General instructions:

Read the questions carefully and make sure your plans work accordingly

For each question you will receive an example output for that question, a student who does not follow these examples will have his grade reduced!

Homework is automatically checked according to the examples below,

Claims of various types arising from non-compliance with these requirements will not be considered.

Another highlight:

There is an organized skeleton that helps you not to make mistakes in printing, using it is a must and not a right, a student who nevertheless chooses not to use it and receives outputs that are different from the examples below (even at the character level) will have a score reduced and his appeal on the subject will be canceled.

Exercise 6:

In this exercise you will practice different operations on linked lists.

In the exercise 3 different questions, each of which will come with its own skeleton.

A user interface will allow you to add nodes to the linked list.

The interface uses a WHILE loop whose stopping condition is the insertion of a negative value into the linked list (in the exercise it is defined that there will be **no negative value nodes**).

After the user enters any negative number, the WHILE loop will stop and the interface will continue to run the relevant question.

Please note: the addition of the nodes is done in a way similar to that of a stack (as we saw in Recitation 9):

- The node that the user enters first will be the last node in the linked list.
- The node the user enters last will be the first node in the linked list (HEAD).

**To be clear: there will not be a linked list with 0 nodes**

Question 1:

Write a function that will receive a linked list (not necessarily sorted), and return a new list, where the first nodes are the even nodes of the given list and after the even ones will be the odd ones.

```
Node* EvenOdd(Node* head)
{

}
```

Example:

```
Original Linked list:
0
1
8
667
56
250
16
-1
Modified Linked list:
16 250 56 8 0 667 1
```

Another example:

```
Original Linked list:
1000
1
8
667
56
250
13
-5
Modified Linked list:
250 56 8 1000 13 667 1
```

Pay attention to a few points:

- 0 is an even number.
- As you can see, after the "separation" the even and the odd side are not sorted among themselves. And the order of appearance in the new list is according to the order in the original.

Explanation: Pay attention to the picture of the second example, the even and odd numbers are printed according to the order of appearance in the original list:

In the original list: (even in red, odd in blue)

13,250,56,667,8,1,1000

In the new list: (even in red, odd in blue)

250,56,8,1000,13,667,1

Question 2:

Write a function that will receive a linked list (not necessarily sorted),

and return a new list consisting of the elements of the list, first in reverse order and then in normal order.

```
Node* mirror(Node* head)
{
    
}
```

Example (the numbers were chosen at random):

```
Original Linked list:
1616
25
13
12
-6
Mirror Linked list:
1616 25 13 12 12 13 25 1616
```

More examples:

```
Original Linked list:
1
1
2
1
-1
Mirror Linked list:
1 1 2 1 1 2 1 1
```

```
Original Linked list:
1
-1
Mirror Linked list:
1 1
```

- If the given list has one node, the function will of course return the list of that node twice (as in the example).

Question 3:

In this question, write a function that receives a sorted linked list in which there are duplicate numbers. Write a function that will delete those duplicates.

```
void NoDuplicates(struct node* head)
{

}
```

Example:

```
Linked list before duplicate removal:
11
11
11
13
13
20
20
-1
Linked list after duplicate removal:
20 13 11
```

Another example:

```
Linked list before duplicate removal:
11
11
11
-10
Linked list after duplicate removal:
11
```

- If the list has one node, the function will of course return the node itself.
- If the list consists of one number and duplicates of the same node, the function will of course return the node itself (example 2).
- You will get the prints you see in the picture in the exercise skeleton.

- Of course you are expected to release the memory created at the end of the code run

Important highlights:

- Make sure the code is clean and has no duplicates (it will not damage the grade, but this is a tip for better work)
- A code that does not stop running in less than 5 seconds will not be checked and will be scored 0.
- The 3 programs must be saved as type C files inside the submission boxes of the form:
    - For the first part:
        - ex5_parta_<id_num>.c
        - For example: ex5_parta_123456789.c
    - For the second part:
        - ex5_partb_<id_num>.c
        - For example: ex5_partb_123456789.c
    - For the third part:
        - ex5_partc_<id_num>.c
        - For example: ex5_partc_123456789.c

The deadline for the exercise is until 11/01/2023

Good luck!