Middle East Technical University Department of Computer Engineering

CENG 140

C Programming

Spring '2023-2024
Take Home Exam 3

Due date: 11.06.2024, Tuesday, 23:59

1 Objectives

In this assignment, you are going to simulate basic functionalities of a simple social network platform. This social network platform has simple functionalities such as **follow user** and **like post**. However, you are also supposed to implement a **cache** in front of your database to speed up the access. Your job is to simulate these 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**:

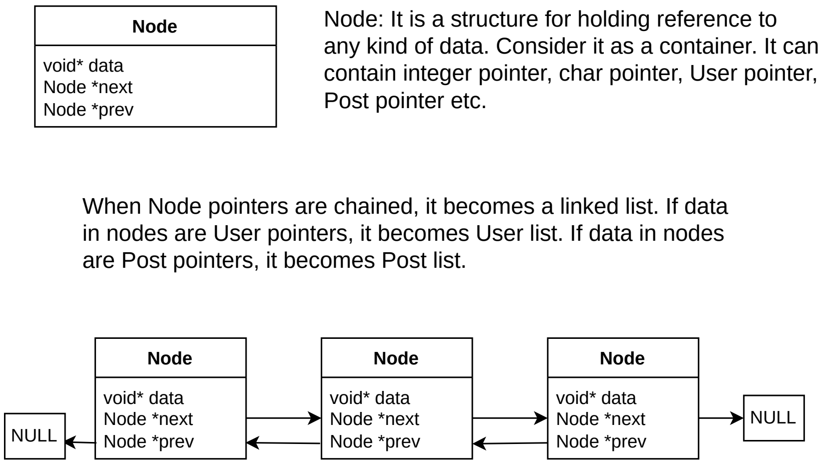
1. There is a global struct called "db", short for database, which keeps all of the social media's data. It has two arrays: *users* and *posts*. You will not directly access these arrays from outside of the caches. These structures are merely for keeping the input read from stdin.
2. There will be two other global structs called UserCache and PostCache 4. These will be initially empty and they will keep the recently accessed users and posts, respectively. Their capacities are limited so if they become full (their size becomes equal to their capacity) and you need to add another entry, you will need to remove the oldest entry.
3. UserCache holds a linked list 1, in which there are nodes that contain pointers to recently accessed User structures.
4. PostCache holds a linked list 1, in which there are nodes that contain pointers to recently accessed Post structures.
5. You will add entries to these caches on two occasions:

- **When a user follows a user.** Say, user 2 follows user 5 then you need to request these users from the UserCache. If these users are not found in the cache, then they are retrieved from the database into the cache. Only then the users are returned to the caller.
- **When a user likes a post.** Say, user 3 likes the post with id 10. Then you need to retrieve user 3 and post 10 from UserCache and PostCache, respectively. If this user and post are not found in their respective caches, they are retrieved from the database by the cache and then inserted into the cache. Only then the user and post are returned to the caller.

For no other reason will there be an insertion, removal or move of one of the cache entries, ie. printing a cache does not change the caches in any way.

6. You will only remove entries from these caches when the caches are full but we must add a new entry to the cache. For example, user 2 followed user 5. You will first retrieve user 2 from the UserCache and then user 5. User 2 is in the cache, it is moved up to the front of the UserCache users linked list, and then user 2 is returned to the caller. Then, user 5 is requested but it is not in the cache and also the cache is full. Now, the tail of the linked list will be evicted and user 5 will be put to the head of the linked list. Only then user 5 will be returned. Similarly, when a user likes a post, the tail of the PostCache linked list may be evicted and a new post may be put to the head of the linked list.
7. All users have *followers*. *followers* member of User structure 2 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 use the existing User* information you had retrieved from the cache and use it for new follower node to preserve consistency. Do not forget to increase the following user's following count and the followed user's follower count.
8. When a user likes a *post* 3, you need to increment post's like count and user's liked count after retrieving the user and the post from the UserCache and the PostCache, respectively. Also accessing this user and post, moves them to the head of their respective linked list if they are already in the linked list, otherwise they are retrieved from the database.
9. After each like and after each follow, you must print the UserCache entries first and then the PostCache entries. To help you with what to print, printUser and printPost methods are given.
10. After all actions are over, "printFollowersInReverse" function will be called. In this function you must iterate UserCache entries in reverse and print their followers in reverse. printf format strings are given for your ease in socialmedia.c.
11. PostCache will be printed in tail to head, UserCache will be printed in head to tail order.
12. "freeLinkedList" method will take the head of a linked list and free all its nodes but not data of its nodes.

3 Structures


Node: It is a structure for holding reference to any kind of data. Consider it as a container. It can contain integer pointer, char pointer, User pointer, Post pointer etc.

When Node pointers are chained, it becomes a linked list. If data in nodes are User pointers, it becomes User list. If data in nodes are Post pointers, it becomes Post list.

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

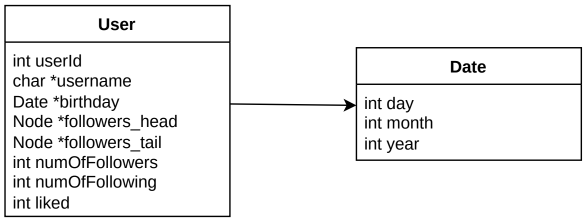


Figure 2: User and Date structures

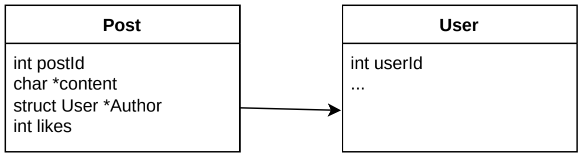


Figure 3: Post structure and author as User*

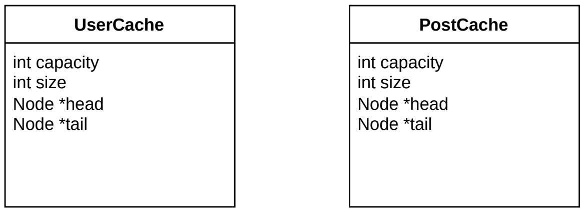


Figure 4: UserCache and PostCache

4 Input File

4.1 Cache Capacities:

Specifies the UserCache and PostCache capacities. The example below specifies 10 and 20 capacities for UserCache and PostCache, respectively.

```
    10 20
```

4.2 Number of users:

Specifies the number of users that joined to social media. That is, how many users you should expect to read in the next subsection.

```
    2
```

4.3 Reading Users:

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 **"socialmedia.h"** file.

```
3 eminem 17 10 1972
5 tupac 16 6 1971
```

You will put these users into their right place in the database. For example, eminem has id 3, thus it is at "database.users[3]".

4.4 Number of posts:

Specifies the number of created posts belong to the users of social media. That is, how many posts you should expect to read in the next subsection

```
    1
```

4.5 Reading Posts:

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
```

4.6 Number of actions:

Specifies the number of actions.

```
4
```

4.7 Following users:

Each following action has 3 parts: Identifier string "F", **following user id**, and **followed user id**. For instance, as you can see in the example below, user1 follows user4 and user5 follows user2.

```
F 1 4
F 5 2
```

When users follow someone, following user information should be appended to the *followers* list of followed user. **You must append at the end of *followers* list.** Also, *numOfFollowers* member of the followed user and *numOfFollowing* member of the following user should be incremented and the *followers_tail* member of the followed user must be updated.

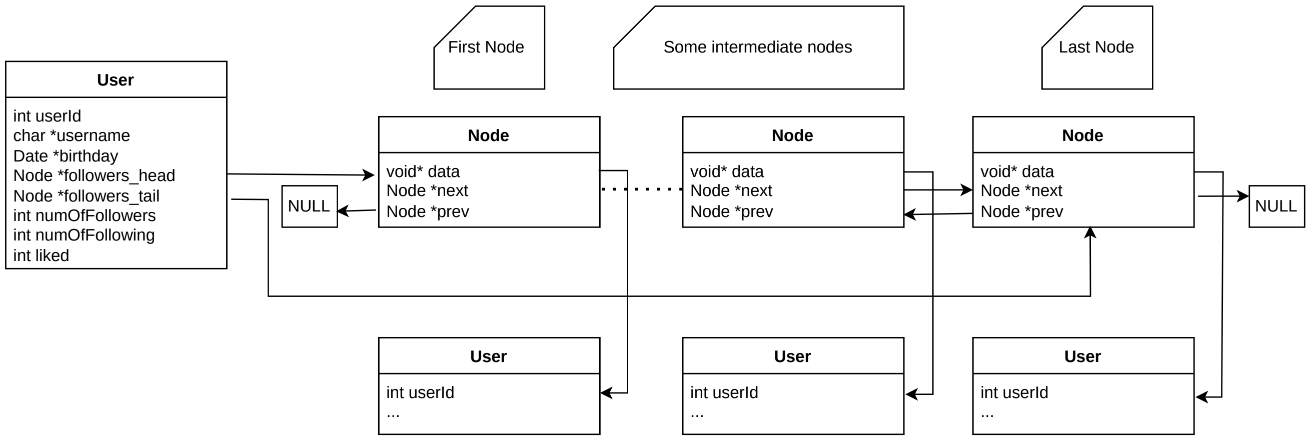


Figure 5: Management of followers member of User

You must make exactly two fetches from the UserCache in the described order. You need to first request the following user from the cache and then request the followed user from the cache so that cache is updated as expected by the grader. For example, for the first follow action given above, you fetch user1 and it becomes the head of the UserCache linked list and then you fetch user4 and it becomes the new head of the UserCache linked list. Thus, UserCache has "user4 - user1 - ..." in its linked list in this order.

After completion of each action, you must print UserCache. For example, you should print once after user1 follows user4 and once more after user5 follows user2. This is already given to you in main function.

4.8 Liking posts:

Each like action has 3 parts: Identifier string "L", **user id that likes**, and **post id that is liked**. For instance, as you can see in the example below, user6 likes post1 and user7 likes post2.

```
L 6 1
L 7 2
```

When a user likes a post, the number of likes the user has given must be incremented. When a post is liked, the number of likes it received must be incremented. Other user and post struct fields needs not to be changed.

You must make exactly one fetch from the UserCache and exactly one fetch from the PostCache. Making extra fetches may cause unwanted results.

After completion of each action, you must print UserCache first and then PostCache. For example, you should print once after user6 likes post1 and once more after user7 likes post2.

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. You can write your helper functions in socialmedia.c file. However, **you can not add new files or you can not change any of the structures given in socialmedia.h.**
4. You should create a new Node for adding follower to user=>followers. You should find existing User* from the UserCache 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*.**
5. You should create a new Node for adding new Post's to user=>posts. You should find existing Post* from PostCache 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*.**
6. 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.

7. Since fetch operations may change the entries and their order in the cache, you should not make unnecessary fetch operations. You should also respect the order of fetch operations given in the homework.

8. A post can be liked more than once by the same user.

6 Regulations

1. **Compiling and Running:** You should compile your program with the given Makefile. It contains compile flags that you should respect. Your submission must be compilable by this Makefile.

2. **Submission:** Submission will be done via ODTUClass. 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 socialmedia.c using VPL.

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. **Discussion Forum:** You must follow the ODTUClass THE3 Discussion for updates on a daily basis.

5. **Grading:** Your methods will be called in isolation and your structures will be checked for correctness.

   - Append to cache: 10 pts
   - Move to front: 10 pts
   - Remove Last: 10 pts
   - Fetch User / Fetch Post: 10 pts
   - Follow User: 10 pts
   - Like Post: 5 pts
   - Free Linked List: 10 pts
   - Print Followers in Reverse: 5 pts
   - Randomized mixture tests: 30 pts

   Received points / Maximum possible points * 100 will be your grade. You are expected to handle edge cases, ie. attempt to remove the last item while the linked list is empty.