Middle East Technical University

Department of Computer Engineering

CENG 140

C Programming

Spring 2020–2021

Take Home Exam 3

Due date: 11 July 2021 - Sunday / 23:59

1 Introduction

In this homework, you are going to be familiar with "struct" and "linked list" concepts by creating a basic network mapping for a street.

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 sample street like in the image below.

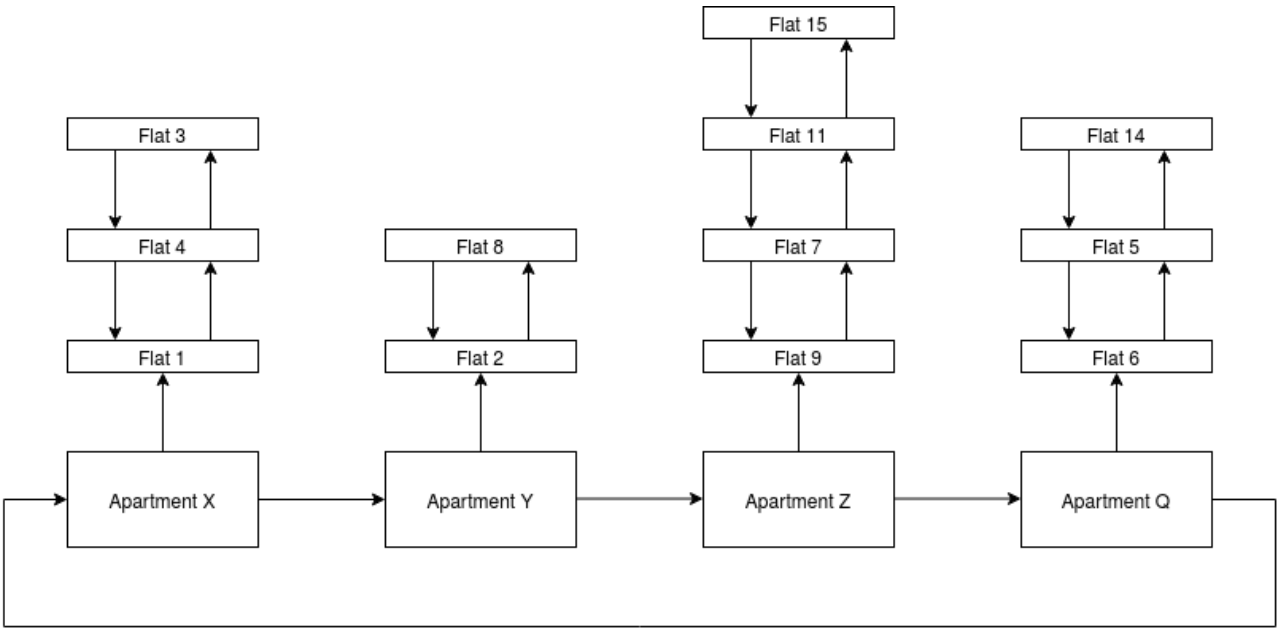
2 Specifications

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.

- 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.
- 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.

To explain visually:



In the street,

- Every apartment has
  - own unique name (char* 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 must 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 (int is_empty), which shows whether there are residents or the flat is empty.

3 Implementation

You can see the structs you will use in this homework below or in the the3.h file. You cannot edit them or you cannot use another structs. You must use these structs in this homework. While evaluating your assignments, your codes will be compiled with the original the3.h file.

```
struct flat
{
    int id;
    int initial_bandwidth;
    int is_empty;
    struct flat* next;
    struct flat* prev;
};
typedef struct flat Flat;

struct apartment
{
    char* name;
    int max_bandwidth;
    struct apartment* next;
    struct flat* flat_list;
};
typedef struct apartment Apartment;
```

3.1 Functions

Important Note: In every function, you must do create/update/delete/move operations on the required nodes and you must not touch the other nodes than you need to change. Moreover, you must update the existing list instead of creating a new list. While evaluating your codes, we will check whether you are changing more than required nodes.

For addition, update or remove operations, you should create and insert the node at the given position or update the information of the given node or remove the node at the given position. However, rest of the given list must be the same.

Especially for the move operation, you must not delete and create again the given nodes. You must MOVE the given nodes.

3.1.1 add_apartment (15 Points)

Function Declaration:

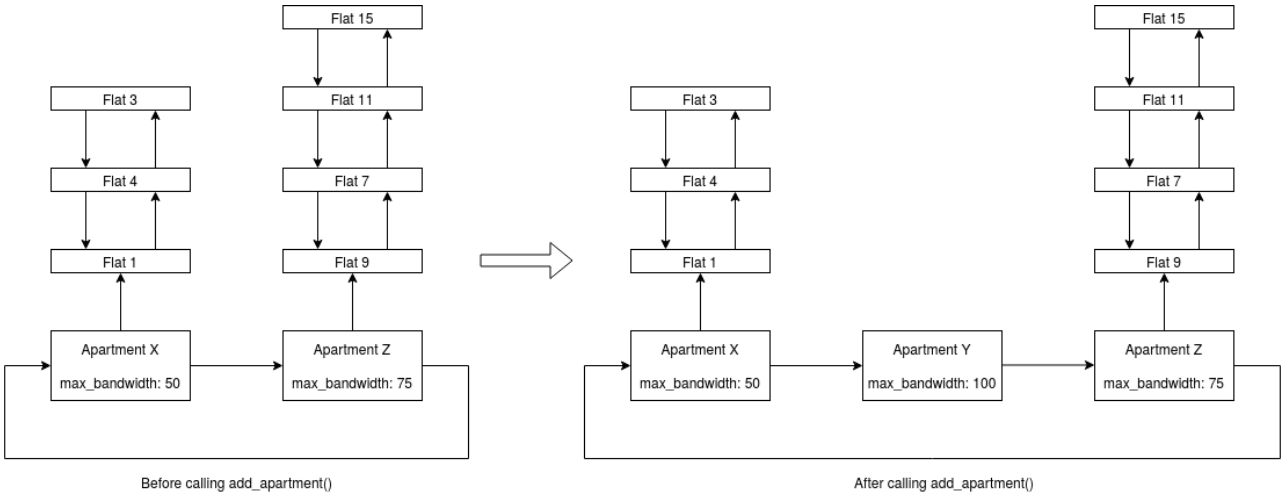
```
Apartment* add_apartment(Apartment* head, int index, char* apartment_name,
                         int max_bandwidth);
```

Explanation:

- This function adds a new apartment at required index in the apartment linked list. If there is an apartment at the given index, you should shift it forward.
- The newly created apartment's name and max_bandwidth values must be apartment_name and max_bandwidth as given in the arguments respectively.
- Given index will always be $0 \leq$ index $\leq$ initial_apartment_count.
- 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, given head argument will be NULL.
- It returns the head of changed apartment linked list.

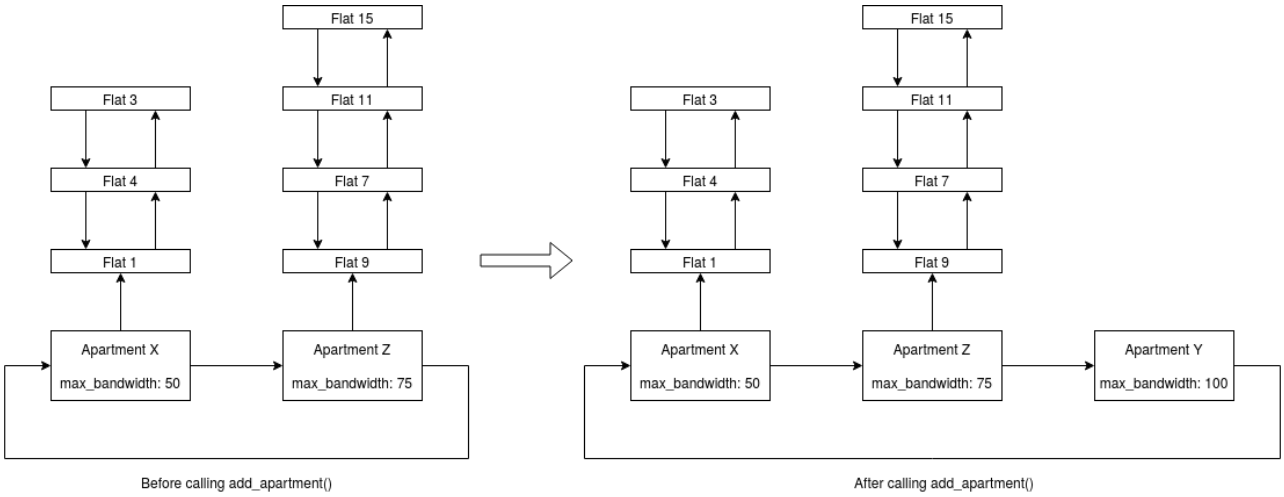
Example - 1:

```
apartment_name: "Y"
index: 1
max_bandwidth: 100
add_apartment(head, index, apartment_name, max_bandwidth);
```



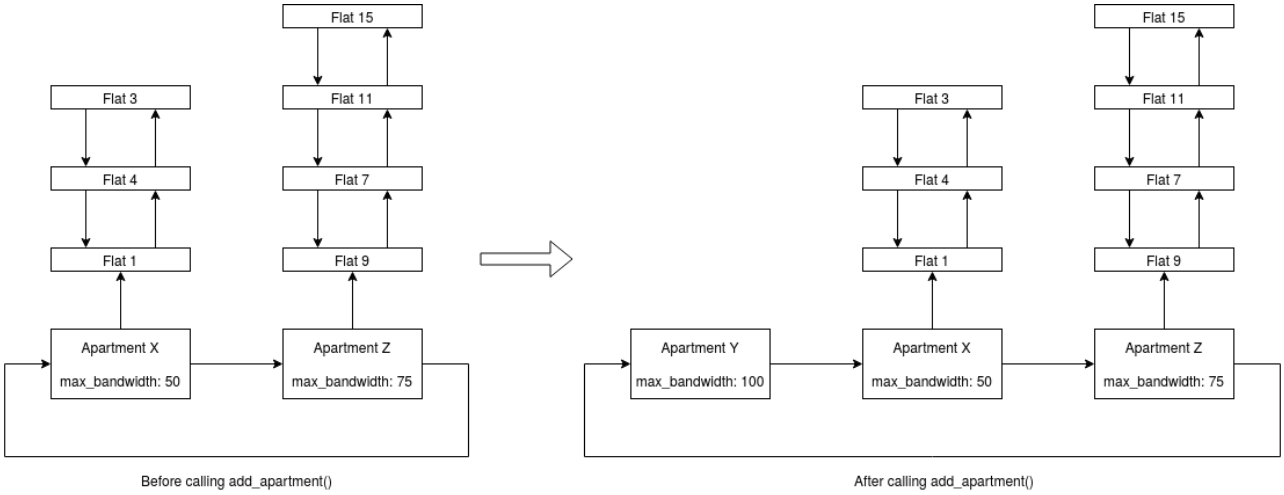
Example - 2:

```
apartment_name: "Y"
index: 2
max_bandwidth: 100
add_apartment(head, index, apartment_name, max_bandwidth);
```



Example - 3:

```
apartment_name: "Y"
index: 0
max_bandwidth: 100
add_apartment(head, index, apartment_name, max_bandwidth);
```



3.1.2 add_flat (20 Points)

Function Declaration:

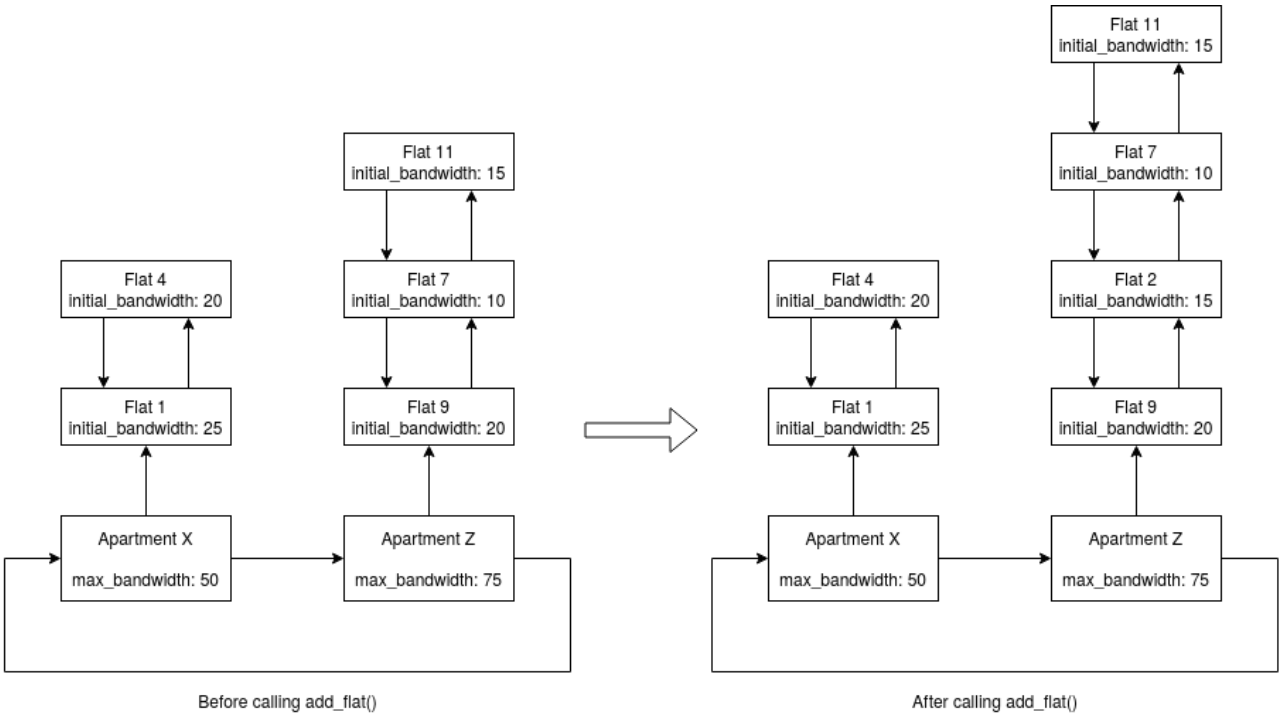
```
void add_flat(Apartment* head, char* apartment_name, int index, int flat_id,
              int initial_bandwidth);
```

Explanation:

- 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, you should shift it forward.
- 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.
- Initially, is_empty flag of the new flat must be 0.
- Given index will always be $0 \leq$ index $\leq$ initial_flat_count of given apartment.
- Given flat_id will be unique in the entire street.
- 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.

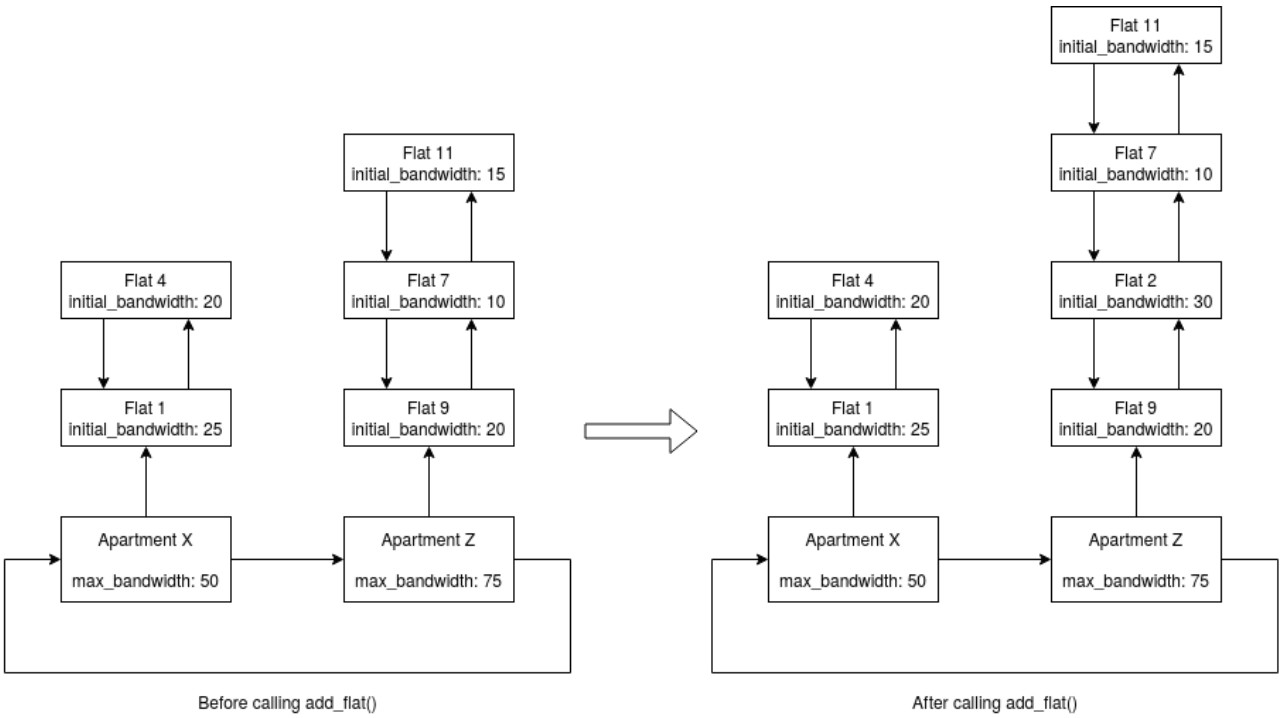
Example - 1:

```
apartment_name: "Z"
index: 1
initial_bandwidth: 15
flat_id: 2
add_flat(head, apartment_name, index, flat_id initial_bandwidth);
```



Example - 2:

```
apartment_name: "Z"
index: 1
initial_bandwidth: 45
flat_id: 2
add_flat(head, apartment_name, index, flat_id initial_bandwidth);
```



3.1.3 remove_apartment (10 Points)

Function Declaration:

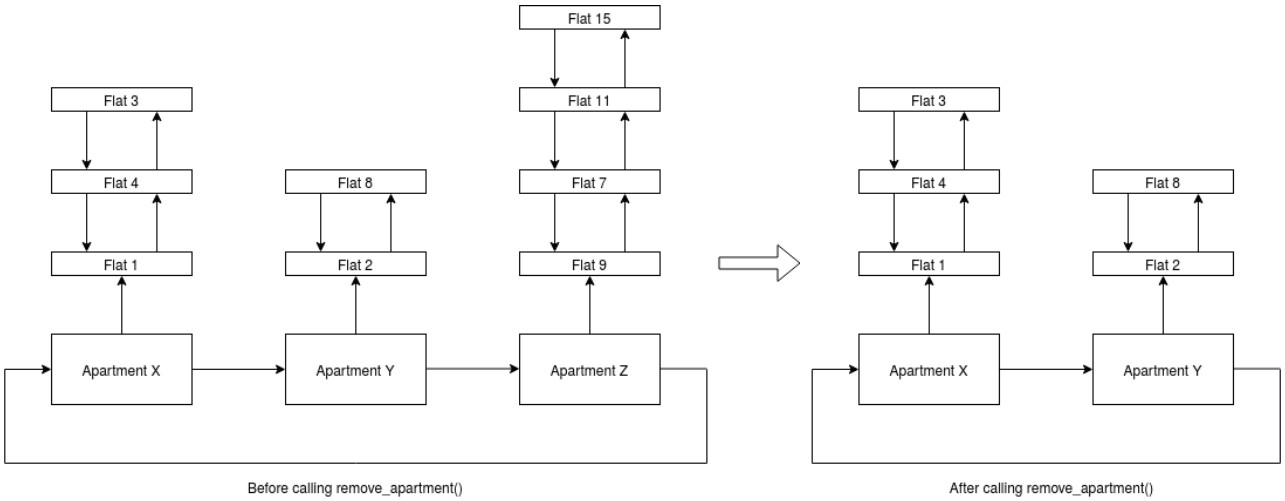
```
Apartment* remove_apartment(Apartment* head, char* apartment_name);
```

Explanation:

- 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.

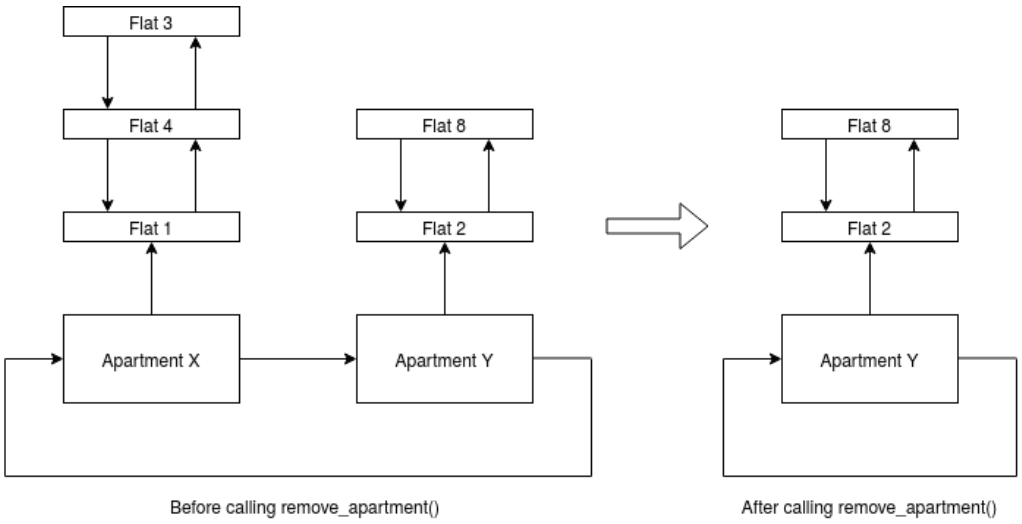
Example - 1:

```
apartment_name: "Z"
remove_apartment(head, apartment_name);
```



Example - 2:

```
apartment_name: "X"
remove_apartment(head, apartment_name);
```



3.1.4 make_flat_empty (10 Points)

Function Declaration:

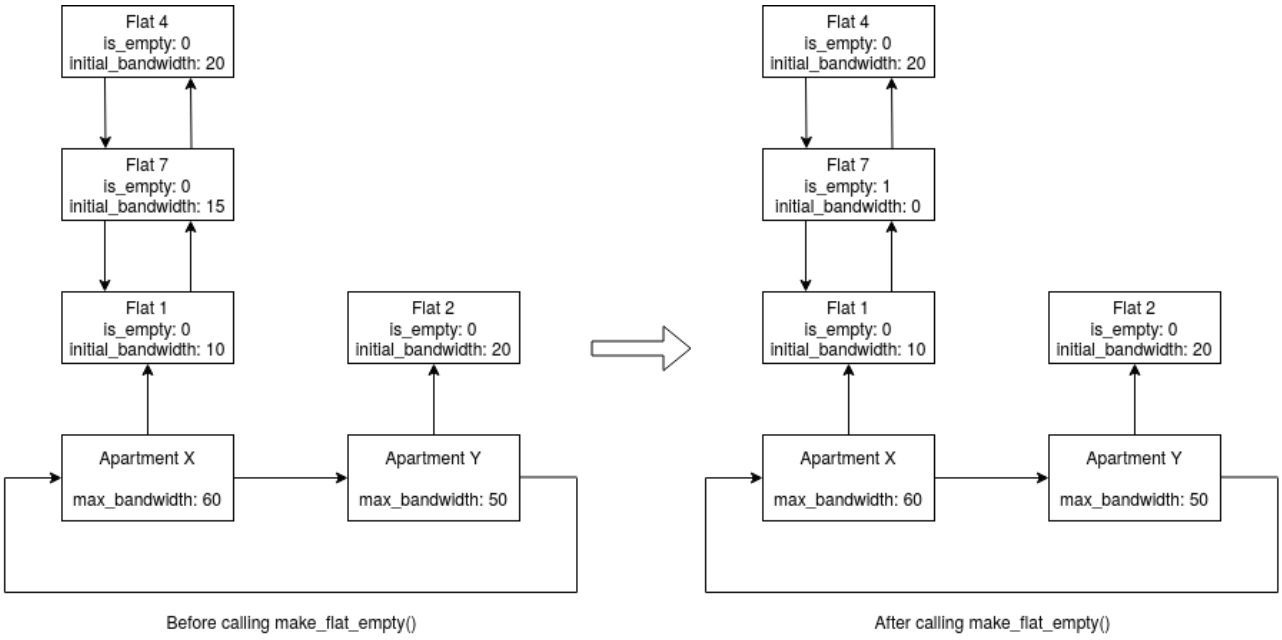
```
void make_flat_empty(Apartment* head, char* apartment_name, int flat_id);
```

Explanation:

- 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.

Example:

```
apartment_name: "X"
flat_id: 7
make_flat_empty(head, apartment_name, flat_id);
```



Before calling make_flat_empty()

After calling make_flat_empty()

3.1.5 find_sum_of_max_bandwidths (10 Points)

Function Declaration:

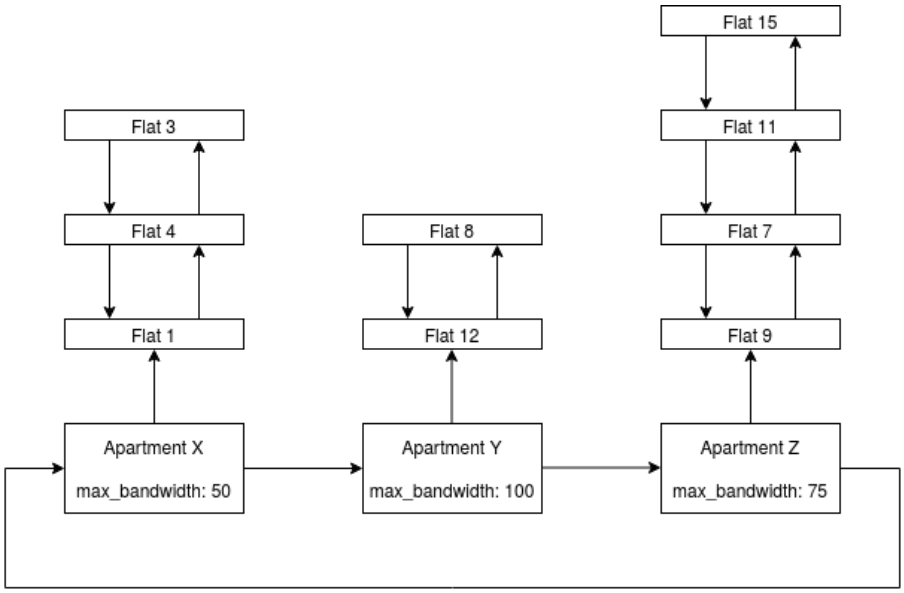
```
int find_sum_of_max_bandwidths(Apartment* head);
```

Explanation:

- 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.

Example:

```
sum = find_sum_of_max_bandwidths(head);
sum: 225 (for the following list)
```



3.1.6 merge_two_apartments (15 Points)

Function Declaration:

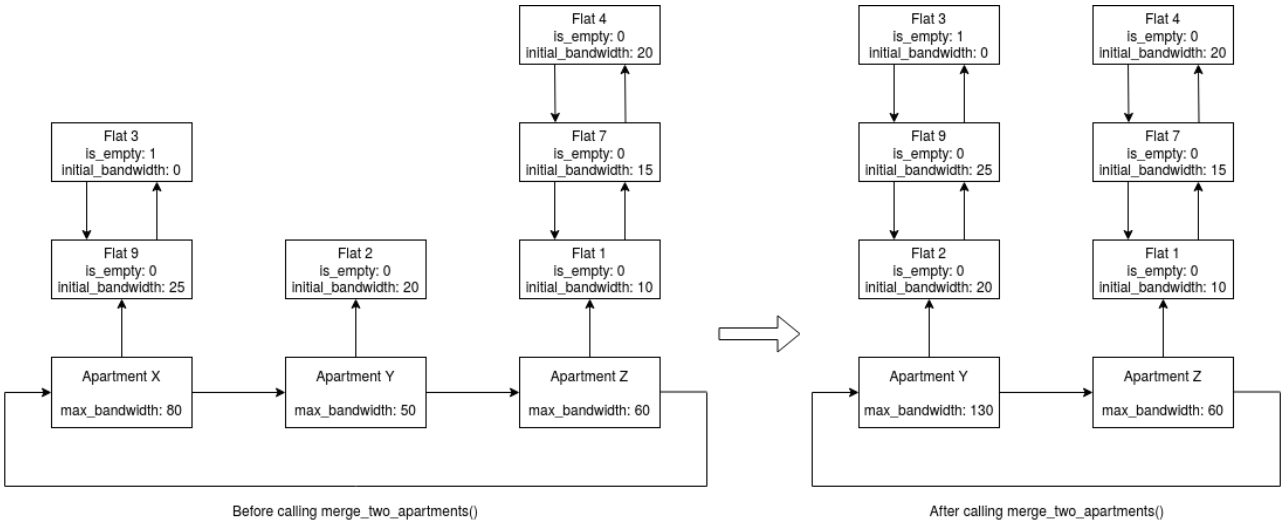
```
Apartment* merge_two_apartments(Apartment* head, char* apartment_name_1,
                                char* apartment_name_2);
```

Explanation:

- 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.

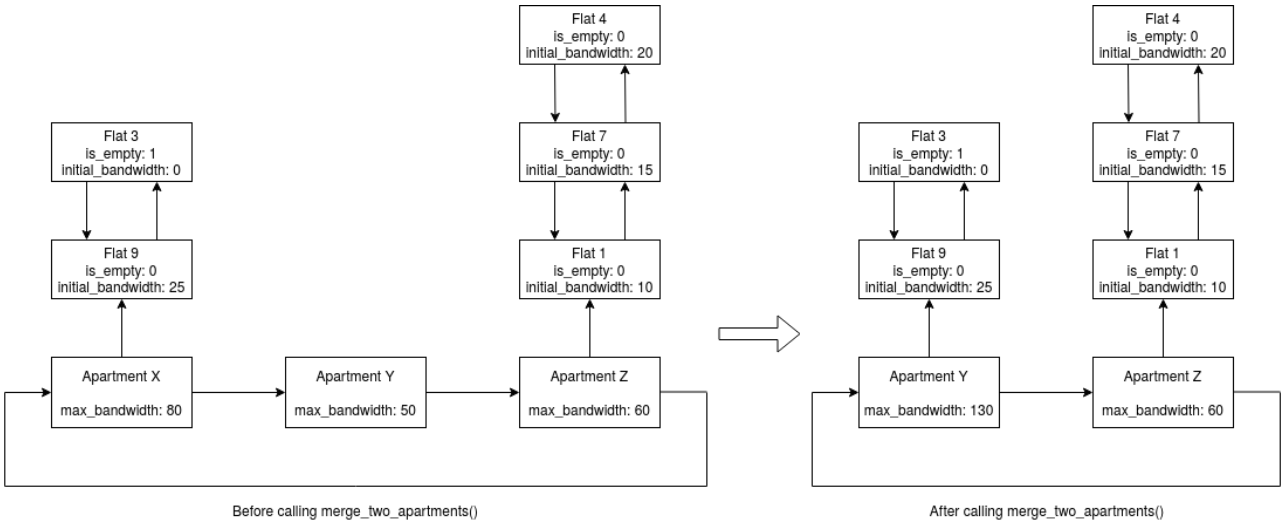
Example - 1:

```
apartment_name_1: "Y"
apartment_name_2 = "X"
merge_two_apartments(head, apartment_name_1, apartment_name_2);
```



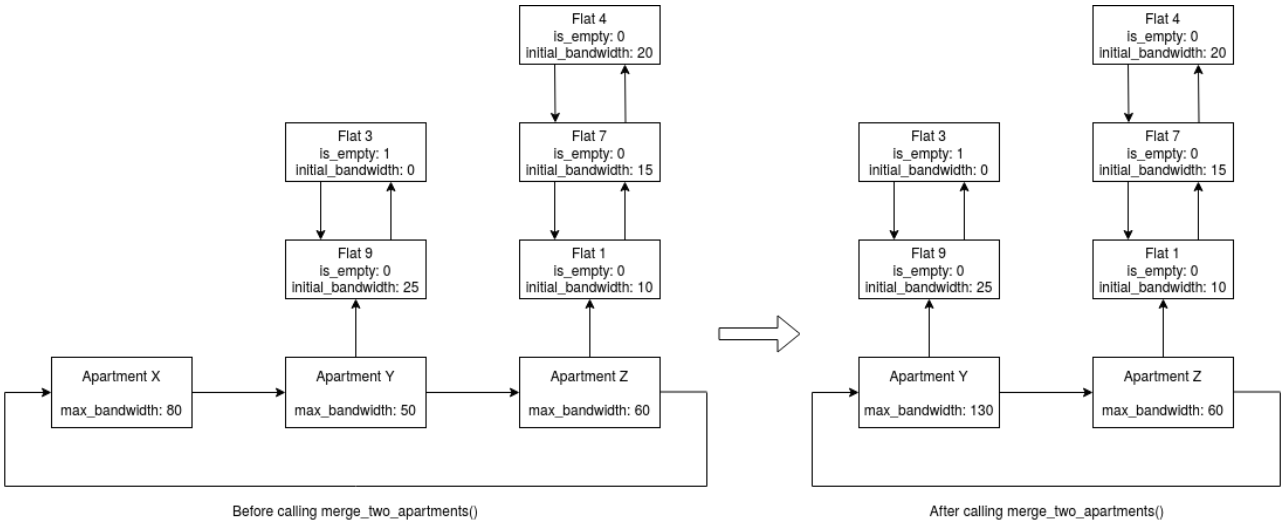
Example - 2:

```
apartment_name_1: "Y"
apartment_name_2 = "X"
merge_two_apartments(head, apartment_name_1, apartment_name_2);
```



Example - 3:

```
apartment_name_1: "Y"
apartment_name_2 = "X"
merge_two_apartments(head, apartment_name_1, apartment_name_2);
```



3.1.7 relocate_flats_to_same_apartment (20 Points)

Function Declaration:

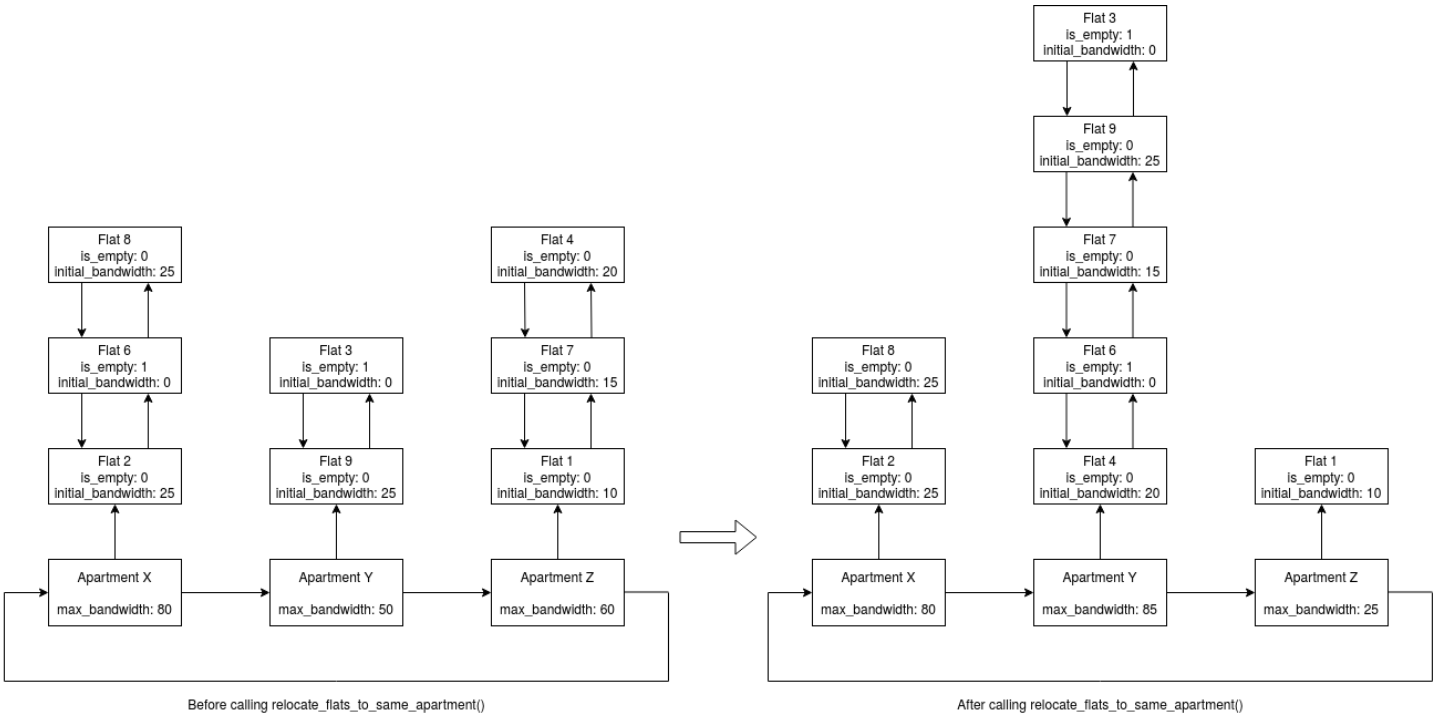
```
void relocate_flats_to_same_apartment(Apartment* head, char* new_apartment_name,
                                      int flat_id_to_shift, int* flat_id_list);
```

Explanation:

- 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 (Apartment* head), in the evaluation steps, this list will be used and compared with the expected list.

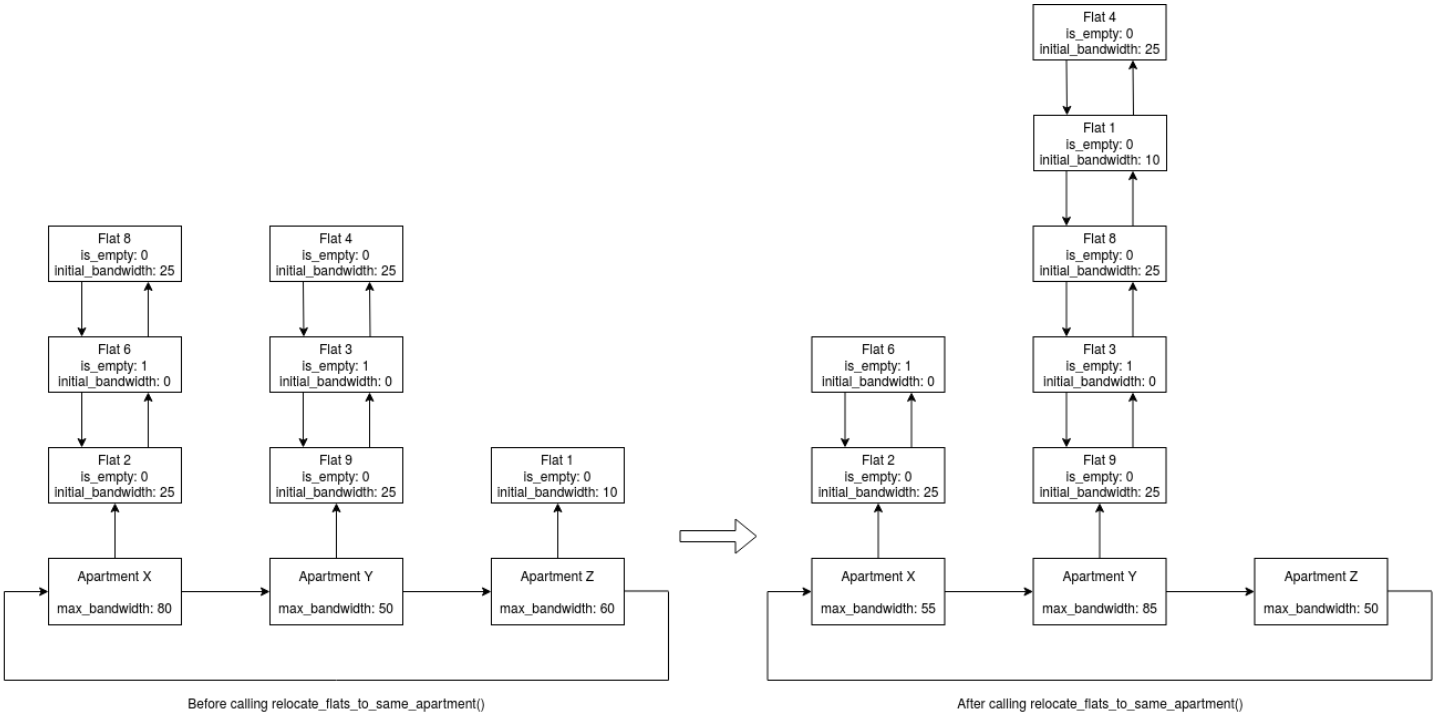
Example - 1:

```
new_apartment_name: "Y"
flat_id_to_shift = 9
flat_id_list = [4, 6, 7]
relocate_flats_to_same_apartment(head, new_apartment_name, flat_id_to_shift, flat_id_list);
```



Example - 2:

```
new_apartment_name: "Y"
flat_id_to_shift = 4
flat_id_list = [8, 1]
relocate_flats_to_same_apartment(head, new_apartment_name, flat_id_to_shift, flat_id_list);
```



3.2 Rules & Regulations

- **This is an individual assignment. Using any piece of code, discussion, explanation, etc. that is not your own is strictly forbidden and constitutes as cheating. This includes friends, previous**

homework, or the Internet. Even if you take only a "part" of the code from somewhere or somebody else, this is also cheating. People involved in cheating will be punished according to the university regulations and will get 0 from the homework.

- **Late submission is NOT allowed and due date is NOT subject to postpone.**
- You cannot use any libraries other than <**stdio.h**>, <**stdlib.h**> and **"the3.h"**.
- You can define your own helper functions.
- As stated before, while evaluating your assignments, your codes will be compiled with the original the3.h file. Therefore, it is recommended that you do not modify the given header file.
- Before submission, you should **test your codes (compile and run) on inek machines** by connecting via SSH. **Do not forget that your assignments will be evaluated on inek machines.**
- Submission will be made via CengClass. Upload a single file named **e<studentID>.c** where <studentID> is your **7-digit** student ID (Example: e1234567.c).
- **Do not write a main function in your source file, it will be provided while testing.**
- Follow the ANSI standards, your codes will be compiled and run with the commands below:

```
$ gcc -ansi -Wall -pedantic-errors e1234567.c test.c -o e1234567
$ ./e1234567
```

- Your assignments will be evaluated with **black box testing**. So, make sure that your functions return exactly the expected output.
- Erroneous inputs will not be tested.
- Follow the announcements on our course page at COW for any updates and clarifications. Please prefer COW firstly for your questions instead of e-mailing if the question does not contain code or a solution. Your question might have already been answered or the answer you get might help your peers.