CSED101. Programming & Problem solving

Spring, 2019

Programming Assignment #5

노융두(yoongdoo0819@postech.ac.kr)

- **Due: 2019.05.31 23:59**
- ***Development Environment*: Windows *Visual Studio 2017***

- ***제출물***
  - **C Code files** (`assn5.c, user.c, user.h, trans.c, trans.h, block.c, block.h`)
    - 제출시, 모든 파일을 하나의 파일로 압축해서 제출할 것. (압축파일명:`assn5.zip`)
    - 프로그램의 소스 코드를 이해하기 쉽도록 반드시 **주석**을 붙일 것.
  - **보고서 파일** (.docx or .hwp) 예) assn5.docx 또는 assn5.hwp
    - AssnReadMe.pdf 를 참조하여 작성할 것.
    - **명예서약(Honor code)**: 표지에 다음의 내용을 포함한다. "나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다." 보고서 표지에 명예서약이 없는 경우는 과제를 제출하지 않은 것으로 처리한다.
    - 소스코드와 보고서 파일을 LMS를 이용하여 제출한다.

- ***주의사항***
  - 각 문제에 해당하는 요구사항을 반드시 지킬 것.
  - 모든 문제의 출력 형식은 아래의 예시들과 동일해야 하며, 같지 않을 시는 감점이 된다.
  - 각 문제에 제시되어 있는 파일이름으로 제출 할 것. 그 외의 다른 이름으로 제출하면 감점 또는 0점 처리된다.
  - 컴파일 & 실행이 안되면 무조건 0점 처리된다.
  - **하루 late시 20%가 감점되며, 3일 이상 지나면 받지 않는다. (0점 처리)**
  - 부정행위에 관한 규정은 POSTECH 전자컴퓨터공학부 학부위원회의 'POSTECH 전자컴퓨터공학부 부정행위 정의'를 따른다. (LMS의 과목 공지사항의 제목 [document about cheating]의 첨부파일인 disciplinary.pdf를 참조할 것.)
  - 과제 작성시 전역변수는 사용할 수 없으며, 사용자 정의 함수를 적절히 작성하도록 한다.
  - 이번 과제에서는 추가 기능 구현에 대한 추가 점수는 없습니다.

■ Problem: 비트코인(Bitcoin)

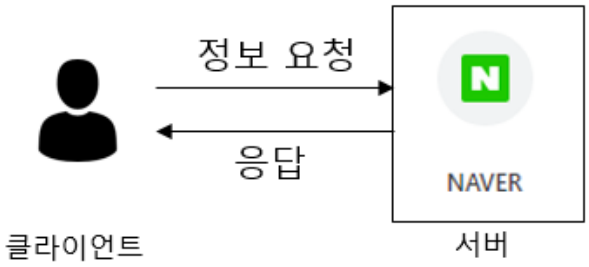
[목적]

- Bitcoin 및 Blockchain에 대하여 간단히 학습하는 시간을 갖는다.
- Structure와 Linked List의 사용법을 익힌다.
- 다중 소스파일의 사용법을 익힌다.

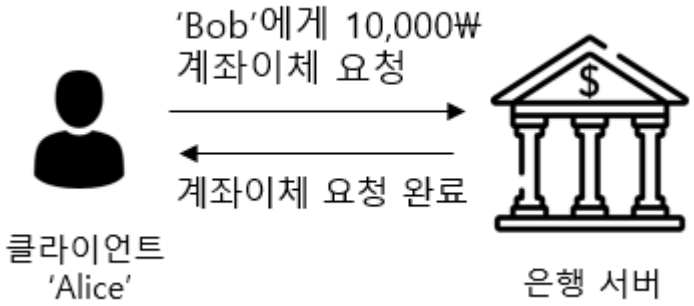
[배경 지식]

● **비트코인(Bitcoin) & 블록체인(Blockchain)이란?**

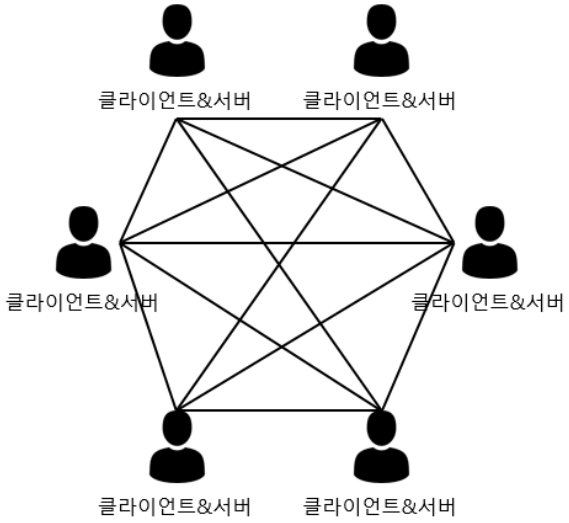
일반적으로 컴퓨터를 통한 서비스는 "클라이언트-서버" 구조를 가집니다. 예를 들면, 다음 그림과 같이 네이버 포털 사이트를 통해 "날씨에 대한 정보를 알려줘"라고 **요청**할 경우, 정보를 요청하는 사람은 클라이언트가 되며 요청된 정보에 **응답**해주는 네이버 포털 사이트는 서버가 됩니다.



비슷한 예시로 은행을 들 수 있습니다. 다음 그림은 계좌이체를 할 경우의 예시입니다.

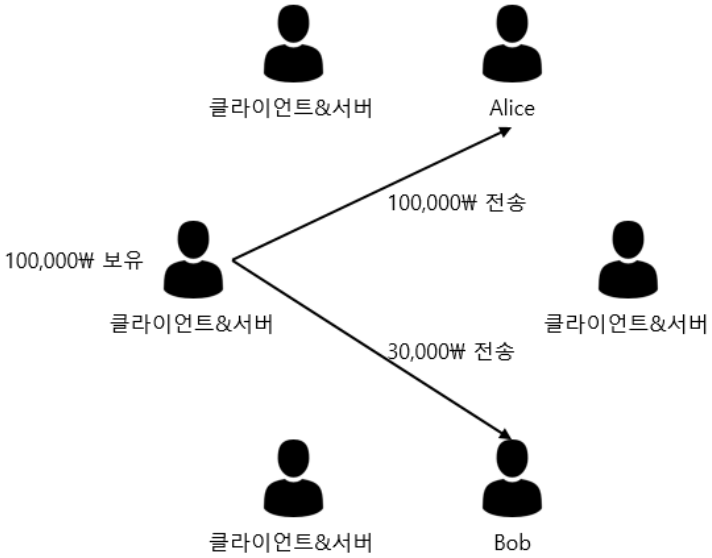


그러나 이러한 예시들과 반대로, 비트토렌트 혹은 P2P 파일 공유 시스템의 구조에서는 모든 사용자가 클라이언트이자 서버의 역할을 동시에 수행하게 됩니다. 자신이 필요한 것(파일 혹은 정보 등)을 다른 임의의 사용자에게 요청을 하며 요청을 받은 임의의 상대방은 그에 대한 응답을 해주는 것입니다. 또한 반대로 다른 사용자가 자신에게 정보를 요청해 올 경우, 마찬가지로 본인이 응답을 줄 수도 있습니다.



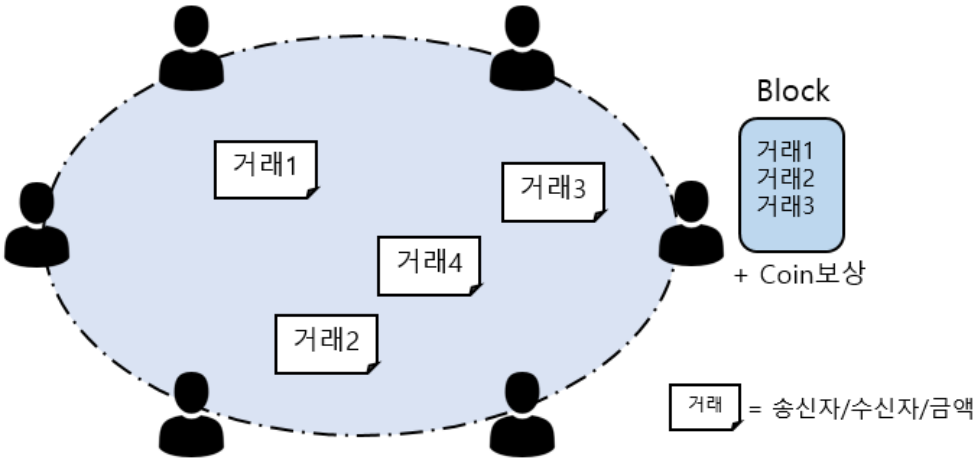
Blockchain은 비트토렌트와 같은 P2P(Peer-to-Peer) 형태의 분산 시스템입니다. Bitcoin은 P2P형태의 Blockchain 시스템에서 동작하는 소프트웨어이며, 즉 온라인 상에서 모든 사용자들이 직접 코인을 거래할 수 있는 것입니다. 그러나 은행과 같은 서버가 없음에 따라 한 가지 유의해야 할 점이 있습니다. 은행 서버를 통해 계좌이체를 할 경우를 예로 들어보겠습니다. 본인의 통장에 100,000₩이 있다고 가정할 때, Alice라는 사람에게 100,000₩을 계좌이체 하며 Bob이라는 사용자에게 30,000₩을 계좌이체 한다고 할 경우 은행 서버가 모두 중재 해주기 때문에 순차적으로 처리가 됩니다(이 경우 A에게 100,000₩을 먼저 전송하므로 금액이 0원이 되어서 B에게는 계좌이체 할 수 없게 됩니다).

그러나 P2P 시스템처럼 중재해주는 서버가 없다면 여러 개의 거래를 요청하는 경우 거래의 순서가 보장되지 않기 때문에 혼란을 겪게 됩니다.



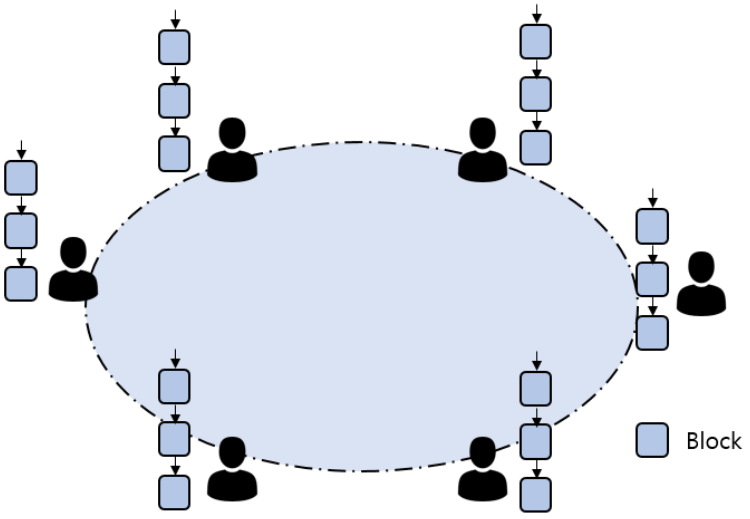
결국 모든 사용자들이 요청한 거래를 한 곳에 모아 순서를 정하고 사용자들끼리 해당 거래들이 옳다고 판단하는 "**합의**"를 해야하는 과정이 반드시 필요합니다. 이때 사용자들의 거래를 한 곳에 담아 놓은 것을 **Block**이라고 표현합니다. **동일한 사용자가 여러 거래를 요청했다고 하더라도 하나의 Block에는 그 거래들 중 하나만 포함될 수 있습니다**(위 그림의 문제를 막기

위해). 거래를 요청한 사용자가 동일하지 않을 경우 여러 거래들을 Block에 포함시킬 수 있습니다. 즉 Block 내에 거래들을 포함시키고 해당 Block을 모든 사용자들이 소유하도록 함으로써 **Block에 포함된 거래들만 옳다고 승인하게 됩니다(Block은 임의의 사용자가 랜덤으로 생성하게 되며 Block을 생성한 보상으로 Coin을 받게 됩니다).**



위의 그림에서 Block은 거래1, 2, 3을 포함하였습니다. 각 거래1, 2, 3은 Block에 포함되었으므로 송신자에서 수신자에게 금액이 전송됩니다. 그러나 거래4는 Block에 포함되지 않았으므로 송신자에서 수신자에게 금액이 전송될 수 없으며 다음 번 Block에 포함될 때까지 실행이 연기됩니다.

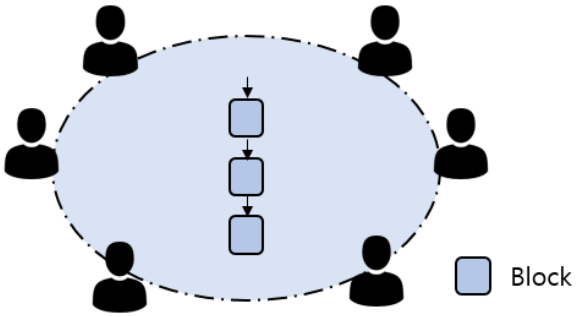
또한 모든 사용자는 이렇게 주기적으로 생성되는 Block들을 하나의 Chain 형태로 연결하여 모든 Block을 유지함으로써 모든 사용자들이 동일한(합의된) Block Chain을 가지고 있는지 확인할 수 있습니다.



BlockChain 기술의 대표적인 장점으로는, 기존의 클라이언트-서버 구조는 악의적인 공격자(해커)가 하나의 서버를 해킹하면 모든 사용자가 서버를 이용할 수 없게 됩니다. 반면 이러한 BlockChain 기술에서는 모든 사용자가 동일한 Block들의 Chain을 유지함으로써 한 명의 사용자가 해킹 공격을 당하더라도 나머지 모든 사용자들이 동일한 정보를 가지고 있기 때문에 해킹 공격의 위험을 대폭 감소시킬 수 있습니다.

[과제 설명 및 요구사항]

본 과제에서는 하나의 컴퓨터(프로그램)에서만 동작하므로 다음과 같이 모든 사용자들이 하나의 Block Chain을 동시에 유지하고 있다고 가정합니다.



Bitcoin 사용자를 여러 명 등록할 수 있으며, 여러 명의 사용자 중 하나의 사용자를 선택하여 등록한 사용자들 간에 거래를 진행 합니다. 또한, Block을 생성하는 사용자를 선택하여 Block을 만들고, 여러 개의 거래들 중 Block에 포함된 거래의 경우에만 금액이 반영되도록 합니다(+, -).

[주의 사항]

1. 본 프로그램은 11개의 명령어(help, useradd, userlist, getUser, chuser, mkGenBLK, mkTx, Txlist, mkBLK, blklist, quit)를 입력 받아 기능을 수행합니다. 각 명령어 별로 함수를 정의하여 사용해야 합니다. 명령어 외에 필요한 함수는 정의하여 사용할 수 있습니다.
2. 이번 과제는 여러 개의 파일로 분할하여 작성합니다.
   - **사용자 리스트(user.h, user.c)**: 이름(name), 보유금액(coin) 등의 정보를 가진 사용자를 구조체로 정의하고 링크드 리스트로 구현합니다. 사용자 정보와 관련된 함수들(사용자 등록 등)이 선언 및 정의 되어야 합니다.
   - **거래 리스트(trans.h, trans.c)**: Transaction을 구조체로 정의하고 거래 리스트를 링크드 리스트로 구현합니다. 거래와 관련된 함수 선언 및 정의를 합니다.
   - **블록 리스트(block.h, block.c):** Block을 구조체로 정의하고 Block 리스트는 링크드 리스트로 구현합니다. 블록 생성과 관련된 함수 선언 및 정의를 합니다.
   - **assn5.c**
     - main() 함수를 포함하여 필요한 함수들 정의
     - main() 함수 내에서는 사용자로부터 명령어를 입력 받아 처리합니다.
3. 이번 과제는 구조체와 연결리스트를 활용하는 것이 목표이므로, 문제에 구조체와 연결리스트를 언급한 부분을 배열을 통해서 해결할 경우 감점 처리합니다.
4. 명시한 에러 처리 외에는 고려하지 않아도 됩니다.
5. 과제 작성시 전역 변수, goto 문을 사용할 수 없습니다.
6. 프로그램 구현 시, main() 함수를 호출하여 사용하지 않습니다. 즉, 소스 코드 내에 main(); 이라고 호출하지 않습니다.
7. 문제의 출력 형식은 채점을 위해 아래의 실행 예시와 최대한 비슷하게 작성해 주세요.

(1) 초기화면

- 프로그램을 실행하면 아래와 같이 사용 가능한 명령어 목록이 출력된 후, 그 아래에 명령어 입력을 대기하는 ">> "표시가 출력됩니다. 이 때 사용자는 명령어를 입력할 수 있습니다.

```
----------------------------
 ★ Mini Bitcoin Simulation ★
----------------------------
help: 명령어 목록 보기
useradd: 사용자 등록
userlist: 등록된 모든 사용자 보기
getUser: 현재 사용자 확인
chuser: 사용자 변경
mkGenBLK: Genesis Block 생성
mkTx: Transaction 생성
Txlist: Transaction 목록 보기
mkBLK: Block 생성
blklist: BlockChain(거래 내역) 확인
quit: 프로그램 종료
----------------------------

>>
```

- help, useradd, userlist, getUser, chuser, mkGenBLK, mkTx, Txlist, mkBLK, blklist, quit는 각 메뉴의 명령어이고, 해당 명령어를 입력한 경우에만 각 기능이 실행됩니다. 이 명령어는 사용자가 명령어 입력 시, 대소문자를 구분하지 않고 동일한 명령어의 기능을 수행하도록 작성합니다. 예를 들면 help, Help, helP, HELP는 동일한 동작을 수행합니다.
- 위의 11가지 명령어 이외의 잘못된 명령어 입력 시, 아래와 같이 "wrong command!"를 출력한 후, 다시 명령어를 입력 받을 준비를 합니다.
  (노란색으로 표시된 문자는 사용자 입력에 해당)

```
>> hello
wrong command!

>>
```

- help 명령어 입력 시, 위 초기 화면이 그대로 다시 출력됩니다. (참고로, 화면 지우기 기능을 사용하지 않습니다.)

(2) 사용자 등록: useradd

- useradd 입력 시, 아래의 화면과 같이 이름을 입력하라는 화면이 출력되고 사용자 입력을 받아 사용자 등록을 할 수 있습니다.
- 여러 사용자를 등록할 수 있으며 처음 등록한 사용자는 0 코인을 보유합니다.
- 사용자 이름은 영문 최대 20글자로 공백이 없다고 가정합니다.

```
>> useradd
Enter user name >> alice
User name : alice
Coin : 0
USER REGISTER SUCCESS

>>
```

(예외 처리)

- 이미 있는 사용자를 입력한 경우, "Already exist"를 출력하고 다시 사용자 입력(>>)을 받을 준비를 합니다.

(3) 등록된 모든 사용자 확인: userlist

- userlist 입력 시, 등록된 모든 사용자 목록이 아래의 화면과 같이 출력됩니다.
- 사용자 목록은 등록된 순(alice, bob, carol, dave 순으로 등록 함)으로 출력되며, 현재 모든 사용자의 보유액이 0 코인임을 볼 수 있습니다.

```
>> userlist
1. Name : alice
   Coin : 0
2. Name : bob
   Coin : 0
3. Name : carol
   Coin : 0
4. Name : dave
   Coin : 0

>>
```

(예외 처리)

- 등록된 사용자가 없는 경우, "There is no current user"를 출력하고 다시 사용자 입력(>>)을 받습니다.

(4) 현재 사용자 확인: getUser

- getUser 입력 시, 현재 거래를 위한 사용자가 아래의 화면과 같이 출력됩니다.
- 처음 거래자는 가장 먼저 등록된 사용자로 기본 설정을 합니다. 이 프로그램에 가장 먼저 등록된 alice의 정보를 볼 수 있습니다.

```
>> getUser
Current user name : alice
Amounts of coin : 0

>>
```

(예외처리)

- 등록된 사용자가 없는 경우, "There is no current user"를 출력하고 다시 사용자 입력(>>)을 받습니다.

(5) 사용자(거래자) 변경: chuser

- chuser 입력 시, 거래를 할 사용자를 아래와 같이 입력 받습니다.
- 아래의 경우, 등록한 사용자들 중 bob을 입력하여 현재 거래자가 bob으로 변경됐음을 알 수 있습니다.

```
>> chuser
Enter use name >> bob
USER CHANGE SUCCESS

>> getUser
Current user name : bob
Amounts of coin : 0

>>
```

(예외처리)

- 등록되지 않은 사용자를 입력한 경우, "There is no user"를 출력하고 다시 사용자 입력(>>)을 받습니다.

(6) Genesis Block 생성: mkGenBLK

- 현재는 등록된 사용자들 모두 보유한 코인이 없으므로 거래를 할 수 없습니다. 이때는 Genesis Block이라는 첫 Block을 생성할 수 있습니다.
- 아래의 예시처럼 mkGenBLK 입력 시, 현재 사용자(bob)가 Genesis Block을 생성합니다.

```
>> mkGenBLK
GENESUS BLOCK CREATION SUCCESS

>> blklist
======Genesis Block======
***Special Transaction***
Sender : Nothing
Receiver : bob
Transferred coin : 100

>>
```

- Genesis Block은 처음 시작된 블록 즉, 그 앞에 어떤 Block도 생성되지 않은 최초의 Block을 말합니다. Genesis Block은 아무런 거래도 포함하지 않고 Block을 생성한 사용자에게만 보상을 주게 됩니다(100Coin).
  Block 생성자에게 보상을 주는 거래를 Special Transaction이라고 합니다. Block 생성자는 이 거래를 통해서 보상금을 받게 됩니다. Genesis Block 생성자 역시 이 Special Transaction을 통해서 보상을 받게 됩니다.
  Special Transaction은 송신자는 존재하지 않으며, 수신자는 Block 생성자인 자신의 이름으로 생성됩니다.
  (Sender : nothing / Receiver : Block generator / Coin : 100)

(예외처리)

- 이미 Genesis Block이 생성됐는데 이 명령어를 입력한 경우, "Genesis block has been already created"를 출력하고 다시 사용자 입력(>>)을 받습니다.

(7) BlockChain(거래 내역) 확인: blklist

- blklist 입력 시, 현재까지의 거래 내역을 최신 Block부터 Genesis Block 순서대로 출력 합니다. (추가 예시는 (10) Block 생성과 (12) 실행 예시 설명 참조)
- 위의 예시((6) Genesis Block 생성 설명에서의 예시)는 하나의 Block이 생성되었고, Genesis Block은 Special Transaction만 포함함을 볼 수 있습니다.
- 현재 사용자인 bob이 genesis block을 생성하였음으로 보상을 받았습니다.
- 명령어 userlist를 입력 하여, 사용자 목록 중 bob이 100 코인을 받은 것을 확인 할 수 있습니다. (아래의 예시)

```
>> userlist
1. Name : alice
   Coin : 0
2. Name : bob
   Coin : 100
```

```
3. Name : carol
   Coin : 0
4. Name : dave
   Coin : 0

>>
```

(8) Transaction 생성: mkTx

- bob은 코인이 생겼으므로, 코인을 전송할 수 있습니다.
- Transaction 생성을 위하여 아래 예시처럼 명령어 mkTx를 입력 한 후, 수신자와 이체할 코인을 순서대로 입력합니다.
- bob이 alice에게 코인을 전송하는 Transaction이 생성되었습니다.

```
>> getUser
Current user name : bob
Amounts of coin : 100

>> mkTx
Receiver name: alice
Amounts of coin to transfer: 20
TRANSACTION CREATION SUCCESS

>>
```

(예외처리)

- 등록되지 않은 사용자를 수신자로 입력한 경우, "There is no user"를 출력하고 다시 사용자 입력(>>)을 받습니다.
- 보유 금액이 전송하려는 금액보다 적은 경우, "Your coin is not enough"를 출력하고 다시 사용자 입력(>>)을 받습니다.

(가정 사항)

- 자기 자신한테 전송하는 거래는 없다고 가정합니다. 즉, 이 입력에 대해서는 고려할 필요가 없습니다.

(9) Transaction 목록 확인: Txlist

- Txlist 입력 시, 아래의 예시와 같이 Transaction을 확인할 수 있습니다.
- 해당 Trsanction은 Block에 포함된 것이 아니므로 bob의 현재 보유 코인은 그대로 100이며, alice의 현재 보유 코인은 그대로 0임을 알 수 있습니다.

```
>> Txlist
Sender : bob
Receiver : alice
Transferred coin : 20

>> userlist
1. Name : alice
   Coin : 0
2. Name : bob
   Coin : 100
3. Name : carol
   Coin : 0
4. Name : dave
   Coin : 0

>>
```

(예외처리)

- Transaction이 하나도 없는 경우 "There is no transaction"을 출력하고 다시 사용자 입력(>>)을 받습니다.

(10) Block 생성: mkBLK

- 다음은 현재 사용자를 carol로 변경 하고, Block 생성을 위하여 명령어 mkBLK를 입력한 예시입니다.

```
>> chuser
Enter use name >> carol
USER CHANGE SUCCESS

>> mkBLK
BLOCK CREATION SUCCESS

>> userlist
1. Name : alice
   Coin : 20
2. Name : bob
   Coin : 80
3. Name : carol
   Coin : 100
4. Name : dave
   Coin : 0

>>
```

- Block이 성공적으로 생성되면, Block에 포함된 거래들을 진행하여 보유 코인에 반영합니다. 이 거래에는 Block을 생성한 사용자에게 주는 보상 거래인 Special Transaction 을 포함하여 진행됩니다.
- 앞에서 생성한 Transaction(Sender: bob, Receiver: alice, Transferred coin: 20)이 진행되어 alice는 bob에게 20코인을 성공적으로 이체를 했고, Block 생성자 carol은 보상으로 100코인을 받았음을 볼 수 있습니다.

- 모든 Block은 보상을 주는 Special Transaction 및 일반 거래(Transaction)를 가집니다. 단, Genesis Block은 단 하나의 Special Transaction만 포함합니다.
- Block을 생성 하려면 일반 거래(Transaction)가 반드시 1개는 포함되어야 하며, 하나의 Block에는 일반 거래가 최대 3개까지 포함될 수 있습니다. 거래들을 Block에 포함시킬 때는 발생한 거래 순서대로 포함해야 합니다.
- 일반 거래(Transaction)이지만 Block에 포함될 수 없는 경우가 있습니다.
  (이에 대한 예시 및 설명은 (14) 추가 실행 예시를 참조)

(예외처리)

- 블록이 하나도 없는 경우, "Create genesis block first"를 출력하고 다시 사용자 입력(>>)을 받습니다.
- 일반 거래(Transaction)가 하나도 없거나, Block에 포함될 수 있는 일반 거래가 하나도 없는 경우, "There is not enough Transaction"을 출력하고 다시 사용자 입력(>>)을 받습니다.

(11) Block 목록 확인: blklist

- 현재 BlockChain을 확인하기 위하여, 아래처럼 명령어 blklist를 입력하면 하나의 Genesis Block과 하나의 Block이 존재함을 볼 수 있습니다.

```
>> blklist
==========Block==========
***Special Transaction***
Sender : Nothing
Receiver : carol
Transferred coin : 100

***Transaction***
Sender : bob
Receiver : alice
Transferred coin : 20


======Genesis Block======
***Special Transaction***
Sender : Nothing
Receiver : bob
Transferred coin : 100

>>
```

(12) 실행 예시

- 이제 많은 유저들이 코인을 보유하게 되었으므로 서로서로 거래를 발생할 수 있게 되었습니다.
- 다음 예시는 여러 사용자들이 다양하게 거래를 요청하였고 현재 Block에 포함되지 않은 Transaction 목록(등록순으로 출력) 입니다. (Block에 포함된 이전에 존재하던 Transaction은 이미 처리된 상태이므로 제외됩니다.)

```
>> Txlist
Sender : carol
Receiver : bob
Transferred coin : 15

Sender : carol
Receiver : alice
Transferred coin : 25

Sender : bob
Receiver : dave
Transferred coin : 40

Sender : alice
Receiver : dave
Transferred coin : 10

>>
```

- 거래는 생성되었지만 아직 block에 포함되지 않았으므로 User들의 코인 보유 현황은 다음과 같이 그대로입니다.

```
>> userlist
1. Name : alice
   Coin : 20
```

```
2. Name : bob
   Coin : 80
3. Name : carol
   Coin : 100
4. Name : dave
   Coin : 0

>>
```

- 아래는 alice가 Block을 생성한 경우로 User들의 코인 보유 현황은 다음과 같이 변경됩니다.

```
>> getUser
Current user name : alice
Amounts of coin : 20

>> mkBLK
BLOCK CREATION SUCCESS

>> Txlist
Sender : carol
Receiver : alice
Transferred coin : 25

>> userlist
1. Name : alice
   Coin : 110
2. Name : bob
   Coin : 55
3. Name : carol
   Coin : 85
4. Name : dave
   Coin : 50

>>
```

- Transaction 목록을 확인해보면, carol이 bob에게 15코인, alice에게 25코인을 전송하는 두 개의 거래를 요청했습니다. 하나의 Block에는 동일한 송신자의 Transaction의 경우 하나만 포함될 수 있으므로, 먼저 요청된 거래가 포함되고 carol이 bob에게 25코인을 전송하는 거래는 Block에 포함되지 못했습니다.
- 또한 하나의 Block은 최소 1개, 최대 3개의 Transaction을 포함할 수 있으므로 다른 Transaction은 모두 포함하였습니다.

```
>> Txlist
Sender : carol
Receiver : alice
Transferred coin : 25

>>
```

- 현재까지의 모든 BlockChain을 확인해보면 다음과 같습니다.
- 출력 순서는 최신 Block부터 Genesis Block 순서로 출력됩니다. 어떠한 Block이 어떠한 Transaction들을 포함했는지 확인 할 수 있습니다.

```
>> blklist
==========Block==========
***Special Transaction***
```

```
Sender : Nothing
Receiver : alice
Transferred coin : 100

***Transaction***
Sender : carol
Receiver : bob
Transferred coin : 15

***Transaction***
Sender : bob
Receiver : dave
Transferred coin : 40

***Transaction***
Sender : alice
Receiver : dave
Transferred coin : 10


==========Block==========
***Special Transaction***
Sender : Nothing
Receiver : carol
Transferred coin : 100

***Transaction***
Sender : bob
Receiver : alice
Transferred coin : 20


======Genesis Block======
***Special Transaction***
Sender : Nothing
Receiver : bob
Transferred coin : 100

>> quit
```

(13) 종료: quit

- quit 입력 시, 프로그램을 종료합니다.
- 프로그램을 종료할 때, 동적 할당된 모든 memory를 반드시 free 시킨 후 종료합니다.
- 해당 명령어를 실행할 때, 3개의 파일을 생성합니다.(아래는 위의 실행 예시에서 quit 명령어를 입력하여 종료한 경우의 예시입니다.)
  현재 BlockChain을 아래와 같은 형식으로 파일 "BlockChain.txt"에 저장합니다.

```
Block
Sender Nothing
Receiver alice
Coin 100
Sender carol
Receiver bob
Coin 15
Sender bob
Receiver dave
Coin 40
Sender alice
```

```
Receiver dave
Coin 10
Block
Sender Nothing
Receiver carol
Coin 100
Sender bob
Receiver alice
Coin 20
Block
Sender Nothing
Receiver bob
Coin 100
```

Transaction.txt: 남은 거래 내역을 아래와 같이 저장합니다.

```
Transaction
Sender carol
Receiver alice
Coin 25
```

User.txt: 사용자 정보(이름, 보유금액)을 아래와 같은 형식으로 저장합니다.

```
User
alice 110
bob 55
carol 85
dave 50
```

(14) 추가 실행 예시

- 아래의 실행 예시를 보면, 등록된 모든 사용자의 목록과 보유 금액을 알 수 있습니다.
- 현재 사용자 Jenny가 Tom에게 80을, Anne에게 80을 전송하려는 2개의 일반 거래를 생성했습니다. (Transaction은 현재 보유 금액이 전송하려는 금액보다 많으면 생성될 수 있습니다.)

```
>> userlist
1. Name : Jenny
   Coin : 100
2. Name : Tom
   Coin : 0
3. Name : Anne
   Coin : 0

>> getUser
Current user name : Jenny
Amounts of coin : 100

>> mkTx
Receiver name: Tom
Amounts of coin to transfer: 80
TRANSACTION CREATION SUCCESS

>> mkTx
Receiver name: Anne
Amounts of coin to transfer: 80
TRANSACTION CREATION SUCCESS

>>
```

- 아래의 예시에서, Transaction 목록을 확인할 수 있습니다. 사용자를 Tom으로 변경 후, Block을 생성했으며, 거래 진행 후 사용자 보유 금액을 확인할 수 있습니다.

```
>> Txlist
Sender : Jenny
Receiver : Tom
Transferred coin : 80

Sender : Jenny
Receiver : Anne
Transferred coin : 80

>> chuser
Enter use name >> Tom
USER CHANGE SUCCESS

>> mkBLK
BLOCK CREATION SUCCESS

>> userlist
1. Name : Jenny
   Coin : 20
2. Name : Tom
   Coin : 180
3. Name : Anne
   Coin : 0

>>
```

- 아래의 예시에서 남은 일반 거래를 확인할 수 있습니다. 현재 사용자 Tom이 다시 Block을 생성하기 위해 명령어(mkBLK)를 입력합니다. 이 경우 일반 거래가 존재하지만, 블록에 포함될 수 있는 일반 거래가 없으므로 블록을 생성할 수 없음을 볼 수 있습니다. Jenny의 현재 보유 금액은 20으로, Anne에게 80을 전송하려는 거래는 실행할 수 없으므로 Block에 포함될 수 없습니다.

```
>> Txlist
Sender : Jenny
Receiver : Anne
Transferred coin : 80

>> mkBLK
There is not enough Transaction

>> Txlist
Sender : Jenny
Receiver : Anne
Transferred coin : 80

>>
```

[헤더 파일 작성]

- 헤더 파일 작성 예시 (1) (user.h)

```
#ifndef USER_H
#define USER_H

// 구조체 정의
…
// 함수 선언
…

#endif
```

- 헤더 파일 작성 예시 (2) (user.h)

```
#pragma once
// 구조체 정의
…
// 함수 선언
…
```

위의 사용자 정의 헤더 파일 user.h를 필요한 곳에서 include 하여 사용합니다.

헤더 파일 작성시 위와 같이 작성을 하는 이유에 대해서 간략하게 조사하여 보고서에 서술합니다.(과제 점수에 포함됩니다.)