugsizkodyazaroglu_Caglayan_hw2_myfuncs.h). However, do not add any other character or phrase to the file names. Sometimes, you may want to use some user defined libraries (such as strutils of Tapestry); in such cases, you have to provide the necessary .cpp and .h files of them as well by using the same naming convention mentioned above. If you use standard C++ libraries, you do not need to provide extra files for them.

You will receive zero if your compressed zip file does not expand or it does not contain the correct files. The naming convention of the zip file is the same. The name of the zip file should be as follows:

SUCourseUserName_YourLastname_YourNames_HWnumber.zip

For example, zubzipler_Zipleroglu_Zubeyir_hw2.zip is a valid name, but

Hw2_hoz_HasanOz.zip, HasanOzHoz.zip

are **NOT** valid names.

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
Albert Levi, Ahmed Salem