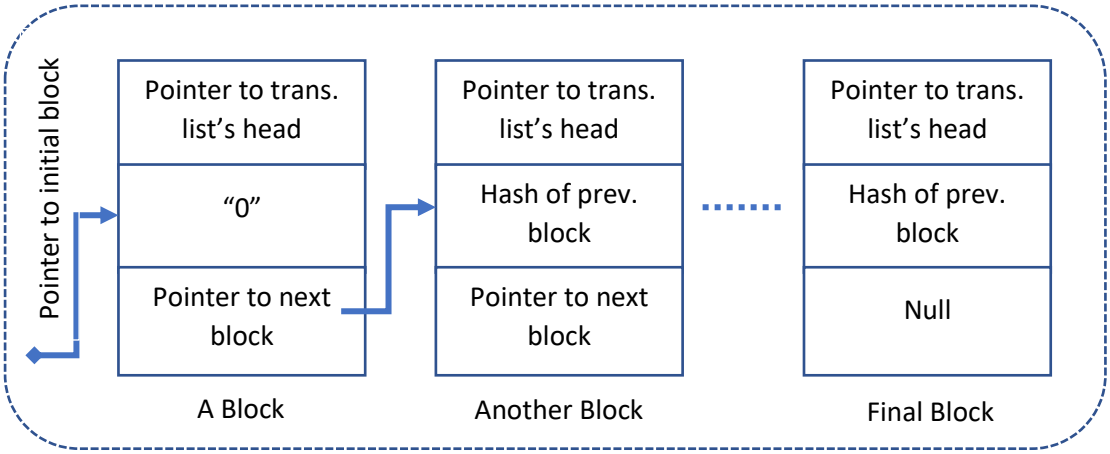
ITU Computer Engineering Department
BLG 223E Data Structures, Spring 2022
Recitation #12

Definition

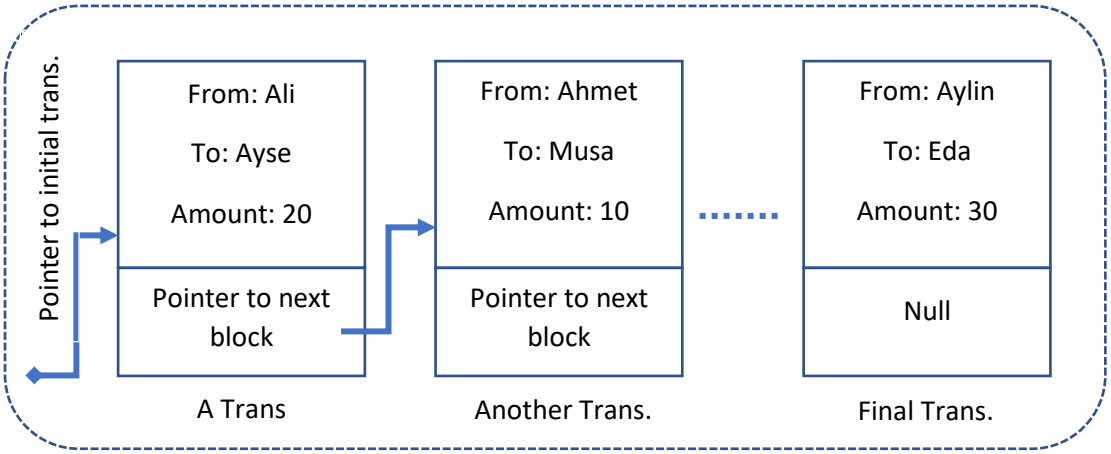
You are asked to implement a very simple block list ledger system that can keep a list of pseudo-monetary transactions among a group of peers in an unmodifiable way. Please pay attention to the following details in your implementations:

- Please read the material in the following link to get an idea about how your system is going to work. You are expected to build a similar but simpler system. DO NOT COPY PASTE CODE FROM THE FOLLOWING LINK
    - https://davenash.com/2017/10/build-a-blockchain-with-c/
- Your program is going to start with a group of individuals where each of the individual has a balance represented with an integer variable.
- During your program run, a series of transactions between pairs of individuals are going to be kept inside your block list's node (**via an additional level of linked list**). Your block list should look like the following diagram:



Block List Structure

- Each block should keep the series of transactions for this block in a linked list where each node contains a transaction between a pair of individuals. An example transaction list should look like the following diagram:



Transaction List Structure

- In your block list, when a block is added, it is going to calculate and include the MD5 hash information of the previous block in itself as a string. Initial block is going to contain the string "0" as the previous block's hash.
- It is mandatory to use the MD5 hash library **provided in the homework announcement**. You can obtain information about the library in the following address:
    - https://create.stephan-brumme.com/hash-library/
    - An example code snippet is provided below:

```
#include "md5.h"

MD5 md5;
string hash = md5("Hello");
```

- In order to generate the hash a block, it is mandatory to use the following approach:
    - concatanate the previous block's hash value to an empty string (do not put any empty characters in between, concatanate directly)
    - concatane a string representing each transaction to the string in the previous step (do not put any empty characters in between, concatanate directly)
    - A transaction list's node should be represented as a string as follows:
        - <form><to><amount>
        - For instance the first trans. node in the above diagram should be represented as a string as: AliAyse20
- At the end of your program you should display hash information of the last block or the total amount after all the transactions has been processed according to the input file's content
- Your program should read the individual, balance and transactions inormation from a file, defined in the following section.

Input-Output

Your program should accept a filename as a command line parameter. In the accepted file, each line will contain a distinct command and parameters that should be processed by your program. In the input, the following commands may be given:

- Each file starts with an initial line containing a series of single word names followed by balances as integers.
- BLOCK (<from><to><amount>)+: Each line starting with the keyword block defines the transacitons in a block that should be added to the block list. Blocks and transactions for each block should be kept in the order they are presented in the file. (<from><to><amount>)+ means there is going to be at least one transaction but there may be several as well.
- BALANCE: Print an alphabetically ordered list of individuals and balances after all the transactions in all the blocks have been orderly issued. (see calico file and inputs)
- CLEAR: Clear the block list, all the previous blocks and transactions need to be removed.
- PRINT: Print the hash value kept in the block list's tail node. If the list is empty print the string "NA".

Only the BALANCE and PRINT commands should produce output. The output they need to produce is explained in the previous definitions.

The input file would be error-free both syntactically and semantically, however transactions may result in negative balance values.

Deliver

Please zip and deliver the directory structure defined below:

- HW3 : Topmost folder, that will contain all the folders in your submission. No other files should be present under this folder in your submission.
- HW3/src: Contains all the *.cpp files
- HW3/src/main.cpp: Contains your main function.
- HW3/src/Node.cpp: Contains your transaction list's node's definition.
- HW3/src/BlockNode.cpp: Contains your block list's node's definition.
- HW3/src/BlockList.cpp: Contains your block list class.
- HW3/src/md5.cpp: The md5 library provided within the homework definition. You should put delive this file as well for your submission to compile.
- HW3/include: Contains all the header files you use
- HW3/bin: An empty directory that will contain objective files when your project is compiled

Please check the calico test file in the homework definition to see how your files will be compiled and tested.

Restrictions and Guidelines

- Compilation environment: Only the code that can be compiled by the environment of the container definition provided in ninova will be accepted.
- Testing of your program will be performed using Calico (https://calico.readthedocs.io/en/latest/). Test cases that will be used to test your homework is provided as an attachment in ninova.
- **STL usage is NOT allowed.**
- **This homework is for individual submissions**. Any kind of code sharing or code adaptation from an external source is strictly forbidden. Submitted code will undergo a plagiarism check process and plagiarsim penalties may be given if the submitted code's similarity is approved by the instuctor.
- Make sure you write your name and number in all of the files of your project, in the following format:
  /* @Author
  Student Name:<studentname>
  Student ID :<studentid>
  Date:<date>*/
- Only electronic submissions through Ninova will be accepted no later than deadline
- Use comments wherever necessary in your code to explain what you did.