Sabancı University

Faculty of Engineering and Natural Sciences

CS204 Advanced Programming

Spring 2024

Homework 2 – Matching Pairs using Linked Lists

Due: 20/3/2024, 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 going to implement a program to match pairs of individuals according to their "like" relationships. You will read the data from an input text file and store the information in two separate singly linked lists (regular one-way linked lists). The "Data Structures" section discusses the details of these linked lists. While reading the data from the input text file, you will manipulate the two singly linked lists according to some rules that are mentioned in the "Program Flow" section.

Input File Structure

The input text file contains a sequence of lines; each line is in the form of:

`subject` likes *`object`*

where `subject` and `object` are names and "likes" is a fixed verb that exists in every line connecting `subject` and `object` names. In other words, each line in the file represents a one-way like-relation of the `subject` towards the `object`. A particular name can appear more than once in the file as both subject and object.

It is assumed that `subject`, as well as `object`, names do not contain any spaces; i.e., each of them will be a single word. They are assumed to be all lowercase letters. It is also assumed that `subject` and `object` of a line will not be the same (i.e. no narcissistic relationships :)). You do not need to check these assumptions in your code.

Note: it is guaranteed that the data inside all the input text files that will be used in this homework (including the ones that will be used in the hidden test cases) will be in the format mentioned above, so you do **not** need to make any input check for that. In other words, once opened successfully, there is no need to check for the validity of the data inside the file, and you can safely start reading and processing the data in the file according to the rules that are mentioned in the "Program Flow" section below.

Data Structures

You should use two singly linked lists to store and manipulate the data that you will read from the input file. Each linked list has its own type of node structure that is explained subsequently.

***likes* linked list:**

The first linked list is what we call *likes* linked list. It is a singly linked list with a node struct mentioned below (`NodeLike`). Each node in this list contains three pieces of information, namely: `subject`, `object`, and `next`. Among them, `subject` is the first word that comes before the "likes" verb of a line in the input file; `object` is the third word of the line that comes after the verb "likes"; and finally, `next` is a pointer (of the same type of the node) to the next node in the *likes* linked list.

```
struct NodeLike
{
    string subject;
    string object;
    NodeLike *next;
};
```

Figure 1 shows a generic *likes* linked list with three nodes and with head named as `headLikes`. In the actual program, this linked list may have any number of nodes or may become empty.

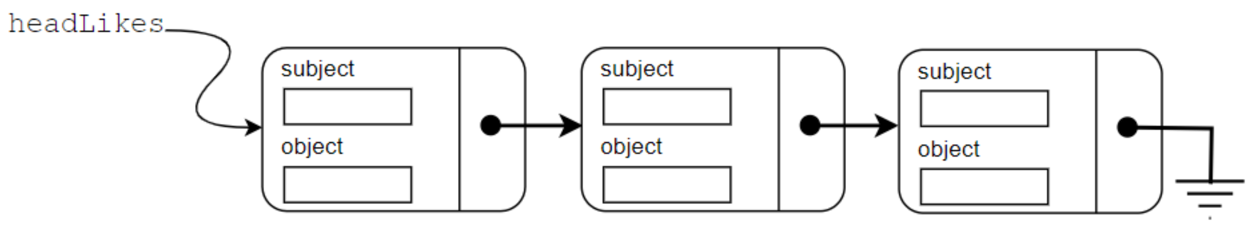


Figure 1: A generic *likes* linked list with three nodes.

***taken* linked list:**

The second linked list is called *taken* linked list. This list is also a singly linked list, but with a different node structure and for a different purpose. The node struct of *taken* linked list is shown below (`NodeTaken`). Each node of this list contains two fields: `name` and `next`. The `name` field stores a person's name, and `next` is a pointer (of the same type of the node) to the next node in the *taken* linked list.

```
struct NodeTaken
{
    string name;
    NodeTaken *next;
};
```

Figure 2 shows a generic *taken* linked list with three nodes and with head named as `headTaken`. Please note that this is just an example list; the actual one to be used in the program may have any number of nodes or may be empty.

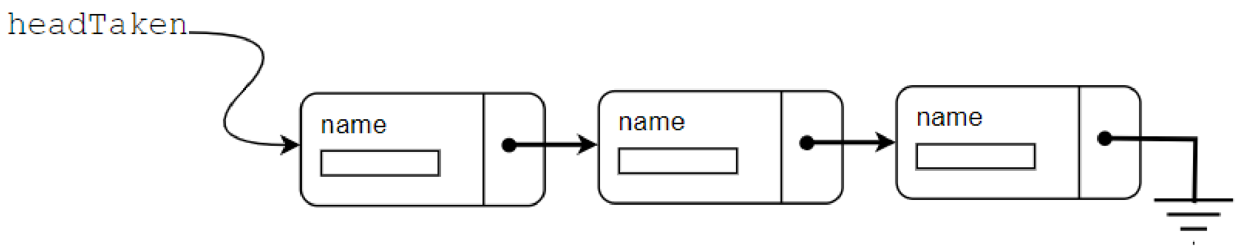


Figure 2: A generic "taken" linked list with three nodes.

Both node structs given above may be used directly or by adding constructors.

You do <u>not</u> have to implement these structures as class(es). However, if you prefer to write one or more classes in the homework, you can do so provided that you perform a proper object oriented design.

You **cannot use arrays, vectors, etc.** with the aim of avoiding linked list usage. In other words, you have to implement the requested data structures as linked lists. If you do not follow this rule (i.e. if you use another data structure), your homework will not be graded.

Program Flow

Your program starts by asking the user to enter a file name for the input text file. Then, your program will try to open the file and check if it is opened successfully or not. If, for some reason, the program could not open the file, it should keep reading another file name until opened successfully.

After successfully opening the file, your program should read the file line by line, and process the data. The rules for processing the data and populating the linked lists will be explained in detail below. Your program will end after reading and processing all the lines in the input text file. Before finishing the program, make sure that you **deallocate all the dynamically allocated memory** and close the file.

Rules for processing the data and using the linked lists

After reading a line (which corresponds to a like-relationship between a `subject` and an o`bject`), you will insert this relationship as a node to the *likes* list under certain conditions. On the other hand, *taken* linked list will be populated by the names that are in a mutual like relationship as both subject and object.

The rules of processing of a "`subject` likes `object`" line of the input file are detailed below.

1. If any of the subject or the object (or both) of this line is already in the *taken* linked list, no new node should be added to the *likes* linked list. Moreover, no change will be done in the *taken* linked list as well.
2. If subject-object pair is already in the *likes* linked list as is, this relationship will be considered as redundant and it should not be added to the *likes* linked list. Nothing will change in the *taken* list as well.
3. Otherwise, if there exists a reverse relationship in the *likes* linked list such that subject of the input line is the object of a node and object of the input line is subject of the same node (e.g. the input line is "mecnun likes leyla" and there exists a node in the *likes* list with subject="leyla" and object="mecnun"), then there is a match between this subject and the object. In such a case, the input subject-object pair will not be added to the *likes* linked list. Moreover, (i) you should delete each node in the *likes* linked list that contains either the subject or the object of that matched relationship; (ii) Then, these matched subject and object should be inserted into the *taken* linked list in ascending alphabetical order.
4. Otherwise, if only the subject of the input line exists in at least one node in the *likes* linked list as subject, then the program should insert a new node containing input subject-object pair in the *likes* linked list *just after* the last node that contains this

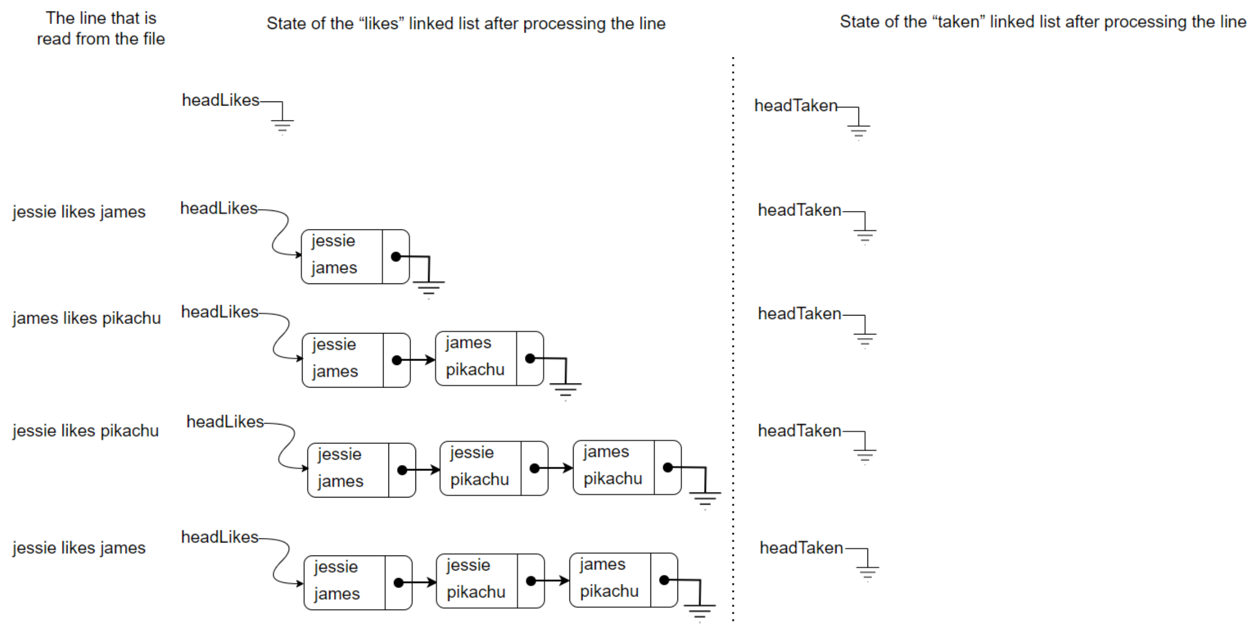
subject's name in its subject field. In this way, we keep the subjects in groups in the *likes* linked list. No alphabetical order here.

5. If the subject is not in the *likes* linked list and is allowed to be added to it, (i.e., it is not in the *taken* linked list), then it should be added to the end of the *likes* linked list.

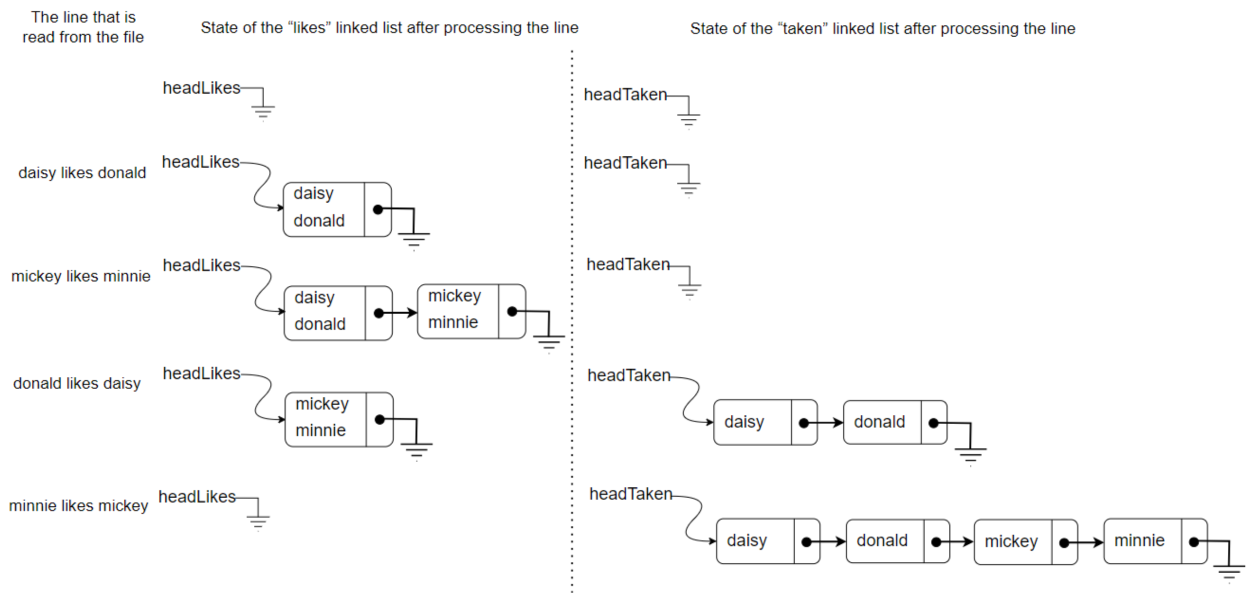
The order of handling the cases above might matter since some cases are not mutually exclusive. If you follow the order mentioned above in your algorithmic logic, you do not see any discrepancy between your output and the output in the sample runs.

Before processing an input line, you should display the content of that line with a counter at the beginning. Moreover, there are some fixed messages to be displayed corresponding to the cases mentioned above. After processing each line of the input text file, your program should print the contents of both the "likes" and the "taken" linked lists. Please refer to the "Sample Runs" section for the format and the content of these outputs.

Below is a running example for file *s1.txt* (provided in the homework package).


The line that is read from the file
State of the "likes" linked list after processing the line
State of the "taken" linked list after processing the line
headLikes
headTaken
jessie likes james
jessie james
james likes pikachu
james pikachu
jessie likes pikachu
jessie pikachu
jessie likes james

Another running example for file *s2.txt* (provided in the homework package).


The line that is read from the file
State of the "likes" linked list after processing the line
State of the "taken" linked list after processing the line
headLikes
headTaken
daisy likes donald
daisy
donald
mickey likes minnie
mickey
minnie
donald likes daisy
minnie likes mickey

Some Important Programming Rules

Please do not use any non-ASCII characters (Turkish or other) in your code (not even as comments). And also do not use non-ASCII characters in your file and folder names. We really mean it; otherwise, you may encounter some errors.

In order to get a full credit, your programs must be efficient and well presented, presence of any redundant computation 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.

You are **not** allowed to use codes found somewhere online. These restrictions include code repositories, sites like geeksforgeeks, codes generated by GenAI tools, etc. Moreover, you are not allowed to use any statement, command, concept, topic that has not been covered in CS201 and CS204 (until now). Trying to find help online would generally cause such a problem. Thus, you always have to find help within the course material.

You cannot use break and continue statements. Global variables cannot be used either.

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 to 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 lab1.cpp */

…

/* End: code taken from lab1.cpp */
```

Since CodeRunner configuration in this homework does not allow to use extra non-standard C++ library/function/class files, if you want to use such functions/classes covered in the classes (such as strutils), you will need to copy the necessary declarations and implementations to your main program by following the citation rules mentioned above.

Submission Rules and Some Grading Tips (PLEASE READ, IMPORTANT)

It'd be a good idea to write your name and last name in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). For example, if your full name is "Satılmış Özbugsızkodyazaroğlu", then you must type it as follows:

// Satilmis Ozbugsizkodyazaroglu

We use CodeRunner in SUCourse+ for submission. No other way of submission is possible. Since the functionality part of the grading process will be automatic, you have to strictly follow these guidelines; otherwise we cannot grade your homework.

The advantage CodeRunner it is that you will be able to test your code against sample test cases. However, the output should be exact, but the textual differences between the correct output and yours can be highlighted (by pressing "show differences" button) on the submission interface.

You should copy the full content of the main .cpp file and paste it into the specified “Answer” area in the relevant assignment submission page on SUCourse+. Then you can test your code via CodeRunner against the sample runs (by pressing the "Check" button). Since you will not upload a file, your local cpp file name is not important.

Even any tiny change in the output format will result in your grade being zero (0) for that particular test case, so please test your programs yourself, and against the sample runs that are available at the relevant assignment submission page on SUCourse+ (CodeRunner).

In the CodeRunner, there are some visible and invisible (hidden) test cases. You will know whether your code has successfully passed all the test cases or not before submitting your code. However, we keep our rights to add more test cases in the grading process after the submission. **Thus, please make sure that you have read this documentation carefully and**

covered/tested all possible cases, even some other cases you may not have seen on CodeRunner or the sample runs. Due to these reasons, **your final grade may conflict with what you have seen on CodeRunner**. We will also **manually** check your code against some criteria, comments, indentations and so on; hence, please do not object to your grade based on the **CodeRunner** results, but rather, consider every detail on this documentation.

You have to manually "Submit" after you test and finish with your code (there is no automatic submission). There is no re-submission. That means, after you submit, you cannot take it back. On the other hand, this does not mean that you have one shot to test CodeRunner. You don't have to complete your task in one time, you can continue from where you left last time, but you should not press submit before finalizing it. Therefore, you should make sure that it's your final solution version before you submit it. Also, we still do not suggest that you develop your solution on CodeRunner but rather on your IDE on your computer.

Last, even if you cannot completely finish your homework, you can still submit.

Please see the syllabus for general homework grading issues.

Plagiarism

Plagiarism is checked by automated tools, and we are very capable of detecting such cases. Be careful with that. Exchange of abstract ideas are totally okay but once you start sharing the code with each other, it is very probable to get caught by plagiarism. So, do NOT share any part of your code to your friends by any means or you might be charged as well, although you have done your homework by yourself.

Homework are to be done personally and you have to submit your own work. **Cooperation will NOT be counted as an excuse.**

Our experience shows that code taken from online sources or generated by AI tools also show resemblance; thus if you try to get such help, you also may be charged by plagiarism with a person that you do not know.

In case of plagiarism, the rules written in the Syllabus apply.

Sample Runs

Sample runs are given below, but these are not comprehensive, therefore you must consider **all possible cases** to get a full mark. User inputs are shown in **bold**.

The input files of the visible test cases are also provided in the homework package. In C-Lion, input text files must be placed in the folder that starts with cmake-build-. This is something different than Visual Studio.

We configured CodeRunner to test these sample runs for you (as visible test cases). However, there also are some hidden test cases that would affect your grade. We will not disclose the hidden test cases before the grading has been completed.

We do **not** recommend you to copy and paste the prompts and messages from this document since some hidden control characters and non-standard characters might cause problems in CodeRunner.

Sample Run 1

```
Please enter the file name:
s1.text
Could not open the file. Please enter a valid file name:
s1
Could not open the file. Please enter a valid file name:
s1.txt
Read line number (1): jessie likes james
jessie likes james relation has been added to the likes list.
Likes list: (jessie, james)
Taken list:

Read line number (2): james likes pikachu
james likes pikachu relation has been added to the likes list.
Likes list: (jessie, james) (james, pikachu)
Taken list:

Read line number (3): jessie likes pikachu
jessie likes pikachu relation has been added to the likes list.
Likes list: (jessie, james) (jessie, pikachu) (james, pikachu)
Taken list:

Read line number (4): jessie likes james
jessie likes james relation already exists in the likes list, so it is not added to it.
Likes list: (jessie, james) (jessie, pikachu) (james, pikachu)
Taken list:

Lists are deleted and the program ends successfully.
```

Sample Run 2

```
Please enter the file name:
s22
Could not open the file. Please enter a valid file name:
s2.txt
Read line number (1): donald likes daisy
donald likes daisy relation has been added to the likes list.
Likes list: (donald, daisy)
Taken list:

Read line number (2): minnie likes mickey
minnie likes mickey relation has been added to the likes list.
Likes list: (donald, daisy) (minnie, mickey)
Taken list:

Read line number (3): mickey likes minnie
Match found: mickey likes minnie and minnie likes mickey
Added to taken list: mickey
Added to taken list: minnie
Any node that has mickey or minnie or both in it is removed from the likes list.
Likes list: (donald, daisy)
Taken list: mickey minnie

Read line number (4): daisy likes donald
Match found: daisy likes donald and donald likes daisy.
Added to taken list: daisy
Added to taken list: donald
Any node that has daisy or donald or both in it is removed from the likes list.
Likes list:
Taken list: daisy donald mickey minnie

Lists are deleted and the program ends successfully.
```

Note here the order of the messages. The message for the subject of the new relation is printed before the one for the object

Sample Run 3

```
Please enter the file name:
sample3
Could not open the file. Please enter a valid file name:
s3.txt
Read line number (1): jasmine likes prince
jasmine likes prince relation has been added to the likes list.
Likes list: (jasmine, prince)
Taken list:

Read line number (2): jasmine likes aladdin
jasmine likes aladdin relation has been added to the likes list.
Likes list: (jasmine, prince) (jasmine, aladdin)
Taken list:

Read line number (3): jasmine likes aladdin
jasmine likes aladdin relation already exists in the likes list, so it is not added to it.
Likes list: (jasmine, prince) (jasmine, aladdin)
Taken list:

Read line number (4): homer likes marge
homer likes marge relation has been added to the likes list.
Likes list: (jasmine, prince) (jasmine, aladdin) (homer, marge)
Taken list:
```

```
Read line number (5): prince likes princess
prince likes princess relation has been added to the likes list.
Likes list: (jasmine, prince) (jasmine, aladdin) (homer, marge) (prince, princess)
Taken list:

Read line number (6): marge likes homer
Match found: marge likes homer and homer likes marge.
Added to taken list: marge
Added to taken list: homer
Any node that has marge or homer or both in it is removed from the likes list.
Likes list: (jasmine, prince) (jasmine, aladdin) (prince, princess)
Taken list: homer marge

Read line number (7): aladdin likes princess
aladdin likes princess relation has been added to the likes list.
Likes list: (jasmine, prince) (jasmine, aladdin) (prince, princess) (aladdin, princess)
Taken list: homer marge

Lists are deleted and the program ends successfully.
```

Sample Run 4

```
Please enter the file name:
s4/txt
Could not open the file. Please enter a valid file name:
s4.txt
Read line number (1): aladdin likes jasmine
aladdin likes jasmine relation has been added to the likes list.
Likes list: (aladdin, jasmine)
Taken list:

Read line number (2): kerchak likes kala
kerchak likes kala relation has been added to the likes list.
Likes list: (aladdin, jasmine) (kerchak, kala)
Taken list:

Read line number (3): fiona likes shrek
fiona likes shrek relation has been added to the likes list.
Likes list: (aladdin, jasmine) (kerchak, kala) (fiona, shrek)
Taken list:

Read line number (4): jasmine likes aladdin
Match found: jasmine likes aladdin and aladdin likes jasmine.
Added to taken list: jasmine
Added to taken list: aladdin
Any node that has jasmine or aladdin or both in it is removed from the likes list.
Likes list: (kerchak, kala) (fiona, shrek)
Taken list: aladdin jasmine

Read line number (5): carl likes ellie
carl likes ellie relation has been added to the likes list.
Likes list: (kerchak, kala) (fiona, shrek) (carl, ellie)
Taken list: aladdin jasmine

Read line number (6): ellie likes carl
Match found: ellie likes carl and carl likes ellie.
Added to taken list: ellie
Added to taken list: carl
Any node that has ellie or carl or both in it is removed from the likes list.
Likes list: (kerchak, kala) (fiona, shrek)
Taken list: aladdin carl ellie jasmine
```

```
Read line number (7): ellie likes carl
Either ellie or carl or both is/are already taken, so the like relation is not added.
Likes list: (kerchak, kala) (fiona, shrek)
Taken list: aladdin carl ellie jasmine

Read line number (8): kala likes kerchak
Match found: kala likes kerchak and kerchak likes kala.
Added to taken list: kala
Added to taken list: kerchak
Any node that has kala or kerchak or both in it is removed from the likes list.
Likes list: (fiona, shrek)
Taken list: aladdin carl ellie jasmine kala kerchak

Read line number (9): donald likes donna
donald likes donna relation has been added to the likes list.
Likes list: (fiona, shrek) (donald, donna)
Taken list: aladdin carl ellie jasmine kala kerchak

Read line number (10): donna likes donald
Match found: donna likes donald and donald likes donna.
Added to taken list: donna
Added to taken list: donald
Any node that has donna or donald or both in it is removed from the likes list.
Likes list: (fiona, shrek)
Taken list: aladdin carl donald donna ellie jasmine kala kerchak

Read line number (11): daisy likes donald
Either daisy or donald or both is/are already taken, so the like relation is not added.
Likes list: (fiona, shrek)
Taken list: aladdin carl donald donna ellie jasmine kala kerchak

Lists are deleted and the program ends successfully.
```

Sample Run 5

```
Please enter the file name:
s5.txt
Read line number (1): naveen likes tiana
naveen likes tiana relation has been added to the likes list.
Likes list: (naveen, tiana)
Taken list:

Read line number (2): cleopatra likes naveen
cleopatra likes naveen relation has been added to the likes list.
Likes list: (naveen, tiana) (cleopatra, naveen)
Taken list:

Read line number (3): antony likes cleopatra
antony likes cleopatra relation has been added to the likes list.
Likes list: (naveen, tiana) (cleopatra, naveen) (antony, cleopatra)
Taken list:

Read line number (4): naveen likes princess
naveen likes princess relation has been added to the likes list.
Likes list: (naveen, tiana) (naveen, princess) (cleopatra, naveen) (antony, cleopatra)
Taken list:

Read line number (5): tiana likes naveen
Match found: tiana likes naveen and naveen likes tiana.
Added to taken list: tiana
Added to taken list: naveen
Any node that has tiana or naveen or both in it is removed from the likes list.
Likes list: (antony, cleopatra)
Taken list: naveen tiana

Read line number (6): antony likes cleopatra
antony likes cleopatra relation already exists in the likes list, so it is not added to it.
Likes list: (antony, cleopatra)
Taken list: naveen tiana

Read line number (7): cleopatra likes antony
Match found: cleopatra likes antony and antony likes cleopatra.
Added to taken list: cleopatra
Added to taken list: antony
Any node that has cleopatra or antony or both in it is removed from the likes list.
Likes list:
Taken list: antony cleopatra naveen tiana

Read line number (8): ebla likes antere
ebla likes antere relation has been added to the likes list.
Likes list: (ebla, antere)
Taken list: antony cleopatra naveen tiana

Lists are deleted and the program ends successfully.
```

Sample Run 6

```
Please enter the file name:
imp6.txt
Read line number (1): aladdin likes jasmine
aladdin likes jasmine relation has been added to the likes list.
Likes list: (aladdin, jasmine)
Taken list:

Read line number (2): cleopatra likes aladdin
cleopatra likes aladdin relation has been added to the likes list.
Likes list: (aladdin, jasmine) (cleopatra, aladdin)
Taken list:

Read line number (3): antony likes cleopatra
antony likes cleopatra relation has been added to the likes list.
Likes list: (aladdin, jasmine) (cleopatra, aladdin) (antony, cleopatra)
Taken list:

Read line number (4): aladdin likes tiana
aladdin likes tiana relation has been added to the likes list.
Likes list: (aladdin, jasmine) (aladdin, tiana) (cleopatra, aladdin) (antony, cleopatra)
Taken list:

Read line number (5): jasmine likes aladdin
Match found: jasmine likes aladdin and aladdin likes jasmine.
Added to taken list: jasmine
Added to taken list: aladdin
Any node that has jasmine or aladdin or both in it is removed from the likes list.
Likes list: (antony, cleopatra)
Taken list: aladdin jasmine

Read line number (6): antony likes cleopatra
antony likes cleopatra relation already exists in the likes list, so it is not added to it.
Likes list: (antony, cleopatra)
Taken list: aladdin jasmine
```

```
Read line number (7): cleopatra likes antony
Match found: cleopatra likes antony and antony likes cleopatra.
Added to taken list: cleopatra
Added to taken list: antony
Any node that has cleopatra or antony or both in it is removed from the likes list.
Likes list:
Taken list: aladdin antony cleopatra jasmine

Read line number (8): jasmine likes antony
Either jasmine or antony or both is/are already taken, so the like relation is not added.
Likes list:
Taken list: aladdin antony cleopatra jasmine

Read line number (9): antony likes jasmine
Either antony or jasmine or both is/are already taken, so the like relation is not added.
Likes list:
Taken list: aladdin antony cleopatra jasmine

Lists are deleted and the program ends successfully.
```

Sample Run 7

```
Please enter the file name:
s7txt
Could not open the file. Please enter a valid file name:
file7.txt
Read line number (1): polat likes elif
polat likes elif relation has been added to the likes list.
Likes list: (polat, elif)
Taken list:

Read line number (2): behlul likes bihter
behlul likes bihter relation has been added to the likes list.
Likes list: (polat, elif) (behlul, bihter)
Taken list:

Read line number (3): behlul likes eysan
behlul likes eysan relation has been added to the likes list.
Likes list: (polat, elif) (behlul, bihter) (behlul, eysan)
Taken list:

Read line number (4): mecnun likes leyla
mecnun likes leyla relation has been added to the likes list.
Likes list: (polat, elif) (behlul, bihter) (behlul, eysan) (mecnun, leyla)
Taken list:

Read line number (5): mecnun likes nur
mecnun likes nur relation has been added to the likes list.
Likes list: (polat, elif) (behlul, bihter) (behlul, eysan) (mecnun, leyla) (mecnun, nur)
Taken list:

Read line number (6): behlul likes menekse
behlul likes menekse relation has been added to the likes list.
Likes list: (polat, elif) (behlul, bihter) (behlul, eysan) (behlul, menekse) (mecnun, leyla) (mecnun, nur)
Taken list:

Read line number (7): behlul likes eysan
behlul likes eysan relation already exists in the likes list, so it is not added to it.
Likes list: (polat, elif) (behlul, bihter) (behlul, eysan) (behlul, menekse) (mecnun, leyla) (mecnun, nur)
Taken list:

Read line number (8): leyla likes mecnun
Match found: leyla likes mecnun and mecnun likes leyla.
Added to taken list: leyla
Added to taken list: mecnun
Any node that has leyla or mecnun or both in it is removed from the likes list.
```

```
Likes list: (polat, elif) (behlul, bihter) (behlul, eysan) (behlul, menekse)
Taken list: leyla mecnun

Read line number (9): behlul likes bihter
behlul likes bihter relation already exists in the likes list, so it is not added to it.
Likes list: (polat, elif) (behlul, bihter) (behlul, eysan) (behlul, menekse)
Taken list: leyla mecnun

Read line number (10): polat likes elif
polat likes elif relation already exists in the likes list, so it is not added to it.
Likes list: (polat, elif) (behlul, bihter) (behlul, eysan) (behlul, menekse)
Taken list: leyla mecnun

Read line number (11): bihter likes behlul
Match found: bihter likes behlul and behlul likes bihter.
Added to taken list: bihter
Added to taken list: behlul
Any node that has bihter or behlul or both in it is removed from the likes list.
Likes list: (polat, elif)
Taken list: behlul bihter leyla mecnun

Lists are deleted and the program ends successfully.
```