Последна редакция: **18.03.2023 г.**

Общи изисквания към домашната работа

Решенията, които предавате, трябва да отговарят на следните общи условия и ограничения:

1. Използвайте ресурсите (процесор, памет, диск) **ефективно**.
2. Постарайте се да подсигурите коректна работа на програмата ви с всякакви входни данни (включително некоректни).
3. Можете да използвате класове, но следвайте добрите практики, обсъдени на лекции.
4. Решение, което **не се компилира** или **грубо нарушава** добрите практики, ще бъде оценено с **0 точки**.
5. Предавайте решенията си като **ZIP архив**, който съдържа само изходни файлове **(.cpp и .hpp)**. Името на ZIP архива трябва да е **spec_groupX_fnY.zip**, където **spec** е специалността Ви (I или IS), **X** е номерът на групата ви, а **Y** е факултетният Ви номер.
6. 6. Не е позволено използването на класа std::string, както и на контейнерите от STL.

OOPCoin (better than FMICoin)

Напишете програма за работа с блокчейн валутата OOPCoin. Програмата трябва да поддържа създаването на потребители и трансакции между тях, като за целта разпознава и изпълнява определени команди, въведени от командния ред.

Създаването на потребител се осъществява с командата `create-user`. Всеки потребител се идентифицира с име и сума, която инвестира в OOPCoin. При създаването на нов потребител за него се генерира уникален идентификатор. След създаването на потребителя се създава и системна трансакция за захранване на сметката му. Тази начална трансакция се изпълнява от системния потребител с идентификатор 0. Трансакцията превежда OOPCoin, равно на 420 монети, за всеки един лев, който потребителят инвестира.

Премахването на потребител е възможно с командата `remove-user`. Тя получава като параметър име на потребителя и ако то е валидно, премахва потребителя от системата. Всички негови OOPCoin се прехвърлят с трансакция на системния потребител с идентификатор 0.

Програмата трябва да поддържа създаването на трансакции (`send-coins`) от потребител до потребител, стига изпращачът да има достатъчно налични монети. Записите за трансакциите съдържат идентификатори на изпращача и получателя, количество монети и времето, в което е направена трансакцията. Времето се изчислява като брой секунди, изминали от началото на 1970 до момента на създаване на транзакцията.

Непразна поредица от трансакции наричаме блок от трансакции. Всеки блок съдържа уникален идентификатор, идентификатор на блока предшественик и хеш-а на блока предшественик. В един блок се записват не повече от 16 трансакции. Така блоковете от трансакции реализират верига, като всеки следващ знае кой е предшестващият и какъв е хешът му. За първият блок идентификаторът на предшественикът му съвпада с идентификаторът на блока, а хешът на предшественика се игнорира.

Вашата програма трябва да записва всички данни в 2 двоични (бинарни) файла - `users.dat` и `blocks.dat`. В `users.dat` са записани всички регистрирани потребители, а данните в blocks.dat са блоковете от трансакции.

При стартирането на програмата трябва да заредите данните от файловете, ако те съществуват, и да ги използвате при изпълнението на програмата. При изход (команда `exit`) вашата програма трябва да се подсигури, че във файловете на диска има последните промени, които потребителят е направил.

Трябва да поддържате команда за верификация (`verify-transactions`) на данните за трансакциите. Верификацията проверява, че всички блокове са валидни и няма блокове, които са извън веригата. Всеки блок, освен първият, трябва да реферира валиден предшественик и хешът на предшественика трябва да е правилен. Възможно е да има повече от 1 блок с еднакъв хеш - това не е проблем. Във всеки блок трябва да има поне 1 трансакция.

Трябва да поддържате команди за извеждане на най-богатите потребители (`wealthiest-users`), както и на най-скъпите блокове (`biggest-blocks`). За всеки потребител трябва да се изведе името му и количеството монети, които е натрупал в резултат на трансакциите. За всеки блок трябва да се изведе общото количество монети и поредният номер на блока. И двете команди трябва да приемат аргумент число, което представлява броя записи (хора или блокове), които трябва да се изведат. В резултат от изпълнението на командите трябва да се генерират текстови файлове, чието име се образува от името на командата и текущото вйреме в секунди. Те съдържат данните за съответните потребители или блокове, по един на ред.

Вашата програма трябва да използва ресурсите на машината (процесор, памет, диск) ефикасно.

Използвайте описаните по-долу структури и функция за хеширане наготово, **без да ги променяте**:

```
unsigned computeHash(const unsigned char *memory, int length) {
   unsigned hash = 0xbeaf;

   for (int c = 0; c < length; c++) {
       hash += memory[c];
       hash += hash << 10;
       hash ^= hash >> 6;
   }

   hash += hash << 3;
```

```
    hash ^= hash >> 11;
    hash += hash << 15;
    return hash;
}

struct User {
    unsigned id;
    char name[128];
};

struct Transaction {
    unsigned sender;
    unsigned receiver;
    double coins;
    long long time;
};

struct TransactionBlock {
    unsigned id;
    unsigned prevBlockId;
    unsigned prevBlockHash;
    int validTransactions;
    Transaction transactions[16];
};
```