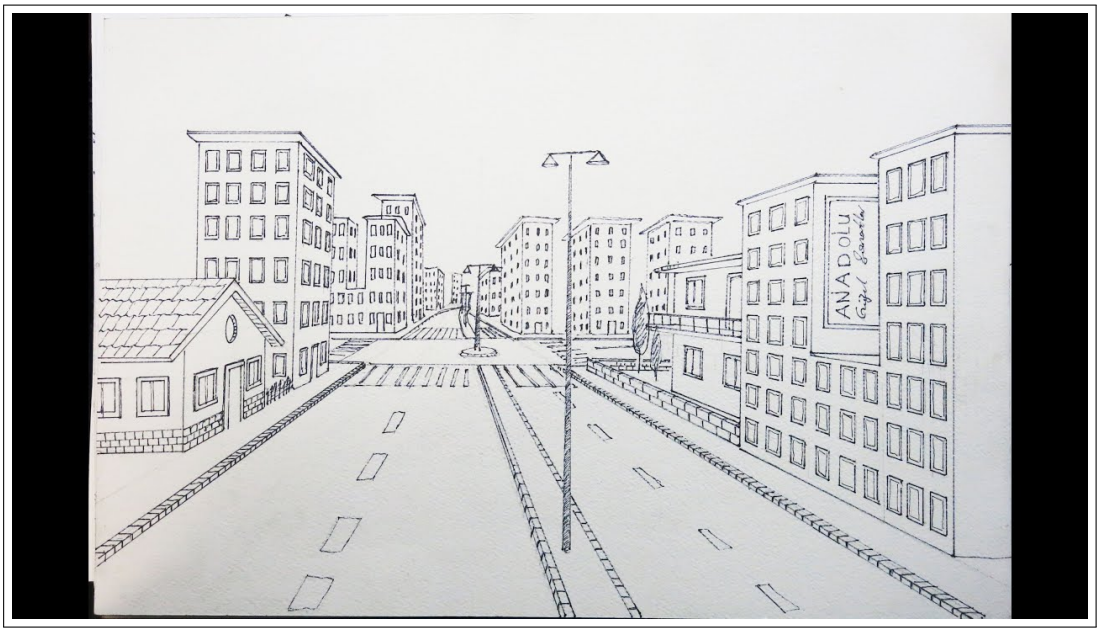
ASSIGNMENT 2

Subject : Linked List
TAs: Merve Ozdes, Bahar Gezici
Programming Lanuage: C++
Due Date: 25.11.2022

1 Introduction

In this assignment, you are expected to create a basic network mapping for a street by using linked list data structure. The Internet comes with cables until our wireless modem in our home, and there is a bandwidth value which is related to how much money you pay for your Internet connection for each month. You are going to create a basic Internet connection mapping in a street.



2 Background

A linked list is a sequence of data structures, which are connected together via links.

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

- **Link:** Each link of a linked list can store a data called an element.
- **Next:** Each link of a linked list contains a link to the next link called Next.
- **LinkedList:** A Linked List contains the connection link to the first link called First.

Linked list can be visualized as a chain of nodes, where every node points to the next node.

Following are the various types of linked list.

- Simple Linked List: Item navigation is forward only.
- Doubly Linked List: Items can be navigated forward and backward. Doubly linked list contains an extra pointer, typically called the previous pointer, together with the next pointer and data which are there in the singly linked list.
- Circular Linked List: In a circular linked list, the last node of the list contains a pointer to the first node of the list.

Following are the basic operations supported by a list.

- Insertion: Adds an element at the beginning of the list.
- Deletion: Deletes an element at the beginning of the list.
- Display: Displays the complete list.
- Search: Searches an element using the given key.
- Delete: Deletes an element using the given key.

Examples for insertion and deletion operations are given below.

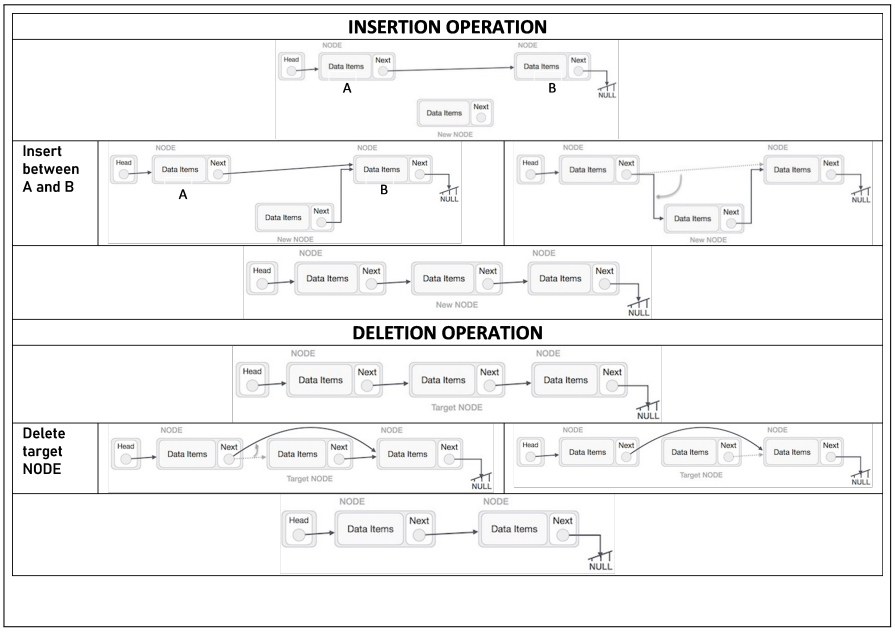


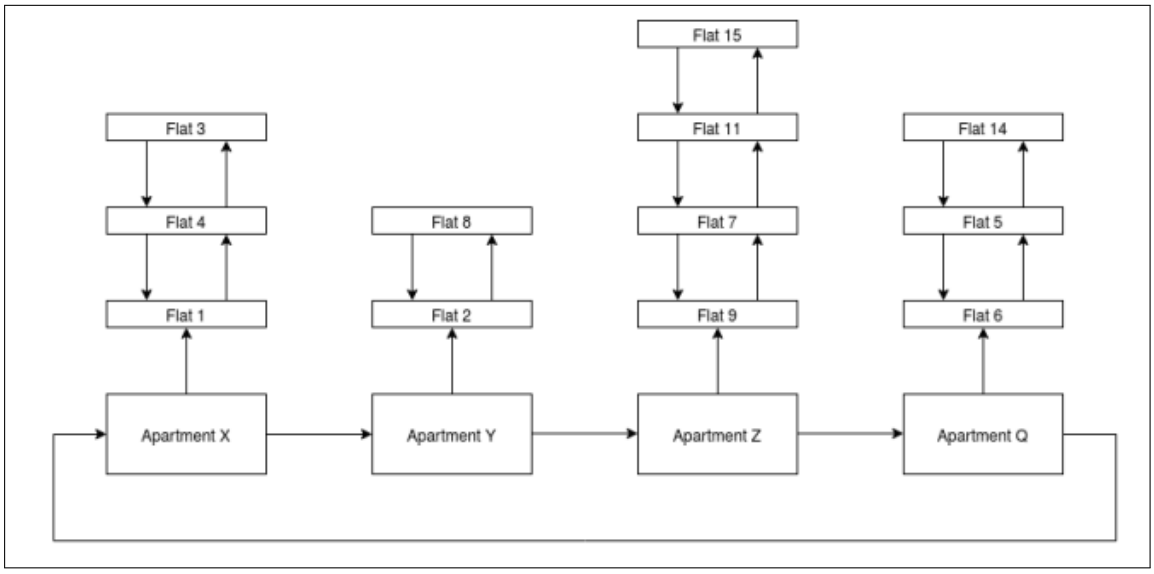
Figure 1: Examples of insertion and deletion operations

3 Experiment

In this experiment, you are expected to write an application that design Internet connection mapping in a street by using linked list data structure. The application will take the input.txt file from command line and read its contents.

There are some specifications for this experiment as given below. In every street, there are many apartments next to each other and you will design an internal Internet connection of one of these streets.

- There is a circular linked list for the street which is called as apartment linked list.
- Every apartment in a street should be considered as a node in the apartment linked list.
- The time complexity must be O(N) when inserting an apartment to apartment linked-list and deleting an apartment from apartment linked-list.
- Every apartment consist of flats, so there must be a doubly linked list for each apartment in the street. Every apartment has own doubly linked list which is called flat linked list for the flats in that apartment.
- Every flat in an apartment should be considered as a node in the apartment's flat linked list.
- The time complexity must be O(1) when inserting an flat to flat linked-list and deleting an flat from flat linked-list.
- Total bandwidth of the apartment must be shared between the flats in the apartment and the sum of the initial_bandwidth values of the flats cannot be more than max_bandwidth value of the apartment.



In the street,

- Every apartment has
  - own unique name, which is used to distinguish an apartment from other apartments,
  - max_bandwidth value (int max_bandwidth), which shows the maximum_bandwidth value of the sum of flats' initial_bandwidth values.
- Every house (flat) has

- an unique ID in the entire street (int id), which is used to distinguish a flat from other flats (ID of a flat is NOT related to the index in the flat list.),
  - **Important Note 1:** IDs of the flats will be unique for the whole street not only for the apartment.
  - **Important Note 2:** ID of a flat will always be a positive integer (1, 2, 3, ...).
- initial_bandwidth value (int initial_bandwidth), which shows the initial bandwidth of the flat,
- is_empty flag, which shows whether there are residents or the flat is empty.

There are several commands will be given in input file for this experiment:

- **add_apartment**
- **add_flat**
- **remove_apartment**
- **make_flat_empty**
- **find_sum_of_max_bandwidths**
- **merge_two_apartments**
- **relocate_flats_to_same_apartment**
- **list_apartments**

3.1 add_apartment

- This function adds a new apartment at required position in the apartment linked list.
- The newly created apartment's name and max_bandwidth values must be apartment_name and max_bandwidth as given in the arguments respectively.
- Given position will be "head" or it will has identifier such as before and after an apartment.
- Given apartment_name will be unique in the apartment linked list.
- flat_list of newly added apartment must be NULL.
- If initially there is no apartment in the apartment linked list, head and last (tail) node will be NULL.
- If insertion is given after the tail, you must store the address of the head node to next of newNode (making newNode the last node), point the current last node to newNode, and make newNode as the last node. As shown in Figure 3, head is Apartment X and tail is Apartment Z. After insertion, new tail will be Apartment Y.
- If insertion is given before the first node (head), you must store the address of the current first node in the newNode (i.e. pointing the newNode to the current first node) and point the last node to newNode (i.e making newNode as head). As shown in Figure 4, head is Apartment X and tail is Apartment Z. After insertion, new head will be Apartment Y.

The command structure is as follows:

```
add_apartment<TAB><apartment_name><TAB><position><TAB><max_bandwith>
OR
add_apartment<TAB><apartment_name><TAB>head<TAB><max_bandwith>
```

Example 1: add_apartment Y before_Z 100

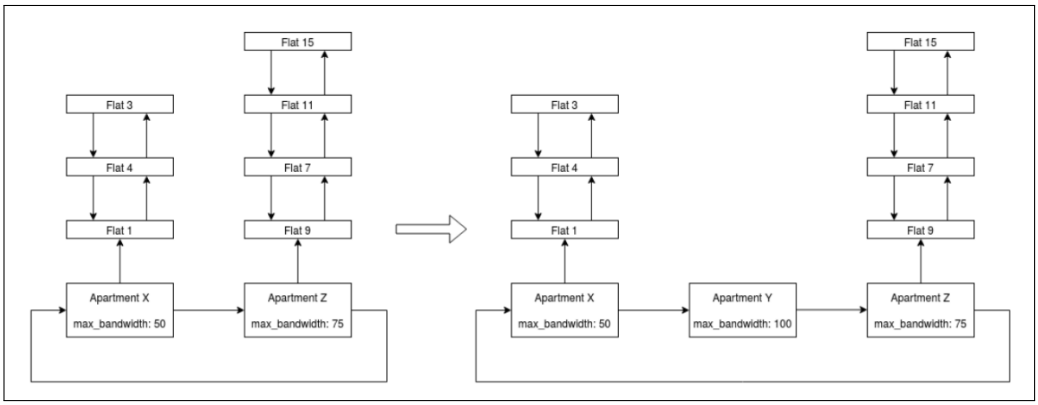


Figure 2: Insertion in between two apartments

Example 2: add_apartment Y after_Z 100

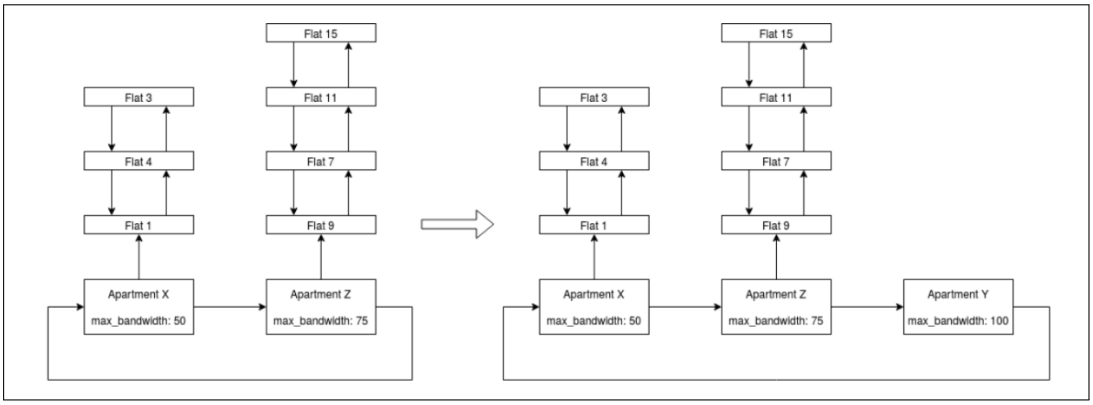


Figure 3: Insertion at the end

Example 3: add_apartment Y before_X 100

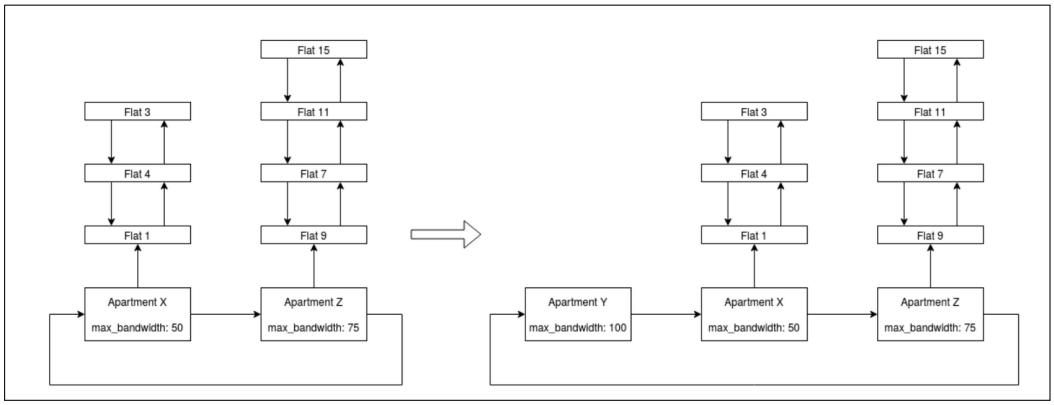


Figure 4: Insertion at the beginning

3.2 add_flat

- This function adds a new flat at required index in the flat linked list of the apartment whose name is given apartment_name. If there is a flat at the given index, add new flat in the correct position.
- ID of the newly created flat must be flat_id as given in the arguments.
- initial_bandwidth value of the flat must be initial_bandwidth as given in the arguments. However, as you can understand, sum of the flat's bandwidth values for an apartment cannot be more than max_bandwidth value of that apartment. Therefore, before assigning the initial_bandwidth value to new flat, you should calculate the maximum bandwidth of the newly added flat can have. If it is less than the given initial_bandwidth value, you should assign the calculated maximum bandwidth value to the new flat instead of the given initial_bandwidth.
- If the maximum bandwidth of the newly added flat can have is zero, then you should assign the 0 to the initial_bandwidth of new flat and 1 to the is_empty flag of the new flat.
- Initially, is_empty flag of the new flat must be 0.
- Given index will always be

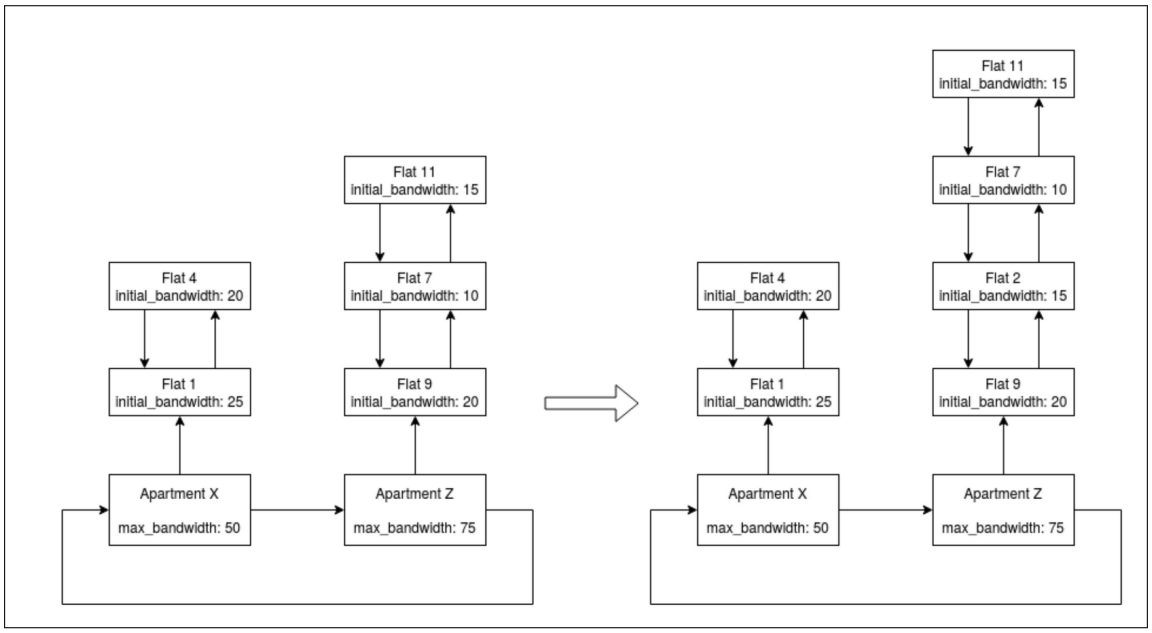
```
0 <= index <= initial_flat_count of given apartment
```

- Given flat_id will be unique in the entire street.
- You should make your operations on the apartment linked list.

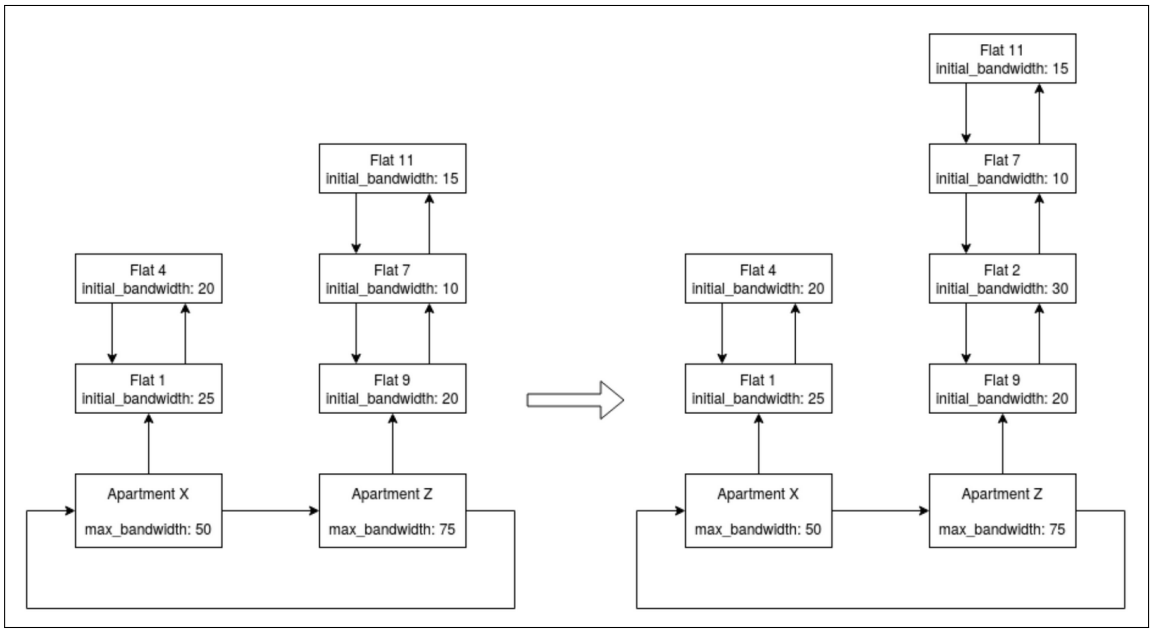
The command structure is as follows:

`add_flat<TAB><apartment_name><TAB><index><TAB><initial_bandwith><TAB><flat_id>`

Example 1: add_flat Z 1 15 2



Example 2: add_flat Z 1 45 2



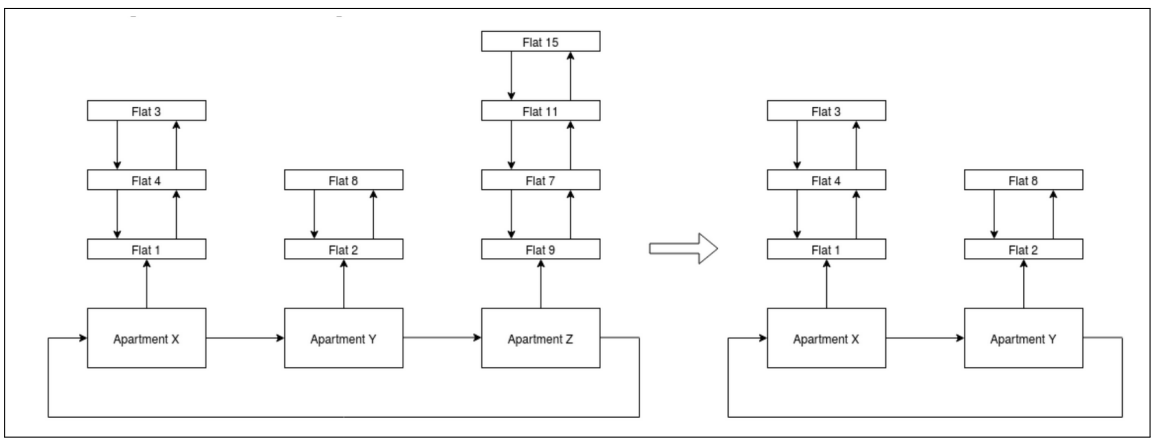
3.3 remove_apartment

- This function removes the apartment whose name is equal to given apartment name from the apartment linked list.
- As you know, when you remove the apartment from the list, you actually did not free the apartment. You should also free the given apartment. Moreover, freeing only the apartment is not also enough, you should also free the flat linked list of removed apartment. -
- After the freeing operations, it should return the changed apartment linked list.
- After remove operation, if there is not any apartment in the apartment linked list, this function must return NULL.

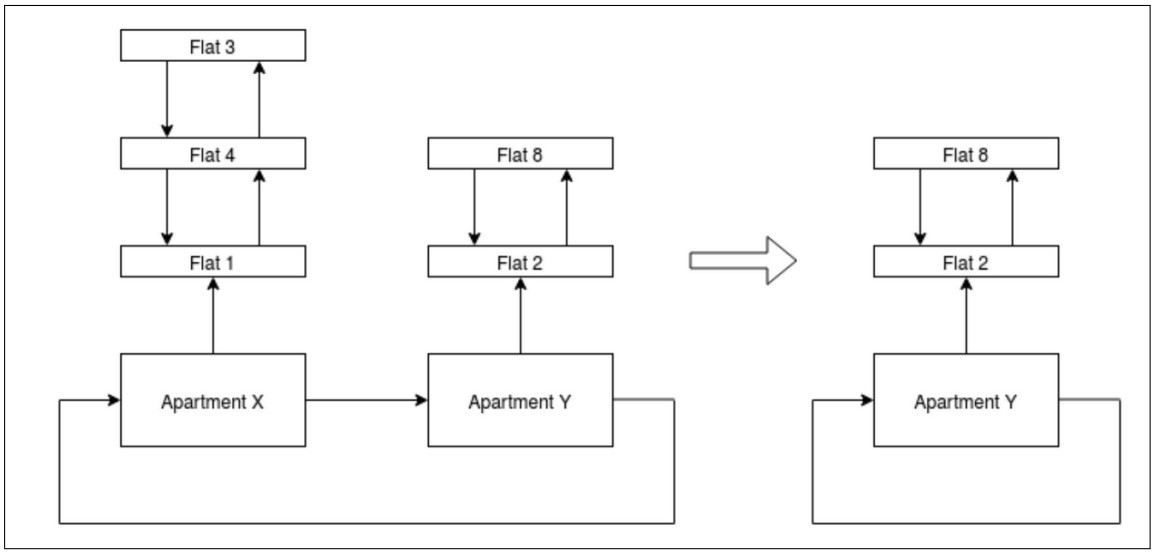
The command structure is as follows:

`remove_apartment<TAB><apartment_name>`

Example 1: remove_apartment Z



Example 2: remove_apartment X



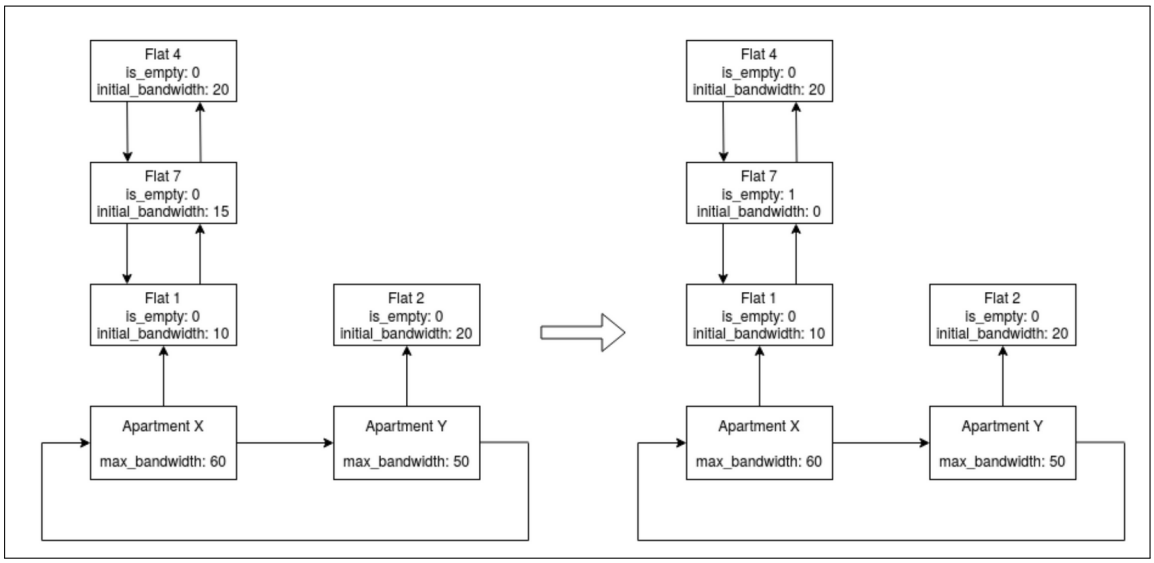
3.4 make_flat_empty

- This function should find the flat whose ID is equal to given flat_id of the apartment whose name is equal to given apartment name. However, it does not remove the flat from the flat linked list, it only changes its is empty flag to 1 and initial bandwidth to 0.
- You should make your operations on the given apartment list (Apartment* head), in the evaluation steps, this list will be used and compared with the expected list.

The command structure is as follows:

`make_flat_empty<TAB><apartment_name><TAB><flat_id>`

Example 2: make_flat_empty X 7



3.5 find_sum_of_max_bandwidth

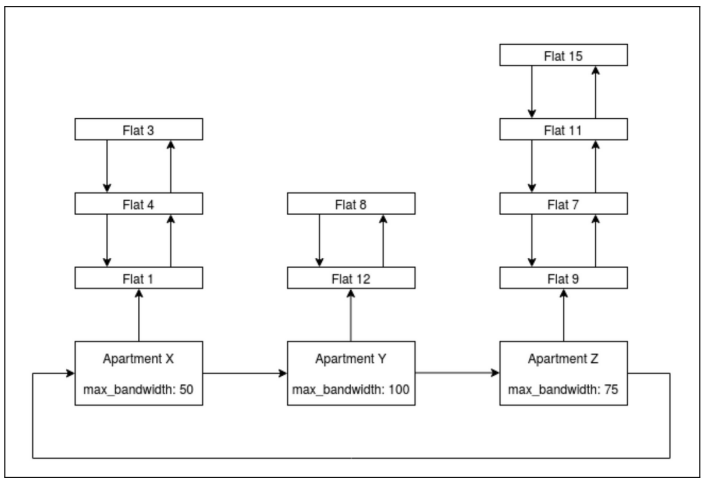
- This function sums the max bandwidth values of the apartments in the given apartment linked list, then returns the sum.
- If there is not any apartment in the given apartment linked list, it must return 0.

The command structure is as follows:

`find_sum_of_max_bandwidth`

Example 1: find_sum_of_max_bandwidth

For the following example sum_of_max_bandwidth is **225**.



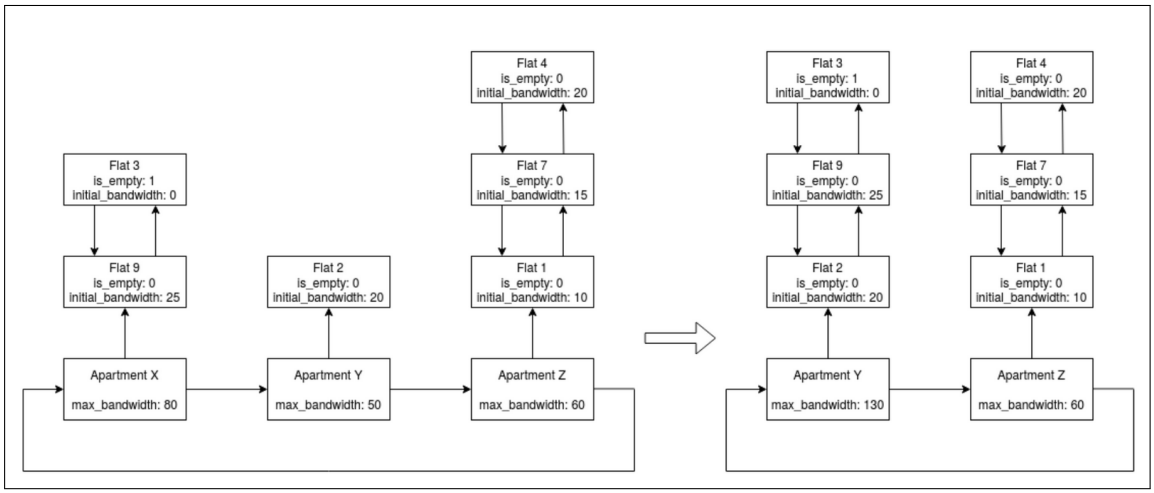
3.6 merge_two_apartments

- This function appends the flats of the second apartment whose name is apartment name 2 to the end of the first apartment whose name is apartment name 1.
- If you firstly delete the given nodes and create again them in the required places, it will NOT be accepted as a solution. You must MOVE the given flats. By changing prev and next pointers of some flats, you must locate the given flats in different places.
- It should add the second apartment's max bandwidth value to the first apartment's max bandwidth value.
- Finally, it removes the second apartment from the apartment linked list, then it returns the changed apartment linked list.
- You should consider the cases where the flat list of the first apartment or second apartment or both of the apartments is NULL.

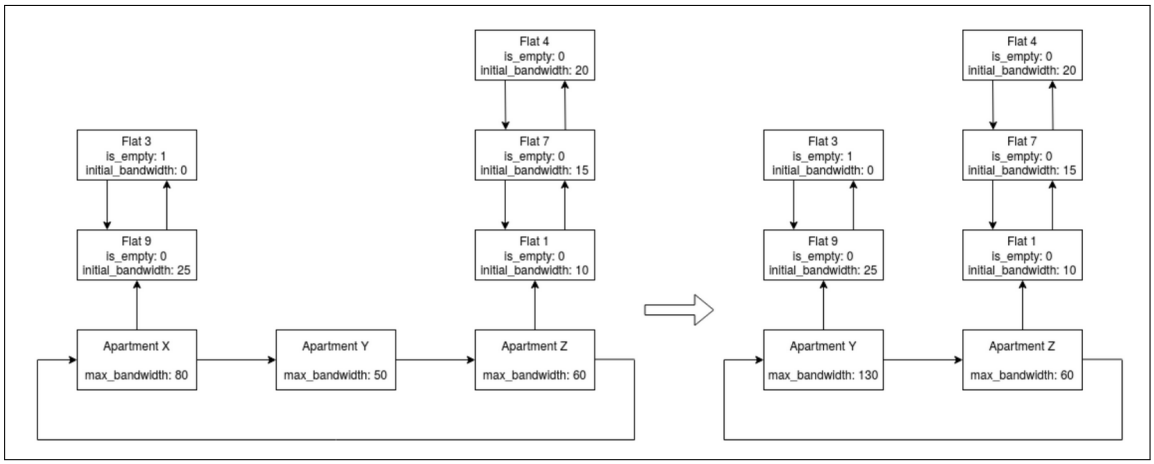
The command structure is as follows:

`merge_two_apartments<TAB><apartment_name_1><TAB><apartment_name_2>`

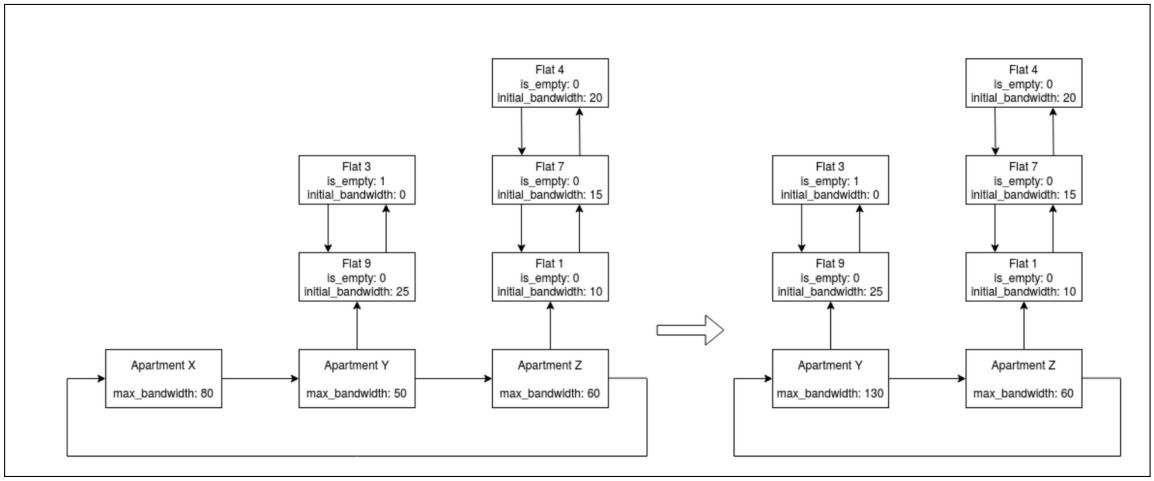
Example 1: merge_two_apartments Y X



Example 2: merge_two_apartments Y X



Example 3: merge_two_apartments Y X



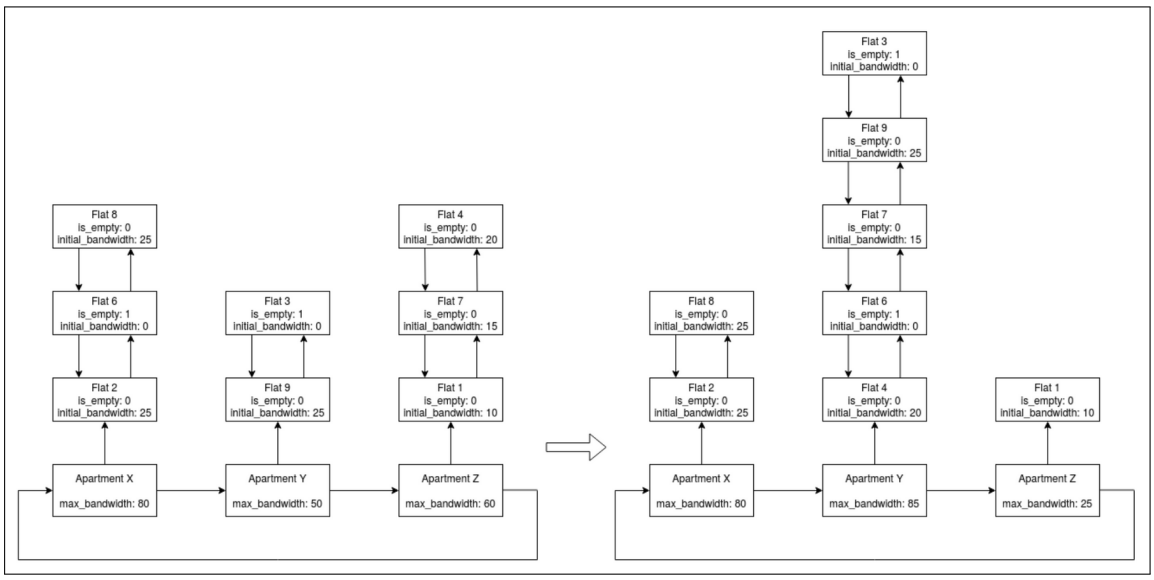
3.7 relocate_flats_to_same_apartments

- This function relocates the different flats in different apartments to a specific place at the same apartment consecutively. new_ apartment_ name is the name of the apartment which different flats in different apartments should be moved in. You must also locate them at the place of the flat whose id is flat_id to shift by shifting it to forward. IDs of the flats that you need to change their apartments are given with flat id list. Flats' apartment name information is not given and they will be in different apartments. Therefore, you should traverse the entire street to find flats' locations. After you find the locations, while relocating the flats, you should preserve their order in the flat id list. In other words, you should place them to flat linked list of the new apartment one by one in the same order at the flat id list.
- If you firstly delete the given nodes and create again them in the required places, it will NOT be accepted as a solution. You must **MOVE** the given flats. By changing prev and next pointers of some flats, you must locate the given flats in different places.
- As you can understand, max_bandwidth value of the new apartment and the old apartment of each flat must be updated. For each relocated flat, you should subtract the flat's initial_bandwidth value from the old apartment and add it to the new apartment.
- There will always be at least one flat to shift in the flat list of the given apartment whose name is new_apartment_name.
- It is guarantied that given flats in the flat id list will not be in the apartment whose name is new apartment name.
- You should make your operations on the given apartment list, in the evaluation steps, this list will be used and compared with the expected list.

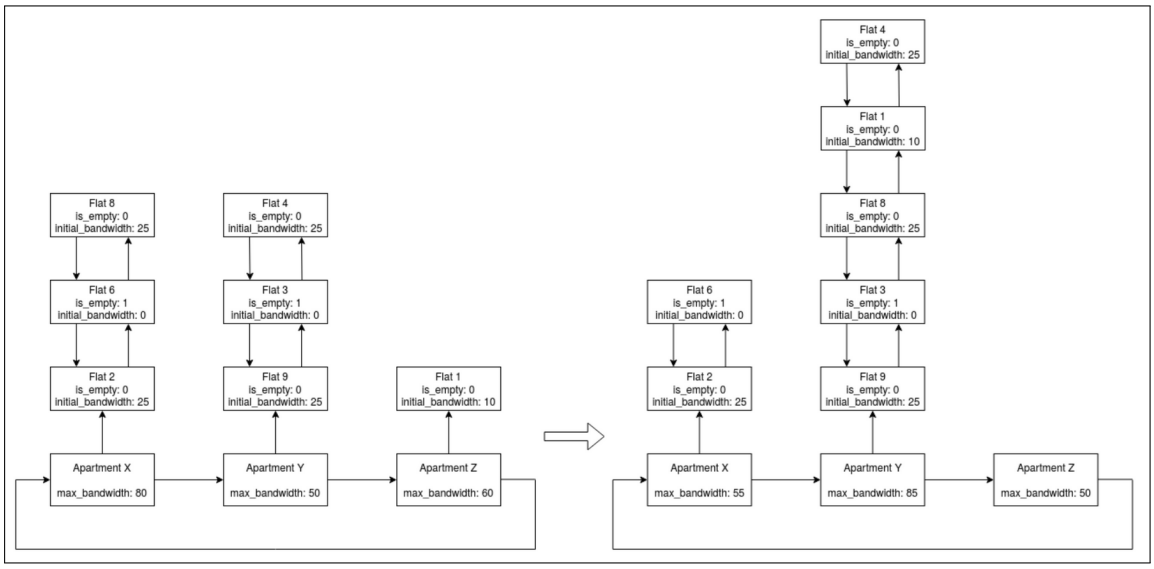
The command structure is as follows:

`relocate_flats_to_same_apartments<TAB><new_apartment_name><TAB><flat_id_to_shift><TAB><flat_id_list>`

Example 1: relocate_flats_to_same_apartments Y 9 [4,6,7]



Example 2: relocate_flats_to_same_apartments Y 4 [8,1]



3.8 list_apartments

- This function lists apartments and their flats with max_bandwidth value and initial_bandwidth values, respectively.

  The command structure is as follows:

  `list_apartments`

4 Inputs and Outputs

For this assignment you have one input file that contains commands. You are expected to produce output.txt file according to input.txt file. The format of input and output files are as shown in Figure 5 and Figure 6. Please create your output file according to the format given to you (There should be a blank line between each command output).

5 Execution

The name of the compiled executable program should be "networkmap". Your program should read input/output file names from the command line, so it will be executed as follows:

```
networkmap [input file name] [output file name]
```

```
./networkmap input.txt output.txt
```

You can see sample input and output in piazza page. The program must run on DEV (dev.cs.hacettepe.edu.tr) UNIX machines. So make sure that it compiles and runs on dev server. If we are unable to compile or run, the project risks getting zero point. It is recom- mended that you test the program using the same mechanism on the sample files (provided) and your own inputs. You must compare your own output and sample output. If your output is different from the sample, the project risks getting zero point, too.

```
add_apartment X head 50
add_apartment Z after_X 75
add_apartment Y before_Z 100
add_apartment Q after_Z 150
add_apartment P after_Q 75
add_apartment R before_Y 125
add_apartment S before_P 100
add_apartment T before_Z 50
add_flat X 0 25 1
add_flat X 1 20 4
add_flat Z 0 20 9
add_flat Z 1 10 7
add_flat Z 2 15 11
add_flat Z 1 45 2
add_flat Q 0 45 10
add_flat Q 1 30 12
add_flat Q 1 25 13
remove_apartment Y
add_flat R 0 25 14
add_flat R 0 30 15
add_flat T 0 55 16
merge_two_apartments T R
merge_two_apartments X P
add_flat S 0 35 17
add_flat S 1 25 18
find_sum_of_max_bandwidths
list_apartments
make_flat_empty Z 2
make_flat_empty Q 10
relocate_flats_to_same_apartment S 17 [4,2,13]
list_apartments
```

Figure 5: input.txt

```
sum of bandwidth: 625

X(125) -> Flat1(25) -> Flat4(20)
T(175) -> Flat16(50) -> Flat15(30) -> Flat14(25)
Z(75) -> Flat9(20) -> Flat2(30) -> Flat7(10) -> Flat11(15)
Q(150) -> Flat10(45) -> Flat13(25) -> Flat12(30)
S(100) -> Flat17(35) -> Flat18(25)

X(105) -> Flat1(25)
T(175) -> Flat16(50) -> Flat15(30) -> Flat14(25)
Z(75) -> Flat9(20) -> Flat7(10) -> Flat11(15)
Q(125) -> Flat10(0) -> Flat12(30)
S(145) -> Flat4(20) -> Flat2(0) -> Flat13(25) -> Flat17(35) -> Flat18(25)
```

Figure 6: output.txt

6 Grading and Evaluation

- Your work will be graded over a maximum of 100 points.
- Your total score will be partial according to the grading policy stated below.

Compiled	10p
Each command 10p*8	80p
Code design, clean and readable code, algorithmic perspective, comments	10p

- **Your code will be tested on dev platform with different inputs. And one output file will be expected as output file. If your program cannot compile on dev server, you cannot get any points. Please be sure that your program can be compile on dev server.**

- You should create and submit a ZIP archive in the following structure for evaluation. An invalid structured archive will cause you partial or full score loss. You are required to submit a Makefile, which will be used to compile your program.

Notes

- Do not miss the deadline.
- Save all your work until the assignment is graded.
- The assignment must be original, individual work. Duplicate or very similar assignments are both going to be considered as cheating.
- Write READABLE SOURCE CODE block
- You can ask your questions via Piazza (https://piazza.com/hacettepe.edu.tr/fall/bbm203) and you are supposed to be aware of everything discussed in Piazza.
- You will use online submission system to submit your experiments. https://submit.cs.hacettepe.edu.tr/ No other submission method (email or etc.) will be accepted. Do not submit any file via e-mail related with this assignment.
- File hierarchy must be zipped before submitted (Not .rar, only .zip files are supported by the system). You must submit your work with the file hierarchy stated below:

<student_id.zip>/(Required)

→src/(Required)

→*.cpp

→*.h

→Makefile (Required)

Policy

All work on assignments must be done **individually** unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an **abstract** way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) **will not be tolerated**. In short, turning in someone else's work (from internet), in whole or in part, as your own will be considered **as a violation of academic integrity**. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.

References

www.tutorialspoint.com/