Middle East Technical University | Department of Computer Engineering

# CENG 140

## C Programming

Fall '2019-2020
Take Home Exam 2: CENGRAM

Due date: January 10, 2019, Friday, 23:59 - Mail: gorkem@ceng.metu.edu.tr

# 1 Objectives

In this assignment, you are going to implement basic functionalities of simple social network platform called Cengram. This social network platform has simple functionalities such as **register**, **follow user**, **create post**, **remove account**. Your job is to simulate those procedures with your skills on C programming. A sample input file will be given to you for this homework. Don't worry, the code for reading input is given to you already, you will only focus on implementing linked list operations to get the desired output. You are going to have hands-on experience on different topics: *memory allocation*, *structs*, *linked lists*, *strings (char pointers)*.

# 2 Specifications

Overall design is like the following. Read each of the specifications **carefully**:

1. There are two main linked lists in the global scope. *userProfiles* and *allPosts*. *userProfiles* is a linked list, in which there are nodes that contain pointers to User structure. *allPosts* is also a linked list, in which there are nodes that contain pointers to Post structure. You will modify these lists when:

    - **A new user is registered.** When a new user is created, it should be **appended to the end** of *userProfiles* linked list. If there are no users, *userProfiles* pointer should be newly created Node*. If there are users, next node after the last user node should be newly created Node*.
    - **A new post is created.** When a new post is created, it should be inserted to *allPosts* according to the *postId*. If there are no posts, *allPosts* pointer should be newly created Node*.
    - **A user account is removed.** When a user account is deleted, all posts of the user should be removed from *allPosts* and user node should be removed from *userProfiles*.

2. All users have *followers*. *followers* member of User structure is a linked list of user nodes. **When a user follows another user**, a new node containing User* information should be created and appended to the end of the *followers* linked list of the followed user. Keep in mind that **you should NOT allocate new memory for follower User object in follow operation**. You should find existing User* information from *userProfiles* and use it for new follower node to preserve consistency.

3. All users have *posts*. *posts* member of User structure is a linked list of post nodes. **When a new post is created**, a new node containing Post* information should be created and appended to the end of the *posts* linked list of the author user (notice the difference with inserting to *allPosts*). Keep in mind that **you should NOT allocate new memory for new post in post operation**. You should find existing Post* information from *allPosts* and use it for new post node to preserve consistency.

4. Death is inevitable. Some of the users may pass away and accounts get deleted. **When a user account is deleted**:

   - Deleted user's posts should be removed from *allPosts*.
   - User node should be removed from *userProfiles*.
   - *followers* linked list of other users and *posts* linked list of deleted user should be empty (first empty the list and set NULL later).
   - All details about User* should be removed and cleaned completely.

   While doing these deletions, you should also change *totalUserCount*, *totalPostCount*, *user-numOfPosts*, *user-numOfFollowers* accordingly.

# 3 Structures

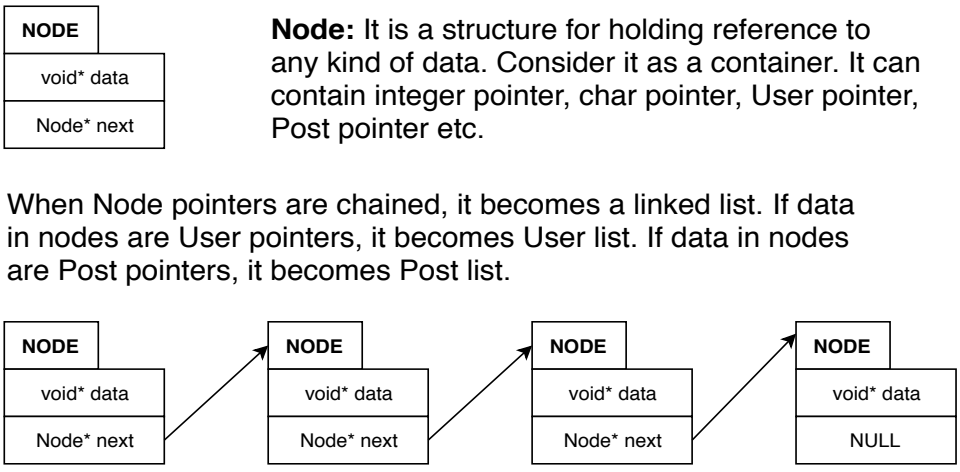


Figure 1: Linked list structure, one linked list to rule them all

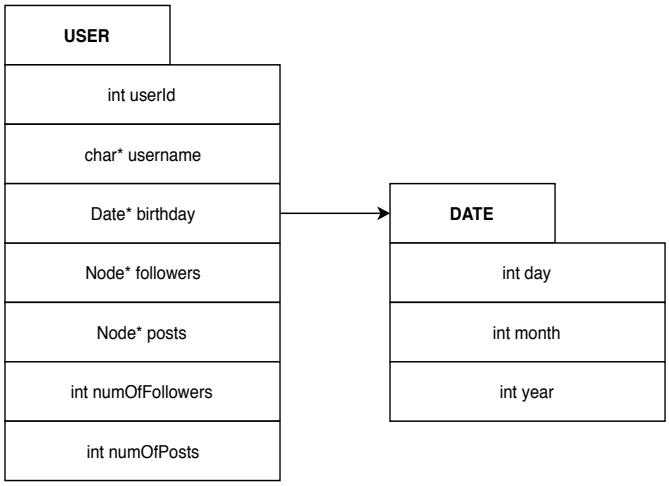


Figure 2: User and Date structures

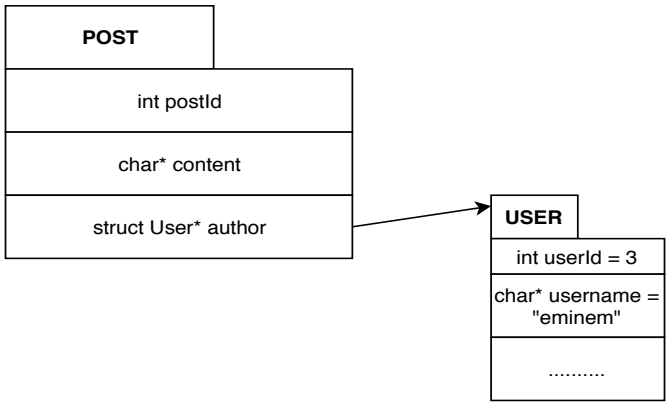


Figure 3: Post structure and author as User*

# 4 Input File

## 4.1 Number of users:

Specifies the number of users that joined to Cengram.

## 4.2 Registering Users (20 points):

Each user has *userId*, *username*, *birthday* information given in each line. Birthday information has *day*, *month*, *year* fields. For instance, below code means that user "eminem" has userId "3" and birthday "17.10.1972". Also "tupac" has userId "5" and birthday "16.6.1971". You need to use **Date structure** for storing birthdays and **User structure** for storing users. All of the structures are given to you in **"Cengram.h"** file.

```
3
eminem
17 10 1972
```

```
5
tupac
16 6 1971
```

In addition to these, you need to register these users into the *userProfiles* list in the global scope. **You can just append at the end of *userProfiles* list.** Also, *totalUserCount* variable in the global scope should be incremented.

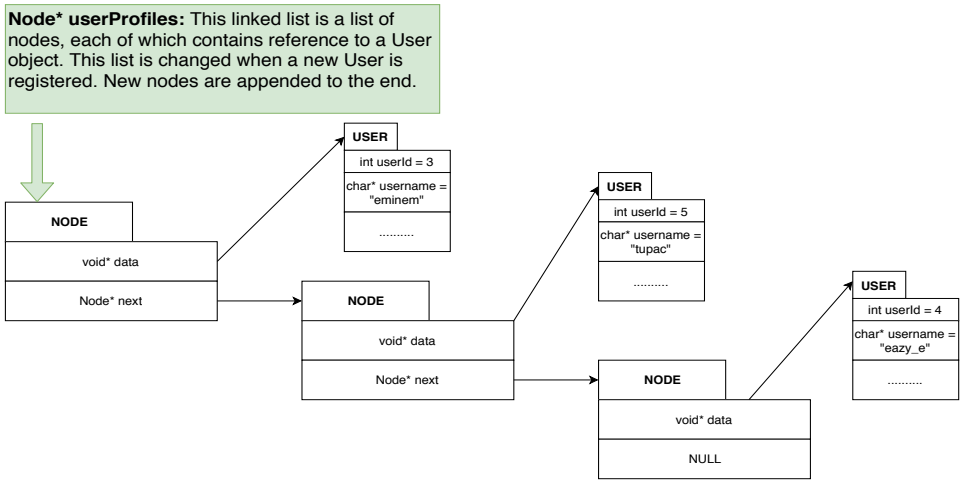


Figure 4: Management of userProfiles in the global scope

## 4.3 Number of follows:

Specifies the number of follow actions between users of Cengram.

## 4.4 Following users (25 points):

Each following action has 2 parameters: **following user id** and **followed user id**. For instance, as you can see in the example below, user1 follows user4 and user5 follows user2.

```
1 4
5 2
```

When users follow someone, following user information should be appended to the *followers* list of followed user. **You can just append at the end of *followers* list.** Also, *numOfFollowers* member of the followed user should be incremented.

## 4.5 Number of posts:

Specifies the number of created posts belong to the users of Cengram.

**Node* followers:** This linked list is similar to userProfiles. It is changed when another User follows current User. New nodes are appended to the end.

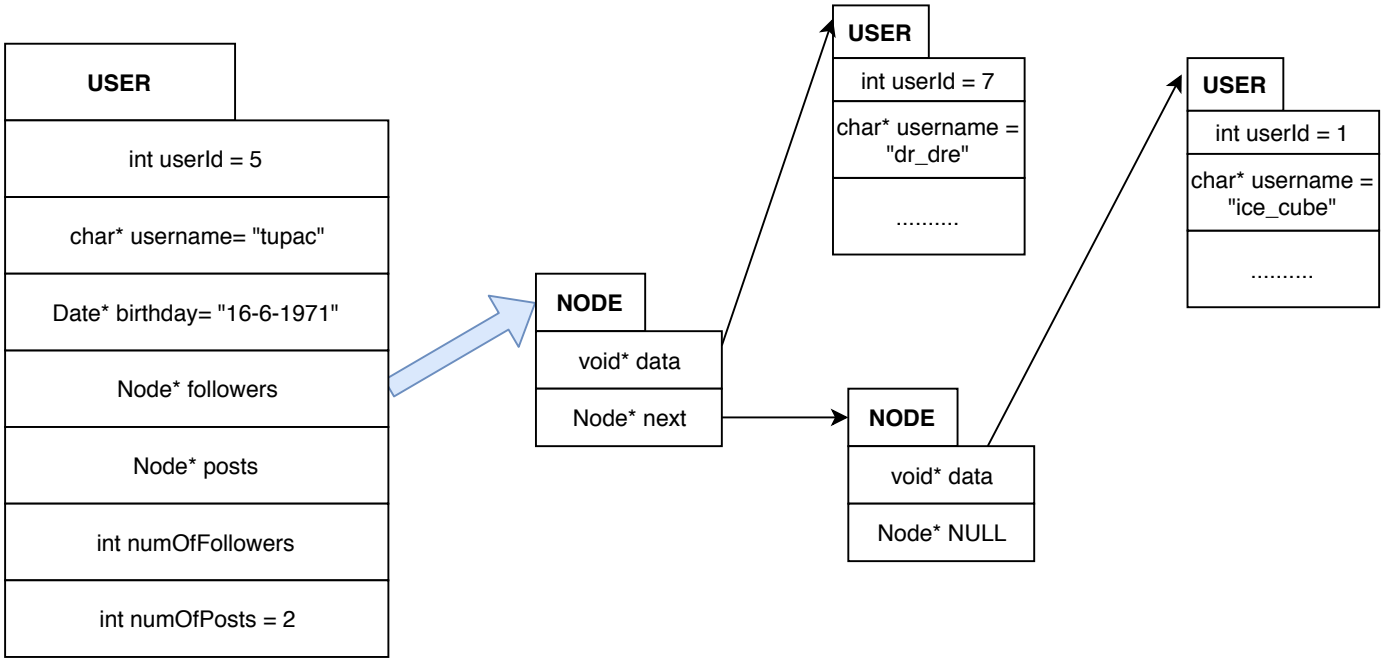


Figure 5: Management of followers member of User

## 4.6 Creating Posts (25 pts):

Each post is defined with a line containing *postId*, *ownerUserId*, number of characters in post *content*. The next line contains the post content. The example input below states that user5 wrote a post with postId "6" and this post contains "107" characters.

```
6 5 107
It ain't about black or white 'cause we human. I hope we see the light before
it's ruined, my ghetto gospel
```

When users make posts, post information should be appended to the *posts* list of author user. **You can just append at the end of *posts* list.** Also, *numOfPosts* member of the author should be incremented.

Furthermore, you need to add new post information to the *allPosts* list in the global scope. **For this list, you need to preserve post order by their *postId*s. At the end of the program execution, *allPosts* should be in ordered by *postId*s, in ascending order.** Finally, don't forget to increment *totalPostCount* in the global scope.

**Node* posts:** This linked list is a list of nodes, each of which contains reference to a Post object. This list is changed when a new Post is added. New nodes are appended to the end.

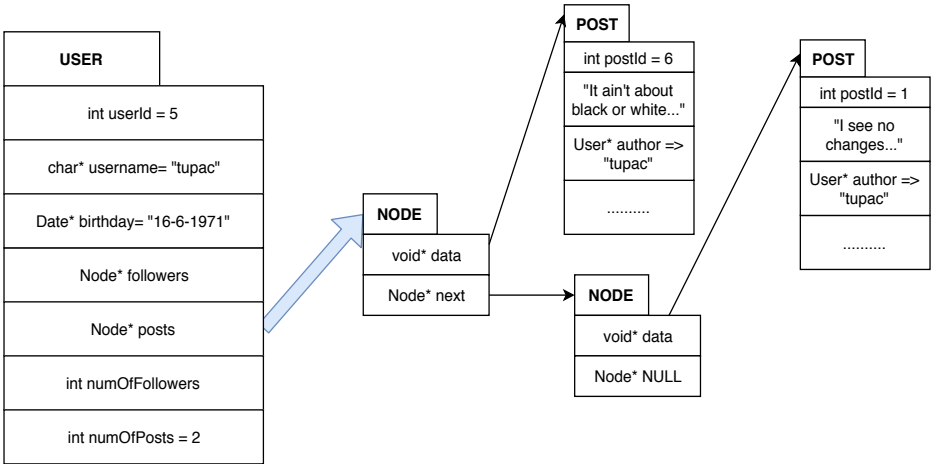


Figure 6: Management of posts member inside User structure

**Node* allPosts:** This linked list is a list of nodes, each of which contains reference to a Post object. This list is changed when a new Post is registered. **New nodes are inserted with respect to their postIds, to preserve ascending order.**

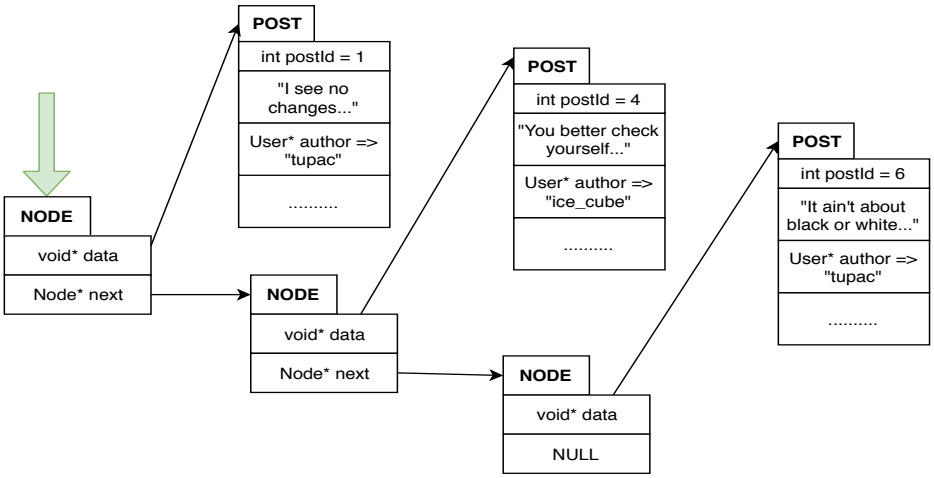


Figure 7: Management of allPosts in the global scope

## 4.7 Number of removed users:

Specifies the number of users who later on decided to delete their accounts from Cengram.

### 4.8 Remove User Account (30 points):

Each integer in the line is a deleted user id. Remove users **COMPLETELY** from the system. Below lines say that 2 user accounts are removed, one of them is user5 and the other one is user4.

```
2
5 4
```

Before you remove the user, you need to remove all references to the user, remove its posts, remove references to deleted posts, remove birthday information, remove username information etc. By completely means that you will leave no clue behind when you delete the user.

## 5 Clarifications, Hints & Tips

1. Start early!
2. Linked list in this homework can be used for storing list of any data structure. It can be confusing at first. However, nobody creates different structure for similar operations on different things. In real life, code repetition is avoided.
3. Go step by step. You can implement appending node to the end, and change it to appending node according to member comparison, you can implement finding a node with respect to a given property etc.
4. To preserve consistency about the data, you have to work on same User and Post pointers for *userProfiles*, user's *followers*, *allPosts*, user's *posts.*
5. You can write your helper functions in Cengram.c file. However, **you can not add new files or you can not change any of the structures given in Cengram.h.**
6. You should create a new Node for adding follower to user=>followers. You should find existing User* from *userProfiles* and use that pointer to fill *data* member of the newly created Node*. **Don't allocate a memory for User* again, only allocate memory for new Node*.**
7. You should create a new Node for adding new Post's to user=>posts. You should find existing Post* from *allPosts* and use that pointer to fill *data* member of the newly created Node*. **Don't allocate a memory for Post* again, only allocate memory for new Node*.**
8. Since Node structure has a member *data* which is a *void pointer*, you need to do **type casting** to read its values as User or Post.

## 6 Regulations

1. **Compiling and Running:** You should compile and run your program with the following commands:

```
>_ gcc Cengram.c Main.c -ansi -pedantic-errors -Wall -o hw2
>_ ./hw2 < input.txt
```

2. **Submission:** Submission will be done via CengClass. Before submitting your code, **make sure that you can compile and run your codes on department's inek machines.** Even if your codes work fine in your local machine, if it doesn't work on inek machines, you will get 0. Finally, please remember that **late submission is NOT allowed.** You should submit your **Cengram.c** in a .zip file and name it like the following:

   ```
   e1234567_the2.zip
   ```

   Where you should replace "1234567" with your own student number.

3. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations and will get 0. Please be aware that there are "very advanced tools" that detect if two codes are similar. So please do not think you can get away with by changing a code obtained from another source.

4. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.