Universidad de Buenos Aires

Facultad de Ingeniería

Algoritmos y programación II (95.12)

Trabajo práctico n.°1

Grupo A

Nombre y Apellido	**Padrón**	**Correo Electrónico**
Aguirre Pedro	97603	paguirre@fi.uba.ar
Fernández Irungaray Martina	99611	mafernandez@fi.uba.ar
Tibaudin Guido	100136	gtibaudin@fi.uba.ar

1° entrega: 3 de diciembre de 2020

Índice

1. Introducción

El presente trabajo práctico tiene como objetivo realizar la implementación de una versión simplificada de la Blockchain denominada `Algochain`. Para ello, se utilizó gran parte del código correspondiente al Trabajo Práctico 0, realizando distintos tipos de correcciones y nuevas implementaciones tanto de funciones como de clases con sus respectivos métodos. También, fue necesario incorporar nuevos métodos de validaciones para una correcta implementación de una `Algochain`.

2. Desarrollo

2.1. Diseño

El código implementado consiste en un programa que, a partir de la línea de comandos, interactúa con el usuario permitiendo que éste realice distintas acciones como consultar el saldo de un usuario, crear transferencias, minar bloques para añadir a la Algochain, cargar Algochains ya existentes, etcétera. Las posibles opciones que el usuario puede ingresar son:

- **init <user><value><bits>**: Genera un bloque génesis para inicializar la Algochain. El bloque asignará un monto inicial `value` a la dirección del usuario `user`. El bloque deberá minarse con la dificultad `bits` indicada. Retorna el hash del bloque génesis.
- **transfer <src><dst1><value1>... <dstN><valueN>**: Genera una nueva transacción en la que el usuario `src` transferirá fondos a una cantidad `N` de usuarios, a cada uno su correspondiente `value`. Retorna el Hash de la transacción en caso de éxito y `FAIL` en caso de falla por invalidez.
- **mine <bits>**: Ensambla y agrega a la Algochain un nuevo bloque a partir de todas las transacciones en la mempool. El minado se efectúa con la dificultad dada por `bits`. Devuelve el hash del bloque en caso de éxito y `FAIL` en caso de falla.
- **balance <user>** Retorna el saldo disponible en la dirección del usuario `user`.
- **block <id>**:Consulta la información del bloque representado por el hash id. Retorna los campos del bloque en caso de éxito y `FAIL` en caso de recibir un hash inválido.
- **txn <id>**:Consulta la información de la transacción representada por el hash id. Retorna los campos de la transacción en caso de éxito y `FAIL` en caso de recibir un hash inválido.
- **load <filename>**: Carga la Algochain presente en el archivo. En caso de éxito devuelve el hash del ultimo bloque y `FAIL` en caso de falla.
- **save <filename>**: Guarda una copia de la Algochain en su estado actual al archivo indicado. Retorna `OK` en caso de éxito y `FAIL` en caso de falla.

Se optó por realizar un análisis detallado de los nuevos requerimientos de la implementación para poder establecer el flujo del programa. A partir de allí, se decidió modificar y complementar los métodos de las clases `List` y `Array`, crear una nueva clase denominada `Unspent`, implementar nuevas funciones en el módulo `Utility` e incluir un nuevo módulo llamado `Option`. Para el manejo de la `mempool` se decidió utilizar un arreglo de transacciones, ya que de esta manera resulta sencillo manipularla, ya que se deberá ir llenando la `mempool` con nuevas transacciones o vaciarla en ciertos casos.

A continuación, se describen brevemente algunos de los cambios desarrollados:

Template List: Se añadieron funciones de búsqueda a partir de determinados parámetros como el `txn_hash` y el hash del bloque. También se agregaron métodos que permiten eliminar todos los nodos de la lista, eliminar el primer nodo y setters y getters del primer y último nodo.

- `findbyblockhash(string blockid)`: este método itera sobre una lista de bloques completa, buscando un bloque que tenga un hash que coincida con `blockid`. Si lo encuentra, retorna el puntero al nodo que contiene dicho bloque. En caso de no encontrar el bloque, retorna `NULL`.

- `findbytxnshash(string txnid)`: este método itera sobre una lista completa de bloques, buscando una transacción en alguno de los bloques que tenga un hash que coincida con `txnid`. Si la encuentra, retorna esa transacción pasada a string (utilizando el método `toString()`), y en caso de que no la encuentre, retorna `FAIL`.

Template Array Se implementó sobre la clase `Array` el método `restartArray` específicamente para el manejo de la `mempool`, ya que era necesario vaciarla en repetidas ocasiones. Este método se llama sobre el `Array` (`mempool`) y lo vacía, es decir, lo reemplaza con un `Array` vacío (empleado mediante el constructor de la clase).

Para comentar sobre los demás objetos del programa, conviene tener presente el siguiente esquema:

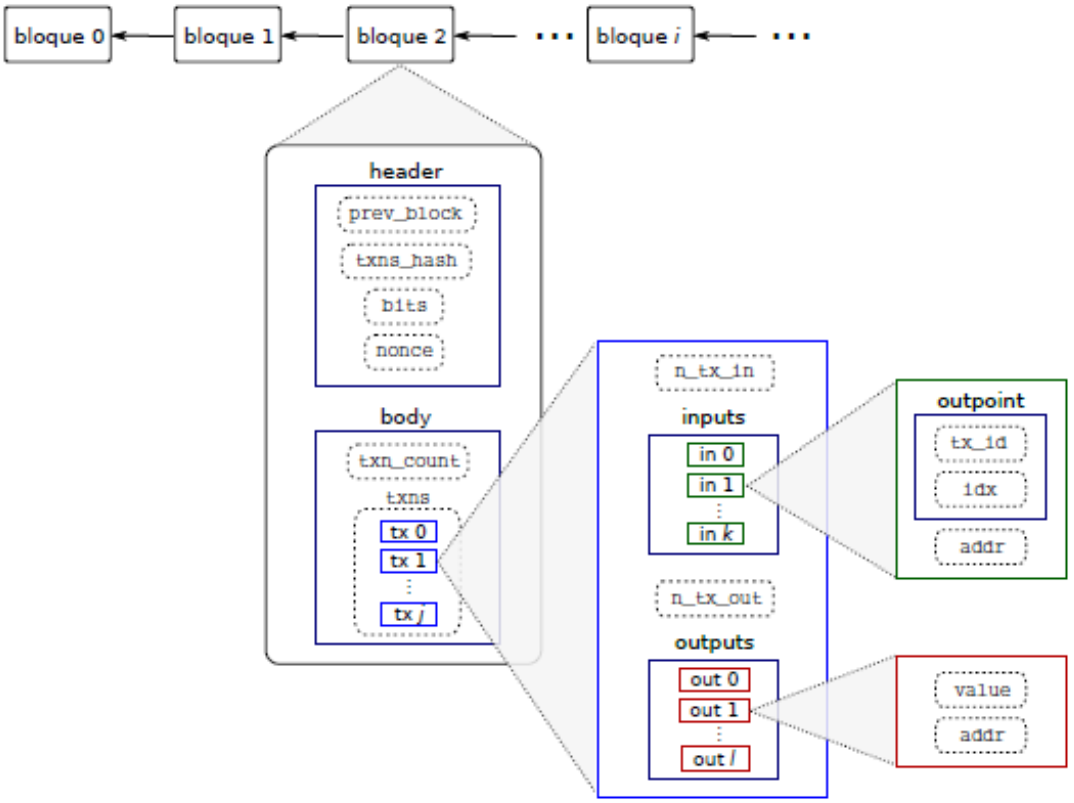

Figura 1: Esquema de alto nivel de la *Algochain*.

Class Unspent

Teniendo en cuenta los objetivos del trabajo práctico, se debió implementar una nueva clase, la clase `Unspent`. A continuación se detallan sus métodos y propiedades: Los objetos de esta clase tendrán un `Outpoint` y un `Value` y representarán cada `Output` disponible para gastar de un usuario. Esta implementación resultó de carácter fundamental para la realización de las funciones `optTransfer` y `optBalance`.

La clase se implementó con los correspondientes métodos `Getters` y `Setters`, sobrecargas de operadores, constructores y destructores.

2.2. Flujo del programa

Al ejecutar el programa se pueden establecer dos tipos de ingreso/salida de datos. El comando `-i` indica que la entrada de los datos será a través de un archivo `.txt`, el comando `-o` indica que la información de salida almacenada luego de realizar las operaciones, se imprimirá en el archivo `.txt`. En el caso en el cual alguno o ambos de los comandos se omitan, el ingreso y salida de información se realizará por defecto a través de la consola.

El programa comienza recibiendo una opción introducida por el usuario. Luego se valida el formato de los parámetros ingresados y se invoca la función correspondiente al comando. Dichas funciones retornarán una cadena de caracteres que se guardará en un vector. Si el stream de salida definido es `cout`, se irá imprimiendo por consola a medida que se invocan las funciones. En el caso en el cual se haya establecido salida por archivo `.txt`, se copiará el contenido del vector en el archivo de salida.

El programa recibe comandos hasta que el usuario presiona `CTRL + D`.

2.3. Opciones

Para determinar las opciones se optó por crear un `<map>`. Una vez que el usuario ingresa por la terminal el comando que desea ejecutar, se invoca la función `callOptions` que será la encargada, a través de un `switch`, de llamar a la función asociada al comando. La implementación de todas estas funciones se realizó en el archivo `option.cpp`.

A continuación, se detalla brevemente el desempeño de cada función.

2.3.1. optInit

La función `optInit` recibe un puntero a la dirección de la Algochain y un arreglo de strings llamado `cmd_arr` donde se encontrará el nombre de la opción (`init`) y los parámetros `<user><value><bits>` ingresados por el usuario. Luego de validar el formato de los datos, la función se encargará de generar y minar, con la dificultad indicada, el primer bloque de la lista, llamado 'Bloque Génesis'. En el caso en el cual la Algochain no esté vacía, la función eliminará todos los nodos pertenecientes a la lista y asignara el Bloque Génesis al comienzo de la lista.

2.3.2. optTransfer

La función `optTransfer` recibe un arreglo de strings que contiene el nombre de la opción, el nombre del usuario que va a realizar la transferencia, y por último una serie de pares `Usuario(Destino)/Valor` (destinatarios de la transacción). Esta función, con la ayuda de la función `totalUnspent`, se encarga de verificar que el usuario de origen de la transferencia tenga los fondos necesarios para realizar la misma. Luego de verificar esto, procede a crear la transacción pertinente y agregarla a la `mempool`. De esta manera se pueden ir acumulando transacciones en la `mempool`, y una vez que se llame a la función `mine`, estas transacciones se minarán y se generará un nuevo bloque en la `Algochain`. Si la transferencia ingresada resulta válida, se retorna el hash de la misma, y si no, se retorna `FAIL`.

2.3.3. optMine

La función `optMine` recibe por parámetros un arreglo de strings conteniendo el nombre de la opción y el valor de la dificultad con la que se realizará el minado. Recibe, también, un puntero a la lista Algochain y otro al arreglo de transacciones de la mempool. Crea el header y body de un bloque a partir de las transacciones presentes en la mempool. Luego, por medio del método `setMerkleHash()` se calcula la raíz de Merkle y se asigna al campo `txns_hash` para luego validar la dificultad. Finalmente se libera la mempool, agrega el bloque a la Algochain y se devuelve el hash del bloque.

2.3.4. optBlock

La función `optBlock` recibe por parámetros un arreglo de strings conteniendo el nombre de la opción y el hash de un bloque y un puntero a la Algochain. Luego itera sobre la lista por medio de la función `findByBlockHash` y retorna la información contenida en el nodo de la lista donde encontró el hash.

2.3.5. optBalance

La función `optBalance` recibe por parámetros un puntero a la lista Algochain, un puntero a un arreglo de transacciones `mempool` y un arreglo de strings `cmd_arr` conteniendo el nombre de la opción y nombre del usuario del cual se desea conocer el saldo `<user>`. Luego, a partir de la función `userUnspent`, se crea una lista de estructuras Unspent de todos los usuarios presentes en la Algochain. Finalmente, la función `totalUnspent` retorna el saldo disponible del usuario, contemplando las transacciones que se encuentran en la Algochain y las que están en la mempool.

2.3.6. optTxn

La función `optTxn` recibe un arreglo de strings `cmd_arr` conteniendo el hash id de una transacción y un puntero a la Algochain. Realiza las validaciones correspondientes y luego, a través del método

`findByBlockHash`, itera sobre la lista hasta encontrar el nodo correspondiente y devuelve la transacción pasada a string.

2.3.7. optSave

La función `optSave` recibe un arreglo de strings `cmd_arr` conteniendo el nombre de la opción y un puntero a la Algochain. Luego realiza las validaciones pertinentes y procede a generar un arreglo donde cada posición contiene un bloque de la Algochain, para luego imprimirlo en el archivo.

2.3.8. optLoad

La función `optLoad` recibe el arreglo `cmd_arr` con el nombre de la opción y el archivo del cual se leerá la nueva `Algochain`. Además recibe la lista `Algochain`, ya que si el `.txt` recibido es válido, se debe reemplazar la `Algochain` anterior por la nueva. Por último, recibe también la `mempool`, ya que ésta debe ser reiniciada si se reemplaza la `Algochain` por una nueva. La función `optLoad` se apoya en la función `validateLoadedAlgochain` para validar por completo la `Algochain` recibida. Esta función itera sobre todos los bloques de la `Algochain` (obtenida del `.txt`) y realiza todas las validaciones pertinentes sobre una `Algochain` para que ésta se considere válida. `validateLoadedAlgochain` se apoya a su vez en las funciones `validateDoubleSpending`, `validateGenesis` y `validateTxnFunds` (ver funcionamiento de las funciones en la sección 5, de código, estas funciones se encuentran en el archivo `validation.cpp`).

En estas funciones se valida lo siguiente sobre la `Algochain` recibida:

- Validación de `Double-spending` (Ningún `Output` puede estar referenciado por más de un `Input`).
- El primer bloque de la cadena debe ser un bloque génesis.
- El `Address` de cada `Output` referenciado por un `Input` debe coincidir con el `Address` del `Input`.
- Todas las transacciones deben tener la misma cantidad de fondos de entrada que de salida.
- Todos los `Inputs` deben referenciar a un `Output` existente.
- Todos los bloques deben cumplir con la dificultad definida (el hash criptográfico de su `Header` tiene que tener una cantidad de bits significativos en 0 como el número de dificultad).
- El `prev_block` de cada bloque tiene que coincidir con el hash del bloque anterior.

Como ya se mencionó, si la `Algochain` cumple todas las validaciones, esta reemplaza a la actual y se retorna el hash del último bloque de la nueva `Algochain`, de lo contrario, se retorna `FAIL` y se continua con la `Algochain` anterior.

3. Árbol de Merkle.

Un árbol de Merkle es una estructura de datos dividida en varios niveles que tiene como finalidad relacionar cada nodo con una raíz única asociada a los mismos. Para lograr esto, cada nodo está identificado con un hash único. Este par de nodos iniciales se asocian luego con un nodo superior llamado nodo padre, quien tendrá un hash resultante de aplicar el `sha256` a la concatenación de los hashes de sus hijos. Esta estructura se repite hasta llegar al nodo raíz o raíz Merkle.

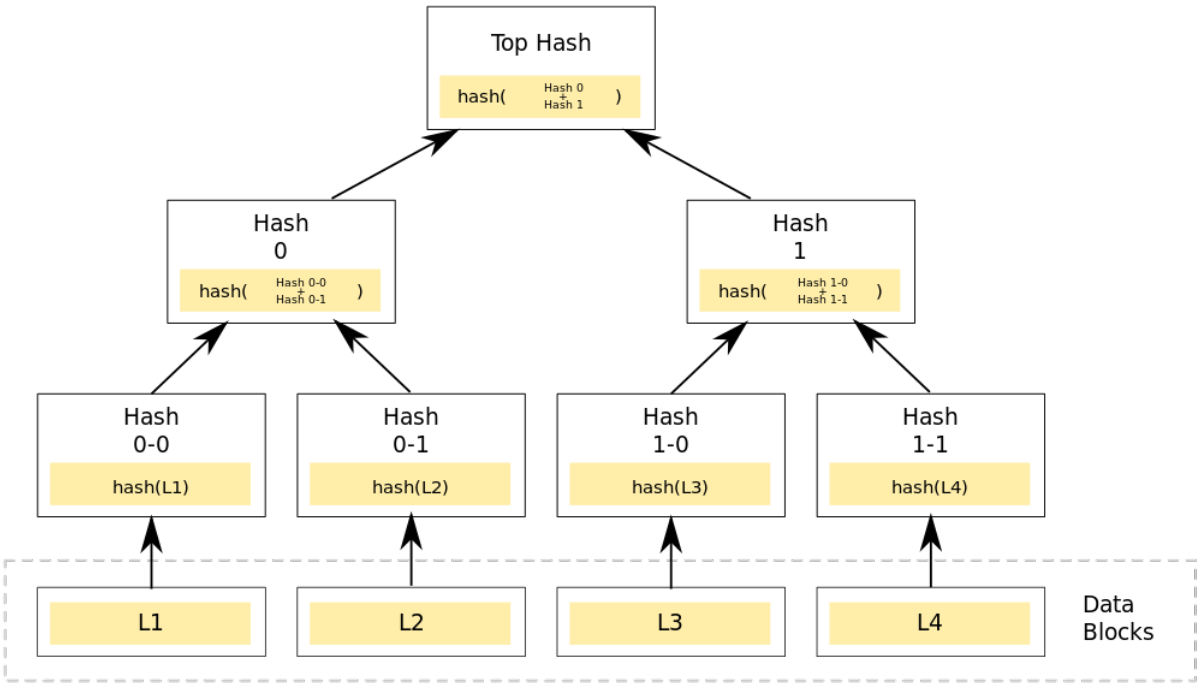

Figura 2: Estructura de un Árbol de Merkle

En este trabajo práctico se utilizó el Árbol de Merkle para asignar el valor de la raíz de Merkle, construido a partir de las transacciones de un bloque, al campo `txns_hash` perteneciente al header del bloque.

Para esta función se decidió implementar un algoritmo recursivo. La función `merkleRoot` recibe como parámetro un vector donde cada posición guardará el hash criptográfico de las transacciones involucradas en el bloque. A partir de allí, si la cantidad de elementos es par, se calcula el doble `sha256` de la concatenación de dos hashes en posiciones consecutivas y se guarda en un nuevo vector llamado `new_hash_arr`. En el caso en el cual la cantidad de elementos es impar, se procederá de la misma manera pero el último elemento a guardar en `new_hash_arr` será el doble `sha256` de la concatenación del último hash consigo mismo.

Luego se invoca la función utilizando como parámetro de entrada el nuevo vector `new_hash_arr`. Esto se repite hasta alcanzar la condición de corte del algoritmo, definida para el caso en el cual el vector de entrada tiene un tamaño igual a 1.

El código correspondiente se muestra a continuación.

```
1
2  string merkelRoot(Array<string> hash_arr)
3  {
4    size_t i;
5    if(hash_arr.getSize()==1)
6      return hash_arr[0]; //Condición de corte
7    Array<string> new_hash_arr(0);
8    if (hash_arr.getSize()%2 ==0)
9    {
10     for(i=0;i<hash_arr.getSize(); i+=2){
11       new_hash_arr.addValueEnd(sha256(sha256(hash_arr[i]+hash_arr[i+1])));
12     }
13   }
14   else
15   {
16     for(i=0;i<hash_arr.getSize()-1; i+=2){
17       new_hash_arr.addValueEnd(sha256(sha256(hash_arr[i]+hash_arr[i+1])));
18     }
19     new_hash_arr.addValueEnd(sha256(sha256(hash_arr[i]+hash_arr[i])));
20   }
21   return hash_value(new_hash_arr);//Llamada recursiva pasando por parametro un arreglo de
       menor tamñano al recibido.
22 }
```

4. Proceso de compilación

La compilación de los archivos que conforman el programa se realiza a través de un archivo `Makefile`. Allí se indica que deben ser compilados todos aquellos archivos cuya extensión sea `.cpp` y se invocan los *flags* `-Wall -pedantic` para la detección de errores y *warnings*.

Luego, el programa es compilado a partir del comando `make` e invocado por medio de cualquiera de las siguientes líneas:

- `./TP1`
- `./TP1 -i` *archivo_entrada* `-o` *archivo_salida*
- `./TP1 -i` *archivo_entrada*
- `./TP1 -o` *archivo_salida*

4.1. Ejecuciones de prueba

En la carpeta comprimida se adjuntó una carpeta "Tests" con una serie de pruebas que sirven para mostrar la robustez del programa. Estas pruebas se realizan especialmente sobre la opción "Load" del programa, ya que es en esa función donde se hallan las validaciones más importantes (se debe comprobar la validez completa de una `Algochain` recibida en un `.txt`). Cabe aclarar que, además, se realizó una gran cantidad de pruebas de ejecución sobre el programa, como por ejemplo el ingreso incorrecto de comandos (ante cualquier incongruencia en el comando ingresado, el programa muestra un error y continúa esperando nuevos comandos), además muchas pruebas exhaustivas sobre el funcionamiento de las funciones "Transfer", "Balance", entre otras. Se decidió no adjuntar todas las pruebas realizadas, ya que harían demasiada extensa a esta sección. A continuación, se muestran capturas de los tests realizados sobre la opción "Load":

En primer lugar, se llevó a cabo una prueba ingresando una `Algochain` con un bloque génesis inválido (el primer bloque de la `Algochain` recibida debe ser siempre un bloque génesis). Cabe aclarar aquí que este bloque debe cumplir con lo siguiente:

- Debe tener un `prev_block` nulo.
- Debe tener un solo input (que debe ser también nulo).
- Debe tener un solo output (este output no tiene restricciones, puede ser de cualquier valor y a cualquier destinatario).

- Este bloque debe además cumplir las mismas condiciones que cualquier bloque de la **Algochain**, es decir, cumplir con la dificultad de minado y que el txn_hash coincida con el hash de Merkle de sus transacciones.

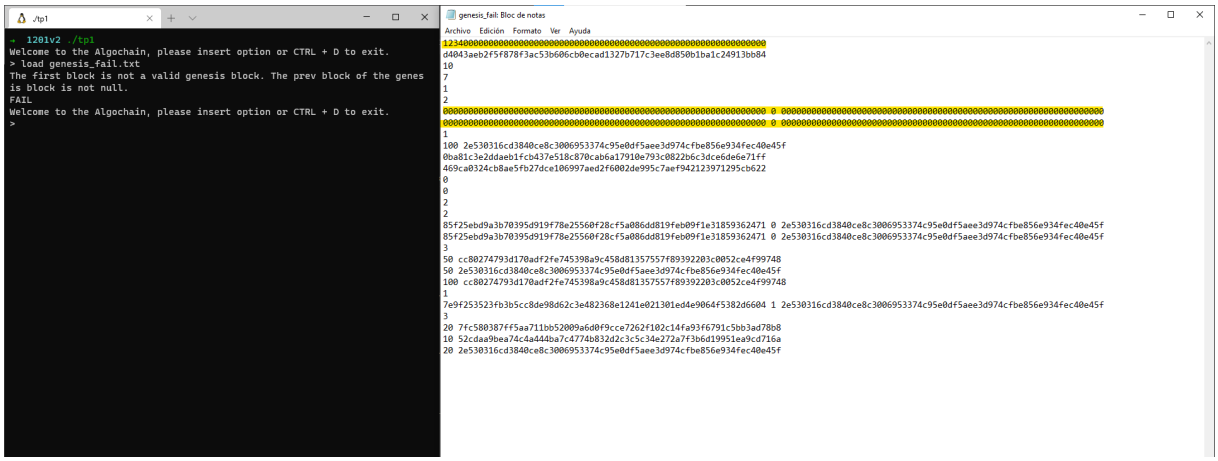

Figura 3: Ejecución del programa ingresando un bloque génesis inválido.

Se realizaron además, pruebas sobre el campo **prev_block** del **header** de cada bloque de la **Algochain**. Este campo debe contener el hash del bloque anterior, y si estos no coinciden, la **Algochain** se considera inválida y no es cargada. Esta validación se realiza sobre todos los bloques de la**Algochain** recibida.

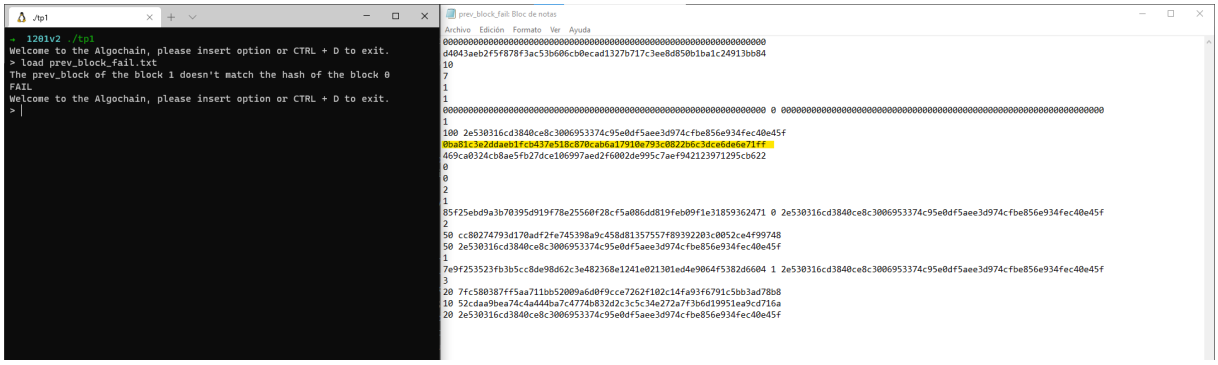

Figura 4: Ejecución del programa ingresando bloques con prev_block inválidos.

Además se debió validar que todos los bloque de la **Algochain** cumplan con la dificultad dada, es decir, que el hash criptográfico del **header** debe contener una cantidad de ceros a la izquierda igual al número “bits”. A continuación, se muestra un ejemplo de esta validación:

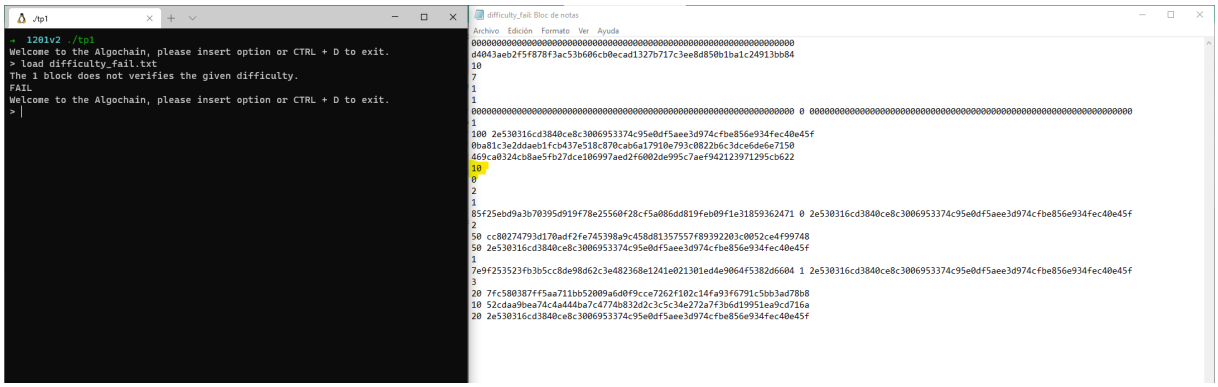

Figura 5: Ejecución del programa ingresando bloques que no cumplan la dificultad dada.

Otra validación muy importante que se realizó fue la de `Double-Spending`. Esta falla se da cuando algún `Output` de la `Algochain` es referenciado más de una vez (por más de un `Input`), ya que esto significaría que los fondos de dicho `Output` están siendo gastado más de una vez. A continuación, se muestra un caso de esta situación:

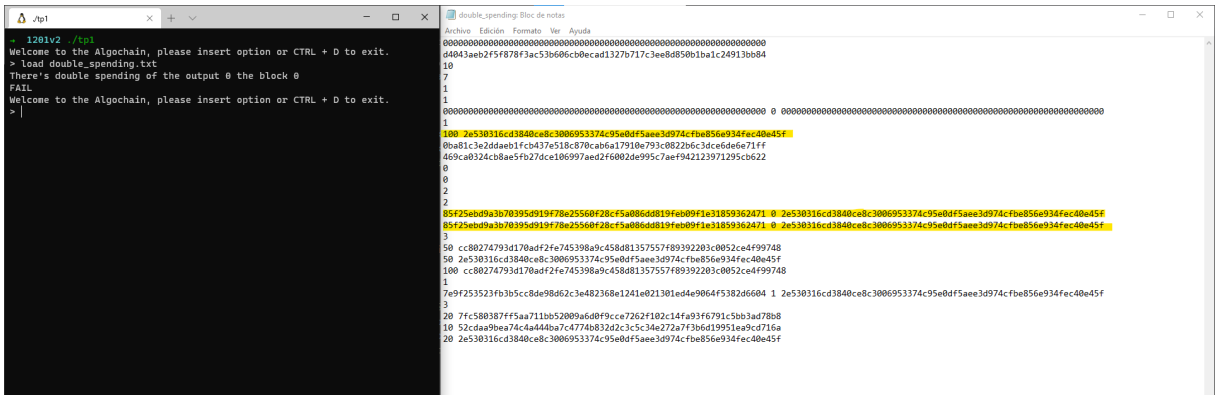

Figura 6: Ejecución del programa ingresando bloques que cometen double-spending.

También se debe verificar que para todas las referencias de fondos, es decir, cada vez que un `Input` referencia a un `Output`, ambos deben tener la misma `Addr` (pertenecer al mismo usuario), si esto no se cumple para cualquier caso, la `Algochain` se considera inválida y se deshecha. Se adjunta a continuación un ejemplo de dicho caso:

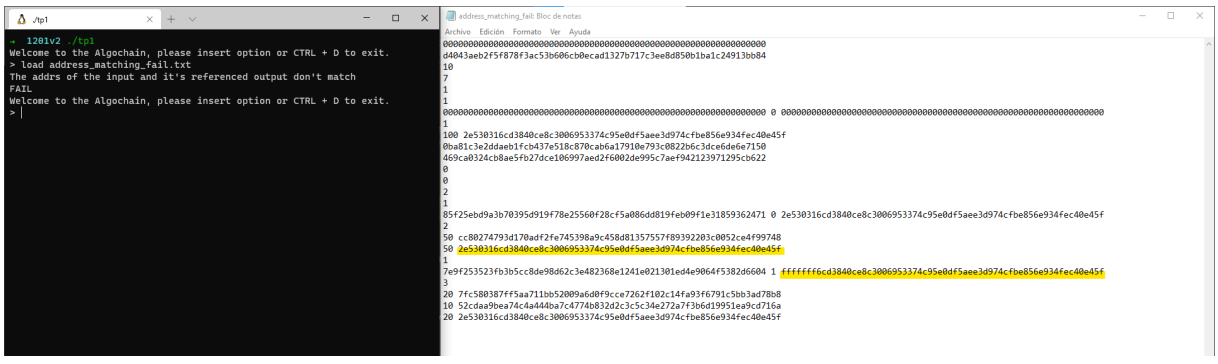

Figura 7: Ejecución del programa ingresando bloques con fallas en las coincidencias de `Address`

Se debe validar además, que todas las transacciones de todos los bloques (excepto del bloque génesis) tengan coincidencia en los fondos de origen y los fondos de destino, es decir, la suma de todos los `Values` de todos los `Outputs` referenciados en una transacción (cada `Input` referencia a un `Output`) debe ser igual a la suma de todos los `Values` de los `Outputs` de la misma transacción. En otras palabras, en cada transacción debe entrar la misma cantidad de `Algocoins` de las que salen.

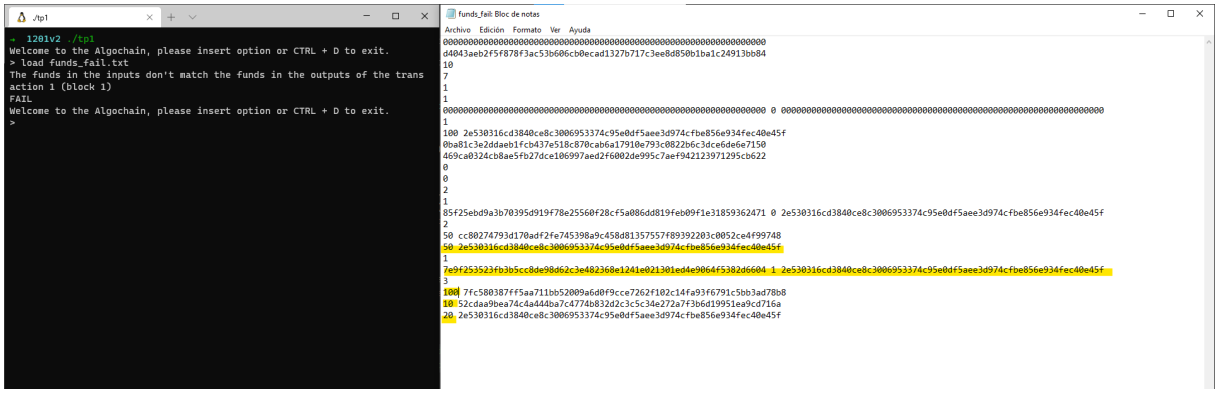

Figura 8: Ejecución del programa ingresando bloques con fondos incorrectos en alguna transacción.

4.2. Análisis de memoria

Se verifica por medio del programa `Valgrind` que no exista fuga de memoria en el programa. Los resultados de la ejecución se muestran a continuación:

```
valgrind --leak-check=full
./TP1 -i input_commands.txt -o output.txt

HEAP SUMMARY:
==291==     in use at exit: 0 bytes in 0 blocks
==291==   total heap usage: 230,965 allocs, 230,965 frees, 14,694,819 bytes allocated
==291==
==291== All heap blocks were freed -- no leaks are possible
==291==
==291== Use --track-origins=yes to see where uninitialised values come from
==291== For lists of detected and suppressed errors, rerun with: -s
==291== ERROR SUMMARY: 614 errors from 92 contexts (suppressed: 0 from 0)
```

4.3. Problemas

- Cabe destacar que si bien al finalizar el programa no se presentan fugas de memoria, al realizar la primera invocación de la función `optTransfer` y `optBalance`, `Valgrind` indica el warning `Uninitialised value was created by a stack allocation` lo cual significa que alguna función está recibiendo una variable sin inicializar. Se pudo rastrear dicha alerta hasta la función `sumUnspent`, en particular generado por la variable `List <Unspent>*`. Esta lista se encarga de guardar la información de los `Output` disponibles para ser utilizados en futuras transacciones (objetos de la clase `Unspent`). Al comienzo del programa esta lista se inicializa vacía ya que al no haberse realizado ninguna operación, no hay ningún usuario con saldo disponible. Sin embargo, si bien se inicializaron todos los constructores de las clases y se realizó un seguimiento de las variables, no se pudo determinar el origen de dicho warning.

- Además, se encontró una serie de problemas de implementación en distintos sectores del código, donde si bien la complejidad algorítmica no era tan grande, se encontraron complicaciones para llevar a cabo ciertas funciones, ya que requerían un alto nivel de abstracción. Un claro ejemplo de esto se vio en las funciones de validación (ver sección 5, archivo `validation.cpp`), donde había una gran cantidad de casos para validar, y las iteraciones se volvían muy largas y confusas. A pesar de estos obstáculos propios de la inexperiencia, se lograron llevar a cabo las validaciones requeridas, aunque probablemente existan maneras más eficientes.

5. Código fuente

5.1. main.cpp

```
#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <cstdlib>
#include <string>

#include "file.h"
#include "transaction.h"
#include "array.h"
#include "block.h"
#include "list.h"
#include "validation.h"
#include "cmdline.h"
#include "sha256.h"
#include "header.h"
#include "messages.h"
#include "utility.h"
#include "options.h"

using namespace std;

// A continuación, se definen las funciones para el manejo de opciones por línea de
    comandos.

static void opt_input(string const &);
static void opt_output(string const &);


static option_t options[] = {
  {1, "i", "input", "-", opt_input, OPT_DEFAULT},
  {1, "o", "output", "-", opt_output, OPT_DEFAULT},
  {0, },
};
static istream *iss = 0;
static ostream *oss = 0;
static fstream ifs;
static fstream ofs;


static void
opt_input(string const &arg){
  // Si el nombre del archivos es "-", usaremos la entrada
  // estándar. De lo contrario, abrimos un archivo en modo
  // de lectura.

  if (arg == "-") {
    iss = &cin;
  } else {
    ifs.open(arg.c_str(), ios::in);
    iss = &ifs;
  }

  // Verificamos que el stream este OK.

  if (!iss->good()) {
    cerr << "cannot open "
         << arg
         << "."
         << endl;
    exit(1);
  }
}

static void
```

```
65 opt_output(string const &arg)
66 {
67   // Si el nombre del archivos es "-", usaremos la salida
68   // estándar. De lo contrario, abrimos un archivo en modo
69   // de escritura.
70
71   if (arg == "-") {
72     oss = &cout;
73   } else {
74     ofs.open(arg.c_str(), ios::out);
75     oss = &ofs;
76   }
77
78   // Verificamos que el stream este OK.
79
80   if (!oss->good()) {
81     cerr << "cannot open "
82          << arg
83          << "."
84          << endl;
85     exit(1);
86   }
87 }
88
89
90
91 int main(int argc, char * const argv[]){
92   //  Definición de instancias a utilizar.
93   string option_response;
94   List <Block> algochain;
95   Array<Transaction> mempool(0);
96   Block block;
97   Array<string> arr_to_file(0);
98   Array<string> fvalues(0);
99   string user_input;
100  map<string,int> opt_map = optionsMap();
101
102  // Manejo de opciones por línea de comandos. Aquí se definen los flujos de entrada y
       salida que se utilizarán
103  // en la ejecución del programa.
104
105  cmdline cmdl(options);
106  cmdl.parse(argc, argv);
107
108  // Lectura del archivo de entrada (si es que se recibe) y asignación del contenido a un
       vector de
109  // strings (fvalues). readFile contempla el caso en el que no se reciba un archivo de
       entrada
110  // (se habilita la inserción de transacciones por medio del flujo de entrada cin).
111
112  size_t i;
113
114  if (iss==&cin){
115    while(!(*iss).eof()){
116      user_input = readCin();
117      if (optionValid(user_input)){
118        option_response = callOption(user_input, opt_map, &algochain, &mempool);
119        cout << option_response << endl;
120        arr_to_file.addValueEnd(option_response);
121      }
122      else if (user_input == "\0")
123      {}
124      else cerr << ERR_WRONG_CMD << endl;
125    }
126  }
127  else{
128    fvalues = readFile(ifs);
129    for(i = 0; i<fvalues.getSize(); i++){
130      if(optionValid(fvalues[i]))
```

```
131         arr_to_file.addValueEnd(callOption(fvalues[i], opt_map, &algochain, &mempool)+
    BREAK_LINE);
132       else cerr<<MSG_LINE_NUMBER<< i + 1 << ERR_INVALID_FILE_FORMAT<<endl;
133     }
134     if(oss==&cout)
135       cout << arr_to_file;
136   }
137   writeFile(ofs, arr_to_file);
138 
139   return 0;
140 
141 }
```

5.2. array.h

```
1  #ifndef ARRAY__H
2  #define ARRAY__H
3  
4  #include <iostream>
5  #include <cstdio>
6  #include <string>
7  #include <sstream>
8  #include "messages.h"
9  
10 #define INIT_SIZE 0
11 #define ARR_DEFAULT_SIZE 2
12 #define DELIM_TX ' '
13 
14 using namespace std;
15 
16 template <typename T>
17 class Array{
18     T * data;
19     size_t size = INIT_SIZE;
20     size_t alloc_size = ARR_DEFAULT_SIZE;
21 
22 public:
23 
24     Array(const Array<T> &);
25     Array(int n=ARR_DEFAULT_SIZE);
26     ~Array();
27 
28     size_t getSize()const;
29 
30     Array <T> & operator=(const Array <T> &);
31     bool operator==(const Array <T> &)const;
32     bool operator!=(const Array <T> &)const;
33     T & operator[](int i) const;
34     void addValueEnd(T info);
35     void restartArray();
36 
37     template <typename TT>
38     friend ostream& operator<<(ostream &os, Array<TT> const&arr);
39 };
40 
41 // Constructores de instancias de la clase Array.
42 
43 template <typename T>
44 Array<T>::Array(int n){
45     data = new T[n];
46     alloc_size = n;
47     size = INIT_SIZE;
48 }
49 
50 template <typename T>
51 Array<T>::Array(const Array<T> & arr){
52     size = arr.size;
53     data = new T[size];
54 
```

```
55      for (size_t i=0; i < size; i++)
56          data[i] = arr[i];
57  }
58
59  // Destructor.
60
61  template <typename T>
62  Array <T>::~Array(){
63      if(data != NULL)
64          delete []data;
65  }
66
67  // Métodos de instancia.
68
69  template <typename T>
70  size_t Array<T>::getSize() const{
71      return size;
72  }
73
74  // Sobrecarga de operadores.
75
76  template <typename T>
77  Array <T> & Array <T>::operator=(const Array <T> & r){
78      if(&r != this){
79          if (size != r.size){
80              T * aux;
81              aux = new T[r.size];
82              delete[] data;
83              size = r.size;
84              data = aux;
85          }
86          for (size_t i = 0; i < size; i++)
87              data[i] = r.data[i];
88      }
89      return *this;
90  }
91
92  template <typename T>
93  bool Array<T>::operator==(const Array & r)const{
94      size_t i;
95      if(size != r.size)
96          return false;
97      else
98          for(i=0; i<size; i++){
99              if(!(data[i] == r.data[i]))
100                 return false;
101         }
102     return true;
103 }
104
105 template <typename T>
106 bool Array<T>:: operator!=(const Array <T> &r) const{
107
108     return !(*this).operator==(r);
109
110 }
111
112 template <typename T>
113 T & Array <T>::operator[](int i) const{
114     if(i<0)
115     {
116         cerr<<ERR_INVALID_INDEX<<endl;
117         exit(1);
118     }
119     else
120         return data[i];
121 }
122
123 template <typename T>
```

```
124 ostream & operator<< (ostream &os, Array<T> const&arr){
125     size_t i;
126 
127     if(arr.size == 0)
128         return os;
129     for(i=0; i<arr.getSize() - 1; i++){
130         os << arr[i] ;
131     }
132     os << arr[i];
133 
134     return os;
135 }
136 
137 //  Método que recibe un dato u objeto y lo agrega al final del vector. Para esto, se
        solicita memoria para la cantidad de
138 //elementos que se almacenarán en el vector de modo de reubicar los valores en dichas
        direcciones y
139 //finalmente se libera la memoria utilizada previamente por el vector.
140 
141 template <typename T>
142 void Array<T>::addValueEnd(T info){
143     T * aux = new T[size+1];
144     for (size_t i=0; i<size; i++){
145         aux[i] = data[i];
146     }
147     size++;
148     delete[] data;
149     data = aux;
150 
151     data[size - 1] = info;
152 }
153 
154 // Método que reinicia un array, es decir, lo reemplaza por uno vacio.
155 
156 template <typename T>
157 void Array<T>::restartArray(){
158     Array <T> aux(0);
159   *this = aux;
160 }
161 
162 #endif
```

5.3. block.cpp

```
1  #include "block.h"
2  #include "messages.h"
3  #include <string>
4  #include "sha256.h"
5  
6  #define BREAK_LINE '\n'
7  
8  using namespace std;
9  
10 // Constructores
11 
12 Block::Block(){
13 }
14 
15 Block::Block(Header h,Body b){
16   header = h;
17   body = b;
18 }
19 
20 Block::Block(const Block &b){
21   header = b.header;
22   body = b.body;
23 }
24 
25 // Destructor.
```

```
26
27 Block::~Block(){
28 }
29
30 // Sobrecarga de operadores.
31
32 Block &Block::operator=(const Block &b){
33   header = b.header;
34   body = b.body;
35   return *this;
36 }
37
38 bool Block::operator==(const Block &b){
39   return (header == b.header && body == b.body);
40 }
41
42 ostream & operator<< (ostream &os, Block block){
43   os << block.header << endl;
44   os << block.body;
45   return os;
46 }
47
48 // Métodos de instancia.
49
50 Header Block::getHeader(){
51   return header;
52 }
53
54 Body Block::getBody(){
55   return body;
56 }
57
58 void Block::setHeader(Header h){
59   header = h;
60 }
61
62 void Block::setBody(Body b){
63   body = b;
64 }
65
66
67 void Block::setTxnsHash(){
68   Array<string> str_vec(0);
69   Array<Transaction> tx_arr = body.getTx_arr();
70
71   for (size_t i = 0; i < tx_arr.getSize(); i++){
72     str_vec.addValueEnd(tx_arr[i].toString());
73   }
74   header.setTxnsHash(str_vec);
75 }
76
77 string Block::toString(){
78   string block_str;
79   size_t i;
80
81   block_str += header.getPrev_block() + BREAK_LINE;
82   block_str += header.getTxns_hash() + BREAK_LINE;
83   block_str += to_string(header.getBits()) + BREAK_LINE;
84   block_str += to_string(header.getNonce()) + BREAK_LINE;
85
86   block_str += to_string(body.getTxn_count()) + BREAK_LINE;
87   for (i = 0; i < body.getTx_arr().getSize() - 1; i++){
88     block_str += body.getTx_arr()[i].toString() + BREAK_LINE;
89   }
90   block_str += body.getTx_arr()[i].toString();
91
92   return block_str;
93 }
94
```

```
string Block::getBlockHash(){
  return (sha256(sha256(toString() + BREAK_LINE)));
}


void Block::setMerkleHash(){
  Array<string> hash_arr(0);

  for (size_t i = 0; i < body.getTx_arr().getSize(); i++){
    hash_arr.addValueEnd(body.getTx_arr()[i].getTxHash());
  }

  if(hash_arr.getSize()==1){
    header.setTxnsHashManually(sha256(sha256(hash_arr[0]+hash_arr[0])));
    return;
  }

  header.setTxnsHashManually(merkleRoot(hash_arr));

}

string Block::getMerkleHash(){
  Array<string> hash_arr(0);

  for (size_t i = 0; i < body.getTx_arr().getSize(); i++){
    hash_arr.addValueEnd(body.getTx_arr()[i].getTxHash());
  }

  if(hash_arr.getSize()==1){
    return sha256(sha256(hash_arr[0]+hash_arr[0]));
  }

  return merkleRoot(hash_arr);
}


string merkleRoot(Array<string> hash_arr){
  size_t i;

  if(hash_arr.getSize()==1)
    return hash_arr[0];

  Array<string> new_hash_arr(0);

  if (hash_arr.getSize()%2 ==0){
    for(i=0;i<hash_arr.getSize(); i+=2){
      new_hash_arr.addValueEnd(sha256(sha256(hash_arr[i]+hash_arr[i+1])));
    }
  }
  else
  {
    for(i=0;i<hash_arr.getSize()-1; i+=2){
      new_hash_arr.addValueEnd(sha256(sha256(hash_arr[i]+hash_arr[i+1])));
    }
    new_hash_arr.addValueEnd(sha256(sha256(hash_arr[i]+hash_arr[i])));
  }
  return merkleRoot(new_hash_arr);
}
```

5.4. block.h

```
#ifndef _BLOCK_H_INCLUDED_
#define _BLOCK_H_INCLUDED_

#include <iostream>
#include <string>

#include "header.h"
#include "body.h"
```

```
#include "array.h"
#include "transaction.h"

#define DEFAULT_PREV_BLOCK "
    ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff"
#define DEFAULT_NONCE 0

using namespace std;

class Block{
    friend class Header;
    friend class Body;
    Header header;
    Body body;

public:
    Block();
    Block(Header h,Body b);
    Block(const Block &);
    ~Block();

    Block &operator=(const Block &);
    bool operator==(const Block &);

    Header getHeader();
    Body getBody();

    void setHeader(Header);
    void setBody(Body);
    void setTxnsHash();
    void setMerkleHash();
    string getBlockHash();
    string toString();
    string getMerkleHash();


    friend ostream & operator<< (ostream &, Block);
};
    string merkleRoot(Array<string> hash_arr);

#endif
```

5.5. body.cpp

```
#include <string>

#include "body.h"
#include "messages.h"

using namespace std;

// Constructores.

Body::Body(){
}

Body::Body(unsigned int n, Array<Transaction> arr){
    txn_count = n;
    tx_arr = arr;
}

Body::Body(const Body &bdy){
    txn_count = bdy.txn_count;
    tx_arr = bdy.tx_arr;
}

// Destructor

Body::~Body(){
```

```
    txn_count = 0;
    tx_arr = 0;
}

// Sobrecarga de operadores.

Body &Body::operator=(const Body &bdy){
    txn_count = bdy.txn_count;
    tx_arr = bdy.tx_arr;
    return *this;
}

bool Body::operator==(const Body &bdy){
    return (txn_count == bdy.txn_count && tx_arr == bdy.tx_arr);
}


ostream & operator<< (ostream &os, Body bdy){
    os << bdy.txn_count << endl;
    os << bdy.tx_arr << endl;
    return os;
}

// Métodos de instancia.

unsigned int Body:: getTxn_count(){
    return txn_count;
}

Array<Transaction> Body:: getTx_arr(){
    return tx_arr;
}

void Body:: setTxn_count(unsigned int n){
    txn_count = n;
}
void Body:: setTx_arr(Array<Transaction> arr){
    tx_arr = arr;
}
```

5.6. body.h

```
#ifndef _BODY_H_INCLUDED_
#define _BODY_H_INCLUDED_

#include <iostream>
#include "array.h"
#include "transaction.h"

using namespace std;

class Body{
    unsigned int txn_count;
    Array<Transaction> tx_arr;

public:

    Body();
    Body(unsigned int, Array<Transaction>);
    Body(const Body &);
  ~Body();

    Body &operator=(const Body &);
    bool operator==(const Body &);

    unsigned int getTxn_count();
    Array<Transaction> getTx_arr();

    void setTxn_count(unsigned int);
```

```
    void setTx_arr(Array<Transaction>);

  friend ostream & operator<< (ostream &os, Body bdy);
};
#endif
```

5.7. cmdline.cpp

```
// cmdline - procesamiento de opciones en la línea de comando.
//
// $Date: 2012/09/14 13:08:33 $
//
#include <string>
#include <cstdlib>
#include <iostream>
#include "cmdline.h"

#define MSG_ERR_OPEN_FILE "Error al abrir el archivo "

using namespace std;

cmdline::cmdline(){
}

cmdline::cmdline(option_t *table) : option_table(table){
  /*
  - Lo mismo que hacer:

  option_table = table;

  Siendo "option_table" un atributo de la clase cmdline
  y table un puntero a objeto o struct de "option_t".

  Se estaría contruyendo una instancia de la clase cmdline
  cargandole los datos que se hayan en table (la table con
  las opciones, ver el código en main.cc)

  */
}

void
cmdline::parse(int argc, char * const argv[]){
#define END_OF_OPTIONS(p)       \
  ((p)->short_name == 0   \
   && (p)->long_name == 0 \
   && (p)->parse == 0)

  // Primer pasada por la secuencia de opciones: marcamos
  // todas las opciones, como no procesadas. Ver código de
  // abajo.
  //
  for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op)
    op->flags &= ~OPT_SEEN;

  // Recorremos el arreglo argv. En cada paso, vemos
  // si se trata de una opción corta, o larga. Luego,
  // llamamos a la función de parseo correspondiente.
  //
  for (int i = 1; i < argc; ++i) {
    // Todos los parámetros de este programa deben
    // pasarse en forma de opciones. Encontrar un
    // parámetro no-opción es un error.
    //
    if (argv[i][0] != '-') {
      cerr << "Invalid non-option argument: "
           << argv[i]
           << endl;
      exit(1);
    }
```

```
62
63      // Usamos "--" para marcar el fin de las
64      // opciones; todo los argumentos que puedan
65      // estar a continuación no son interpretados
66      // como opciones.
67      //
68      if (argv[i][1] == '-'
69          && argv[i][2] == 0)
70        break;
71
72      // Finalmente, vemos si se trata o no de una
73      // opción larga; y llamamos al método que se
74      // encarga de cada caso. -input
75      //
76      if (argv[i][1] == '-')
77        i += do_long_opt(&argv[i][2], argv[i + 1]);
78      else
79        i += do_short_opt(&argv[i][1], argv[i + 1]);
80    }
81
82    // Segunda pasada: procesamos aquellas opciones que,
83    // (1) no hayan figurado explícitamente en la línea
84    // de comandos, y (2) tengan valor por defecto.
85    //
86    for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op) {
87  #define OPTION_NAME(op) \
88    (op->short_name ? op->short_name : op->long_name)
89      if (op->flags & OPT_SEEN)
90        continue;
91      if (op->flags & OPT_MANDATORY) {
92        cerr << "Option "
93             << "-"
94             << OPTION_NAME(op)
95             << " is mandatory."
96             << "\n";
97        exit(1);
98      }
99      if (op->def_value == 0)
100       continue;
101     op->parse(string(op->def_value));
102   }
103 }
104
105 int
106 cmdline::do_long_opt(const char *opt, const char *arg){
107   // Recorremos la tabla de opciones, y buscamos la
108   // entrada larga que se corresponda con la opción de
109   // línea de comandos. De no encontrarse, indicamos el
110   // error.
111   //
112   for (option_t *op = option_table; op->long_name != 0; ++op) {
113     if (string(opt) == string(op->long_name)) {
114       // Marcamos esta opción como usada en
115       // forma explícita, para evitar tener
116       // que inicializarla con el valor por
117       // defecto.
118       //
119       op->flags |= OPT_SEEN;
120
121       if (op->has_arg) {
122         // Como se trada de una opción
123         // con argumento, verificamos que
124         // el mismo haya sido provisto.
125         //
126         if (arg == 0) {
127           cerr << "Option requires argument: "
128                << "--"
129                << opt
130                << "\n";
```

```
131           exit(1);
132         }
133         op->parse(string(arg));
134         return 1;
135       } else {
136         // Opción sin argumento.
137         //
138         op->parse(string(""));
139         return 0;
140       }
141     }
142   }
143
144   // Error: opción no reconocida. Imprimimos un mensaje
145   // de error, y finalizamos la ejecución del programa.
146   //
147   cerr << "Unknown option: "
148        << "--"
149        << opt
150        << "."
151        << endl;
152   exit(1);
153
154   // Algunos compiladores se quejan con funciones que
155   // lógicamente no pueden terminar, y que no devuelven
156   // un valor en esta última parte.
157   //
158   return -1;
159 }
160
161 int
162 cmdline::do_short_opt(const char *opt, const char *arg)
163 {
164   option_t *op;
165
166   // Recorremos la tabla de opciones, y buscamos la
167   // entrada corta que se corresponda con la opción de
168   // línea de comandos. De no encontrarse, indicamos el
169   // error.
170   //
171   for (op = option_table; op->short_name != 0; ++op) {
172     if (string(opt) == string(op->short_name)) {
173       // Marcamos esta opción como usada en
174       // forma explícita, para evitar tener
175       // que inicializarla con el valor por
176       // defecto.
177       //
178       op->flags |= OPT_SEEN;
179
180       if (op->has_arg) {
181         // Como se trata de una opción
182         // con argumento, verificamos que
183         // el mismo haya sido provisto.
184         //
185         if (arg == 0) {
186           cerr << "Option requires argument: "
187                << "-"
188                << opt
189                << "\n";
190           exit(1);
191         }
192         op->parse(string(arg));
193         return 1;
194       } else {
195         // Opción sin argumento.
196         //
197         op->parse(string(""));
198         return 0;
199       }
```

```
200       }
201     }
202
203     // Error: opción no reconocida. Imprimimos un mensaje
204     // de error, y finalizamos la ejecución del programa.
205     //
206     cerr << "Unknown option: "
207          << "-"
208          << opt
209          << "."
210          << endl;
211     exit(1);
212
213     // Algunos compiladores se quejan con funciones que
214     // lógicamente no pueden terminar, y que no devuelven
215     // un valor en esta última parte.
216     //
217     return -1;
218 }
```

5.8. cmdline.h

```
1  #ifndef _CMDLINE_H_INCLUDED_
2  #define _CMDLINE_H_INCLUDED_
3
4  #include <string>
5  #include <iostream>
6
7  #define OPT_DEFAULT   0
8  #define OPT_SEEN      1
9  #define OPT_MANDATORY 2
10
11 struct option_t{
12   int has_arg;
13   const char *short_name;
14   const char *long_name;
15   const char *def_value;
16   void (*parse)(std::string const &); // Puntero a función de opciones
17   int flags;
18 };
19
20 class cmdline {
21
22   // Este atributo apunta a la tabla que describe todas
23   // las opciones a procesar. Por el momento, sólo puede
24   // ser modificado mediante contructor, y debe finalizar
25   // con un elemento nulo.
26
27   option_t *option_table;
28
29   // El constructor por defecto cmdline::cmdline(), es
30   // privado, para evitar construir "parsers" (analizador
31   // sintáctico, recibe una palabra y lo interpreta en
32   // una acción dependiendo su significado para el programa)
33   // sin opciones. Es decir, objetos de esta clase sin opciones.
34
35   cmdline();
36   int do_long_opt(const char *, const char *);
37   int do_short_opt(const char *, const char *);
38
39 public:
40
41   explicit cmdline(option_t *);
42   void parse(int, char * const []);
43 };
44
45 #endif
```

5.9. file.cpp

```
#include <fstream>
#include <string>
#include "array.h"
#include "file.h"
#include "messages.h"


using namespace std;

// Función que realiza la lectura del archivo de entrada (input file). Se leen las líneas del archivo de texto y se las
// inserta a un vector de cadenas (mediante addValueEnd). Luego se retorna dicho vector para ser validado posteriormente.
// Además, si no se recibe ningún archivo vía línea de comandos, se utiliza el flujo de entrada cin.

string readCin(){
  string str;

  cout << MSG_CMD_INSERT_FX << endl;
  cout << "> ";
  getline(cin, str);
  if (str != "\0")
    return str;
  return str;
}


Array<string> readFile(fstream &fileName){
  Array<string> v(0);
  string str;

  if (fileName.is_open()){
    while(getline(fileName,str)){
      v.addValueEnd(str);
    }
    fileName.close();
  }
  else
    cerr << ERR_CANT_OPEN_FILE << endl;
  return v;
}


// Función que recibe un bloque validado y procesado y lo imprime en un archivo de salida .txt. Si no se recibe
// ningún archivo de salida por línea de comandos, se procede a imprimir el bloque por el flujo de salida cout.

void writeFile(fstream &outFile, Array <string> arr){
  if(outFile.is_open()){
    outFile << arr;
    outFile.close();
  }
}
```

5.10. file.h

```
#ifndef FILE__H
#define FILE__H

#include <string>
#include <iostream>

#include "block.h"

```

```
using namespace std;

Array<string> readFile(fstream &fileName);
Array<string> splitStr(string str_, char delim);
void writeFile(fstream &outFile, Array <string> arr);
string readCin();

#endif
```

5.11. header.cpp

```
#include <string>
#include <iostream>
#include "header.h"
#include "sha256.h"
#include "validation.h"
#include "messages.h"
#include "utility.h"


using namespace std;

// Constructores.

Header::Header(){
}

Header::Header(string prevb, string txhash, unsigned int b, unsigned int n){
    prev_block = prevb;
    txns_hash = txhash;
    bits = b;
    nonce = n;
}

Header::Header(const Header &headr){
    prev_block = headr.prev_block;
    txns_hash = headr.txns_hash;
    bits = headr.bits;
    nonce = headr.nonce;
}

// Destructor.

Header::~Header(){
    prev_block.clear();
    txns_hash.clear();
    bits = 0;
    nonce = 0;
}

// Sobrecarga de operadores

Header &Header::operator=(const Header &headr){
    prev_block = headr.prev_block;
    txns_hash = headr.txns_hash;
    bits = headr.bits;
    nonce = headr.nonce;
    return *this;
}

bool Header::operator==(const Header &headr){
    return (prev_block == headr.prev_block &&
    txns_hash == headr.txns_hash &&
    bits == headr.bits &&
    nonce == headr.nonce);
}

// Métodos de instancia.

```

```
59 string const Header:: getPrev_block(){
60     return prev_block;
61 }
62
63 string const Header:: getTxns_hash(){
64     return txns_hash;
65 }
66
67 unsigned int const Header:: getBits(){
68     return bits;
69 }
70
71 unsigned int const Header:: getNonce(){
72     return nonce;
73 }
74
75 void Header:: setPrevBlock(string str){
76     prev_block = str;
77 }
78 void Header:: setTxnsHashManually(string str){
79     txns_hash = str;
80 }
81
82  // Método que recibe un arreglo de cadenas de caracteres donde cada posición corresponde
       a una línea del archivo de entrada.
83  // Luego procede a generar una sola cadena de caracteres con dicha información y procede
       a calcular el doble hash de dicha cadena
84  // para cargar el valor al miembro txns_hash.
85
86 void Header:: setTxnsHash(Array<string> str_arr){
87     string tx_str,hash;
88     size_t i = 0;
89
90     while(i<str_arr.getSize()){
91         tx_str.append(str_arr[i] + '\n');
92         i++;
93     }
94     hash = sha256(sha256(tx_str));
95     txns_hash = hash;
96 }
97
98 void Header:: setBits(unsigned int b){
99     bits = b;
100 }
101
102 void Header:: setNonce(unsigned int n){
103     nonce = n;
104 }
105
106 ostream & operator<< (ostream &os, Header headr){
107     os << headr.prev_block << endl;
108     os << headr.txns_hash << endl;
109     os << headr.bits << endl;
110     os << headr.nonce;
111     return os;
112 }
113
114 // Método que devuelve una cadena de caracteres con todos los miembros de la clase
       separados por un salto de línea.
115
116 string Header:: toStr(){
117     string hdr_str;
118
119     hdr_str += getPrev_block() + BREAK_LINE;
120     hdr_str += getTxns_hash() + BREAK_LINE;
121     hdr_str += to_string(getBits()) + BREAK_LINE;
122     hdr_str += to_string(getNonce()) + BREAK_LINE;
123
124     return hdr_str;
```

```
}

// Este metodo comienza realizando el doble hash del header completo y luego, en base al valor de la dificultad ingresado por línea de comando
// procede a validar el hash, mientras no logre cumplir con dicho parametro, irá incrementando el campo nonce y volverá a calcular el doble hash
// del header completo hasta hallar el nonce que cumpla la dificultad.

void Header:: validateHash(unsigned int difficulty){
    string hdr_str,hdr_hash;
    hdr_str = toStr();
    hdr_hash = sha256(sha256(hdr_str));
    map<char,int> zeros(zeroBitsMap());

    while(!validateDifficulty(hdr_hash, zeros, difficulty)){
        setNonce(getNonce() + NONCE_INCREM);
        hdr_str = toStr();
        hdr_hash = sha256(sha256(hdr_str));
    }
}

bool Header::validateHeaderDifficulty(){
    map<char,int> zeros(zeroBitsMap());
    string hdr_hash = sha256(sha256(toStr()));

    return validateDifficulty(hdr_hash, zeros, bits);
}
```

5.12. header.h

```
#ifndef _HEADER_H_INCLUDED_
#define _HEADER_H_INCLUDED_

#include <iostream>
#include <string>
#include "array.h"

#define NONCE_INCREM 1
#define BREAK_LINE '\n'

using namespace std;

class Header {
    string prev_block;
    string txns_hash;
    unsigned int bits;
    unsigned int nonce;

public:

    Header();
    Header(string prev_block ,string txns_hash, unsigned int bits, unsigned int nonce);
    Header(const Header &);
    ~Header();
    Header &operator=(const Header &);
    bool operator==(const Header &);

    string const getPrev_block();
    string const getTxns_hash();
    string toStr();
    unsigned int const getBits();
    unsigned int const getNonce();

    void setPrevBlock(string str);
    void setTxnsHashManually(string str);
    void setTxnsHash(Array<string> tx_arr);
    void setBits(unsigned int b);
    void setNonce(unsigned int n);
    void validateHash(unsigned int difficulty);
```

```
40      bool validateHeaderDifficulty();
41
42      friend ostream & operator<< (ostream &os, Header headr);
43  };
44
45
46  #endif
```

5.13. input.cpp

```
1   #include <string>
2   #include "input.h"
3   #include "array.h"
4   #include "file.h"
5   #include "output.h"
6   #include "messages.h"
7   #include <iostream>
8
9   using namespace std;
10
11  // Constructores.
12
13  Input::Input(){
14    Outpoint outpoint;
15    addr.clear();
16  }
17
18  Input::Input(Outpoint oup, string add){
19    outpoint = oup;
20    addr = add;
21  }
22
23  Input::Input(const Input &out){
24    outpoint = out.outpoint;
25    addr = out.addr;
26  }
27
28  // Destructor.
29
30  Input::~Input(){
31  }
32
33  // Sobrecarga de operadores.
34
35  Input &Input::operator=(const Input &out){
36    outpoint = out.outpoint;
37    addr = out.addr;
38    return *this;
39  }
40
41  bool Input::operator==(const Input &out){
42    return (outpoint == out.outpoint && addr == out.addr);
43  }
44
45  ostream & operator<< (ostream &os, Input input){
46    os << input.outpoint << DELIM_DATA << input.addr;
47    return os;
48  }
49
50  // Métodos de instancia.
51
52  Outpoint Input::getOutpoint(){
53    return outpoint;
54  }
55  string const Input::getAddr(){
56    return addr;
57  }
58
59  void Input::setOutpoint(Outpoint out){
```

```
60    outpoint = out;
61  }
62
63  void Input::setAddr(string add){
64    addr = add;
65  }
66
67   // Método que recibe una cadena de caracteres con toda la información correspondiente a
        los miembros de la clase Input.
68   // Luego separa dicha cadena por medio de la función splitStr, la cual retorna un vector
         con los valores de los miembros los cuales
69   // serán asignados por medio de los setters.
70
71  void Input::setInput(string str){
72    Array<string> str_field(0);
73    str_field = splitStr(str, DELIM_TX);
74    Input newinput;
75    Outpoint new_outpoint(str_field[0], stoul(str_field[1]));
76
77    setAddr(str_field[2]);
78    setOutpoint(new_outpoint);
79  }
80
81  string Input::toString(){
82    string input_str;
83    input_str += outpoint.getTx_id();
84    input_str += " ";
85    input_str += to_string(outpoint.getIdx());
86    input_str += " ";
87    input_str += addr;
88
89    return input_str;
90  }
```

5.14. input.h

```
1   #ifndef _INPUT_H_INCLUDED_
2   #define _INPUT_H_INCLUDED_
3
4   #include <iostream>
5   #include <string>
6   #include "outpoint.h"
7
8   using namespace std;
9
10  class Input {
11    friend class Outpoint;
12    Outpoint outpoint;
13    string addr;
14
15  public:
16    Input();
17    Input(Outpoint, string);
18    Input(const Input &);
19    ~Input();
20    Input &operator=(const Input &);
21    bool operator==(const Input &);
22
23    Outpoint getOutpoint();
24    string const getAddr();
25
26    void setInput(string);
27    void setOutpoint(Outpoint);
28    void setAddr(string);
29    string toString();
30
31    friend ostream & operator<< (ostream &, Input);
32  };
33
```

```
34
35 #endif
```

5.15. list.h

```
1  #ifndef LIST_H
2  #define LIST_H
3
4  #include <iostream>
5
6  #include "block.h"
7  #include "sha256.h"
8  #include <string>
9
10
11 #define STATUS_FAIL "FAIL"
12
13 template <typename T> class List;
14
15 template<typename T>
16 class Node {
17   friend class List <T>;
18
19 private:
20   T data;
21   Node <T> *next;
22   Node <T> *prev;
23
24 public:
25   Node(T &value);
26   T getData();
27   Array <T> getAll();
28   Node <T>* getNext();
29
30   void remove(List<T> *);
31   void removeAll();
32
33 };
34
35
36 template <typename T>
37 class List{
38 private:
39   Node<T> *first;
40
41 public:
42   List();
43   ~List();
44
45
46   Node<Block>* findByBlockHash(string blockid);
47   string findByTxnHash(string txnid);
48   Node<T>* getFirst();
49   void setFirst(Node<T> *);
50   void addNodeEnd(Node <T> node);
51   void deleteFirstNode();
52   Node<T> getLastNode();
53   void destroy();
54
55   void print() const;
56   bool isEmpty() const;
57 };
58
59 // En esta clase fueron definidos métodos pensados para utilizar en futuros trabajos prá
       cticos, ya que exceden el
60 // alcance de éste. Con esto presente, algunas de las funciones contienen mensajes "
       hardcodeados" dentro.
61
62 using namespace std;
```

```
63
64 template <typename N> Node<N> ::Node(N &value){
65   data = value;
66   next = 0;
67 }
68
69 template <typename T> T Node<T>::getData(){
70   return data;
71 }
72
73 template <typename T> List<T>::List(){
74   first = 0;
75 }
76
77 template <typename T> List<T>::~List(){
78   Node<T> *aux = first;
79
80   while (!isEmpty()){
81     first = first->next;
82     delete aux;
83     aux = first;
84   }
85 }
86
87
88 template <typename T> void List<T>::addNodeEnd(Node <T> node){
89   Node<T> *aux1;
90   Node<T> *aux2;
91   aux1 = new Node<T>(node);
92
93   if(isEmpty()){              //si la lista está vacia, lo agrega al comienzo
94     first = aux1;
95     first->next = 0;
96     first->prev = 0;
97   } else{
98     aux2 = first;
99     while(aux2->next)
100      aux2 = aux2->next;
101    aux2->next =aux1;
102    aux1->prev =aux2;
103    aux1->next = 0;
104  }
105 }
106
107
108 template <typename T> void List<T>::print() const{
109   int i=0;
110   Node<T> *aux=first;
111
112   if(!isEmpty()){
113     while(aux){
114       cout << "El nodo " << i <<" de la lista contiene el valor " << endl << aux->data <<
      endl;
115       aux = aux->next;
116       i++;
117     }
118   }
119 }
120
121 template <typename T> bool List<T>:: isEmpty() const{
122   return (first==0);
123 }
124
125
126 template <typename T> void List<T>::deleteFirstNode(){
127   Node<T> *aux1, *aux2;
128
129   if (!isEmpty()){
130     if (first->next==first){
```

```
      delete first;
      first=0;
    }
  else
  {
    aux1=first;
    aux2=aux1->next;
    while (aux2->next!=first)
      aux2=aux2->next;
    aux2->next=aux1->next;
    first =first->next;
    delete aux1;
  }
  }
}


template <typename T> void List<T>::setFirst(Node<T> * node){
  first = node;
}



template <typename T> void Node<T>::remove(List<T> * list){
  if(prev && next){
    prev->next = next;
    delete this;
  }else if(prev){
    prev->next = 0;
    delete this;
  }else if(next){
    next->prev = 0;
    list->setFirst(next);
    delete this;
  }else{
    next = 0;
    list->destroy();
  }
}

template <typename T> void Node<T>::removeAll(){
  if(next)
    next->removeAll();
  delete this;
}

template <typename T> Node<T> List<T>::getLastNode(){
  Node<T> *aux = first;

  while(aux->next)
    aux = aux->next;
  return (*aux);
}

template <typename T> void List<T>::destroy(){

  while(!isEmpty()){
    first->removeAll();
    first = 0;
  }

  return;

}

template <typename T> Node<Block>* List<T>::findByBlockHash(string blockid){
  Node <Block> *aux = first;
  string block_hash;

```

```
  while(aux){
    block_hash = (aux->data).getBlockHash();
    if (block_hash==blockid)
      return aux;
    aux=aux->next;
  }
  return NULL;
}



template <typename T> Array <T> Node<T>::getAll(){
  Node <T> *aux = this;
  Array <T> arr;

  while(aux){
    arr.addValueEnd((*aux).getData());
    aux = aux->next;
  }
  return arr;
}


template <typename T> Node <T>* List<T>::getFirst(){
  return first;
}

template <typename T> Node <T>* Node<T>::getNext(){
  return next;
}

template <typename T> string List<T>::findByTxnHash(string txnid){
  Node <Block> *aux = first;
  Array<Transaction> txn_arr;
  unsigned int txn_count;
  size_t i;

  while(aux){
    txn_count = ((aux->data).getBody()).getTxn_count();
    txn_arr = ((aux->data).getBody()).getTx_arr();

    for(i = 0; i < txn_count; i++){
      if(txnid == txn_arr[i].getTxHash()){
        return txn_arr[i].toString();
      }
    }
    aux=aux->next;
  }
  return STATUS_FAIL;
}

#endif
```

5.16. makefile

```
target = TP1

extension = cpp

CXXSTD = c++11
CXXFLAGS =  -Wall -pedantic

LD = $(CXX)

fuentes ?= $(wildcard *.$(extension))

occ := $(CC)
ocxx := $(CXX)
```

```
orm := $(RM)
old := $(LD)

all: $(target)

o_files = $(patsubst %.$(extension),%.o,$(fuentes))

$(target): $(o_files)
  $(LD) $(o_files) -o $(target) $(LDFLAGS)

clean:
  $(RM) $(o_files) $(target)

valgrind:
  valgrind --leak-check=full --track-origins=yes ./TP1

txtvalgrind:
  valgrind --leak-check=full ./TP1 -d 3  -i entrada_cmd.txt -o salida_cmd.txt

test1:
  ./TP0 -i test1.txt -d 3
test2:
  ./TP0 -i test2.txt -d 3
test3:
  ./TP0 -i test3.txt -d 3
test4:
  ./TP0 -i test4.txt -d 3
test5:
  ./TP0 -i test5.txt -d 3
test6:
  ./TP0 -i test6.txt -d 3
test7:
  ./TP0 -i test7.txt -d 3
```

5.17. messages.h

```
// Command line messages
#define MSG_CMD_INSERT_FX "Welcome to the Algochain, please insert option or CTRL + D to exit."

// Error messages
#define ERR_CANT_OPEN_FILE "Can't open the file"
#define ERR_FILE_NO_DATA "File did not contain any data"
#define ERR_EMPTY_FILE "The file is empty"
#define ERR_INPUTS_QUANT "Wrong amount of inputs"
#define ERR_FORMAT_QUANT_INPUT "Input amount is not a digit"
#define ERR_FILE_INCOMPLETE "The file is incomplete"
#define ERR_OUTPUTS_QUANT "Wrong amount of outputs"
#define ERR_FORMAT_QUANT_OUTPUT "Output amount is not a digit"
#define ERR_INPUT_TX_ID_HEXA "Transaction ID hash of the input is not hexadecimal"
#define ERR_INPUT_TX_ID_LENGTH "Wrong length of Transaction ID hash of the input"
#define ERR_INPUT_IDX "IDX of the input is not a valid number"
#define ERR_INPUT_ADDR_HEXA "Address hash of the input is not hexadecimal"
#define ERR_INPUT_ADDR_LENGTH "Wrong length of Address hash of the input"
#define ERR_INPUT_ONE_ARG "Only one argument of input given (should have three)"
#define ERR_INPUT_TWO_ARGS "Only two arguments of input given (should have three)"
#define ERR_INPUT_TOO_MANY_ARGS "Too many arguments of input given (should only have three)"
#define ERR_OUTPUT_VALUE "Value of the output is not a valid number"
#define ERR_OUTPUT_ADDR_HEXA "Address hash of the output is not hexadecimal"
#define ERR_OUTPUT_ADDR_LENGTH "Wrong length of Address hash of the output"
#define ERR_OUTPUT_ONE_ARG "Only one argument of output given (should have two)"
#define ERR_OUTPUT_TO_MANY_ARGS "Too many arguments of output given (should only have two)"
#define ERR_INVALID_INDEX "Invalid index"
#define ERR_EMPTY_LIST "List is empty"


#define ERR_WRONG_CMD "The option isn't correct "
```

```

#define MSG_LINE_NUMBER "Line number "
#define ERR_INVALID_FILE_FORMAT " in file doesn't fit any option format"
```

5.18. options.cpp

```
#include "validation.h"
#include "list.h"
#include "block.h"
#include "sha256.h"
#include <map>
#include "array.h"
#include "utility.h"
#include "options.h"
#include "file.h"
#include <fstream>
#include <iostream>
#include "messages.h"
#include "unspent.h"

#define ERR_TRANSFER_FORMAR "Incorrect transfer option format"

map<string,int> optionsMap(){
  map<string,int> options;
  options[OPT_1] = 0;
  options[OPT_2] = 1;
  options[OPT_3] = 2;
  options[OPT_4] = 3;
  options[OPT_5] = 4;
  options[OPT_6] = 5;
  options[OPT_7] = 6;
  options[OPT_8] = 7;
  return options;
}

string callOption(string str, map<string,int> opt_map, List<Block> * list, Array <
    Transaction> *mempool){
  Array<string> cmd_arr = splitStr(str, ' ');
  string str_to_file;

  switch (opt_map[cmd_arr[0]]){
    case 0:
      str_to_file = optInit(cmd_arr, list, mempool);
      break;

    case 1:
      str_to_file = optTransfer(cmd_arr,list, mempool);
      break;

    case 2:
      str_to_file = optMine(cmd_arr,list, mempool);
      break;

    case 3:
      str_to_file = optBalance(cmd_arr, list, mempool);
      break;

    case 4:
      str_to_file = optBlock(cmd_arr, list);
      break;

    case 5:
      str_to_file = optTxn(cmd_arr, list);
      break;

    case 6:
      str_to_file = optLoad(cmd_arr, list, mempool);
      break;
```

```
62
63      case 7:
64        str_to_file = optSave(cmd_arr, list);
65        break;
66    }
67    return str_to_file;
68
69  }
70
71  bool optionValid(string opt){
72    Array <string> cmd_str;
73
74    cmd_str = splitStr(opt, ' ');
75    return cmd_str[0] == OPT_1 || cmd_str[0] == OPT_2  || cmd_str[0] == OPT_3 || cmd_str[0]
         == OPT_4  || cmd_str[0] == OPT_5  ||
76    cmd_str[0] == OPT_6  || cmd_str[0] == OPT_7  || cmd_str[0] == OPT_8 ;
77  }
78
79
80  string optInit (Array<string> cmd_arr, List <Block> * algochain, Array<Transaction> *
        mempool){
81    if (!validateInitFormat(cmd_arr))
82      return STATUS_FAIL;
83
84    string block_hash;
85
86    if(!(*algochain).isEmpty()){
87      (*algochain).destroy();
88    }
89
90    Output out(stof(cmd_arr[2]), sha256(cmd_arr[1]));
91    Input in; // Constructor de Input vacío que a su vez llama a constructor de outpoint
        vacío (el input debe referenciar un outpoint nulo)
92
93    in.setAddr(NULL_HASH);
94
95    Array<Input> arr_in(0);
96    arr_in.addValueEnd(in);
97    Array<Output> arr_out(0);
98    arr_out.addValueEnd(out);
99
100   Transaction txn(arr_in, arr_out);
101
102   Array<Transaction> arr_txn(0);
103   arr_txn.addValueEnd(txn);
104
105   Body bod(GENESIS_N_TXN, arr_txn);
106   Header hdr(NULL_HASH, DEFAULT_TXN_HASH , (unsigned int)stoi(cmd_arr[3]), DEFAULT_NONCE
        );
107
108   Block block(hdr, bod);
109   block.setMerkleHash();
110   hdr = block.getHeader();
111
112   hdr.validateHash(stoul(cmd_arr[3]));
113
114   block.setHeader(hdr);
115
116   Node<Block> node(block);
117   (*algochain).addNodeEnd(node);
118
119   (*mempool).restartArray();
120
121   return block.getBlockHash();
122
123 }
124
125
126 string optBlock (Array<string> cmd_arr, List <Block> * algochain){
```

```
  if(!validateOptBlockFormat(cmd_arr))
    return STATUS_FAIL;

  Node <Block> * bk_node;
  Block bk;

  bk_node = (*algochain).findByBlockHash(cmd_arr[1]);

  if(bk_node == NULL)
    return STATUS_FAIL;

  return ((*bk_node).getData()).toString();
}




string optSave(Array <string> cmd_arr,  List <Block> * algochain){
  if(!validateSaveFormat(cmd_arr))
    return STATUS_FAIL;

  ofstream savefile;

  Array<Block> blocks_arr(0);
  Node <Block> *first_node = (*algochain).getFirst();

  blocks_arr = (*first_node).getAll();

  savefile.open(cmd_arr[1]);

  if(savefile.is_open()){
    for(size_t i = 0; i<blocks_arr.getSize();i++)
      savefile << blocks_arr[i].toString()<<endl;
    savefile.close();
    return STATUS_OK;
  }
  else return STATUS_FAIL;
}

string optLoad(Array <string> cmd_arr,  List <Block> * algochain, Array<Transaction> *
    mempool){
  if(!validateLoadFormat(cmd_arr))
    return STATUS_FAIL;

  Array<string> fvalues(0);
  fstream loadfile;
  Array<Array<string>> block_string_arr(0);
  Array<Block> block_array(0);
  size_t i;

  loadfile.open(cmd_arr[1]);

  if (loadfile.is_open()){
    fvalues = readFile(loadfile);
    block_string_arr = fileToBlock(fvalues);
  }else
  {
    cerr << ERR_CANT_OPEN_FILE << endl;
    return STATUS_FAIL;
  }
  for(size_t i=0;i<block_string_arr.getSize() ;i++){
    if(!validateBlockFormat(block_string_arr[i])){
      return STATUS_FAIL;
    }
  }

  for (i = 0; i < block_string_arr.getSize(); i++){
    Block curr_block;
    curr_block = stringToBlock(block_string_arr[i]);
```

```
195       block_array.addValueEnd(curr_block);
196     }
197
198     if(!validateLoadedAlgochain(block_array))
199       return STATUS_FAIL;
200
201     (*algochain).destroy();
202
203     for (i = 0; i < block_array.getSize(); i++){
204       Node<Block> node(block_array[i]);
205       (*algochain).addNodeEnd(node);
206     }
207
208     (*mempool).restartArray();
209
210     return block_array[i - 1].getBlockHash();
211 }
212
213 string optTxn (Array<string> cmd_arr, List <Block> * algochain){
214     if(!validateTxnFormat(cmd_arr)){
215       return STATUS_FAIL;
216     }
217
218     string txn;
219
220     txn = (*algochain).findByTxnHash(cmd_arr[1]);
221
222     if(txn==STATUS_FAIL)
223       return STATUS_FAIL;
224
225     return txn;
226 }
227
228
229
230 string optTransfer(Array<string> cmd_arr,List <Block> * algochain, Array<Transaction> *
        mempool){
231     if(!validateTransferFormat(cmd_arr)){
232       return STATUS_FAIL;
233     }
234
235     float sum;
236     float unspent;
237     float funds = 0;
238
239     List<Unspent> unspent_list;
240     Array<Output> out_arr(0);
241     Array<Input> in_arr(0);
242
243     sum = sumValue(cmd_arr);
244     userUnspent(sha256(cmd_arr[1]), &unspent_list, algochain);
245
246     unspent = totalUnspent(&unspent_list, mempool,sha256(cmd_arr[1]));
247
248     if(unspent < sum || unspent == 0)
249         return STATUS_FAIL;
250
251     Node <Unspent> *it_unspent = unspent_list.getFirst();
252
253     for(size_t i = 2; i<cmd_arr.getSize();i+=2){
254       out_arr.addValueEnd(Output(stof(cmd_arr[i+1]), sha256(cmd_arr[i])));
255     }
256     while(it_unspent && funds < sum)
257     {
258       Input new_input(it_unspent->getData().getOutpoint(),sha256(cmd_arr[1]));
259       funds+=it_unspent->getData().getValue();
260       in_arr.addValueEnd(new_input);
261     }
262     if((funds-sum)>0)
```

```
263     out_arr.addValueEnd(Output(funds-sum, sha256(cmd_arr[1])));
264
265   Transaction new_tx(in_arr,out_arr);
266
267   (*mempool).addValueEnd(new_tx);
268
269   return new_tx.getTxHash();
270 }
271
272 string optBalance(Array<string> cmd_arr, List <Block> * algochain, Array<Transaction> *
      mempool){
273   if(!validateBalanceFormat(cmd_arr))
274     return STATUS_FAIL;
275
276   List<Unspent> unspent_list;
277   float unspent;
278
279   userUnspent(sha256(cmd_arr[1]), &unspent_list, algochain);
280   unspent = totalUnspent(&unspent_list, mempool,sha256(cmd_arr[1]));
281   return to_string(unspent);
282 }
283
284 string optMine(Array<string> cmd_arr, List<Block> *algochain,  Array <Transaction> *
      mempool){
285   if(!validateMineFormat(cmd_arr))
286     return STATUS_FAIL;
287
288   Body new_body;
289   Header new_header;
290   string prev_block_hash;
291
292   new_body.setTxn_count((*mempool).getSize());
293   new_body.setTx_arr(*mempool);
294
295   Block prev_block = (*algochain).getLastNode().getData();
296   new_header.setPrevBlock(prev_block.getBlockHash());
297
298   new_header.setBits(stoul(cmd_arr[1]));
299   new_header.setNonce(DEFAULT_NONCE);
300
301   Block new_block(new_header,new_body);
302   new_block.setMerkleHash();
303
304
305   new_header = new_block.getHeader();
306   new_header.validateHash(stoul(cmd_arr[1]));
307
308   new_block.setHeader(new_header);
309
310   (*mempool).restartArray();
311
312   Node <Block> node(new_block);
313   (*algochain).addNodeEnd(node);
314
315   return new_block.getBlockHash();
316
317 }
```

5.19. options.h

```
1 #ifndef _OPTIONS_H_INCLUDED_
2 #define _OPTIONS_H_INCLUDED_
3
4 #include "array.h"
5 #include <string>
6 #include <map>
7 #include "list.h"
8
9 #define TX_DELIM " "
```

```
#define GENESIS_N_TXN 1
#define NULL_HASH "0000000000000000000000000000000000000000000000000000000000000000"
#define DEFAULT_TXN_HASH "TRANSACTION HASH ACA"

#define OPT_1 "init"
#define OPT_2 "transfer"
#define OPT_3 "mine"
#define OPT_4 "balance"
#define OPT_5 "block"
#define OPT_6 "txn"
#define OPT_7 "load"
#define OPT_8 "save"

#define STATUS_FAIL "FAIL"
#define STATUS_OK "OK"

using namespace std;

string optBlock (Array<string> cmd_arr, List <Block> * algochain);
string optSave(Array <string> cmd_arr,  List <Block> * algochain);
string optTxn(Array <string> cmd_arr,  List <Block> * algochain);
string optInit (Array<string> cmd_arr, List <Block> * algochain, Array<Transaction> *
    mempool);
string optLoad(Array <string> cmd_arr,  List <Block> * algochain, Array<Transaction> *
    mempool);
string optBalance(Array<string> cmd_arr, List <Block> * algochain, Array<Transaction> *
    mempool);
string optMine(Array<string> cmd_arr, List<Block> *algochain, Array <Transaction> *
    mempool);
string optTransfer (Array<string> cmd_arr, List <Block> * algochain, Array<Transaction> *
     mempool);

bool optionValid(string opt);
map<string,int> optionsMap();

string callOption(string str, map<string,int> opt_map, List<Block> * list, Array <
    Transaction> *mempool);

#endif
```

5.20. outpoint.cpp

```
#include "outpoint.h"
#include "messages.h"

#define NULL_HASH "0000000000000000000000000000000000000000000000000000000000000000"

using namespace std;

// Constructores.

Outpoint::Outpoint(){
    tx_id = NULL_HASH;
    idx = 0;
}

Outpoint::Outpoint(string tx, unsigned int id){
    tx_id = tx;
    idx = id;
}
Outpoint::Outpoint(const Outpoint &out){
    tx_id = out.tx_id;
    idx = out.idx;
}

// Destructor.

Outpoint::~Outpoint(){
}
```

```
28
29 // Sobrecarga de operadores
30
31 Outpoint &Outpoint::operator=(const Outpoint &out){
32     tx_id = out.tx_id;
33     idx = out.idx;
34     return *this;
35 }
36
37 bool Outpoint::operator==(const Outpoint &out){
38     return tx_id == out.tx_id && idx == out.idx;
39 }
40
41 ostream & operator<< (ostream &os, Outpoint outp)
42 {
43     os<< outp.tx_id<< DELIM_DATA << outp.idx;
44     return os;
45 }
46
47 // Métodos de instancia.
48
49 string Outpoint::getTx_id(){
50     return tx_id;
51 }
52
53 unsigned int const Outpoint::getIdx(){
54     return idx;
55 }
56
57 void Outpoint::setTx_id(string txid){
58     tx_id = txid;
59 }
60
61 void Outpoint::setIdx(unsigned int id){
62     idx = id;
63 }
```

5.21. outpoint.h

```
1  #ifndef _OUTPOINT_H_INCLUDED_
2  #define _OUTPOINT_H_INCLUDED_
3
4  #include <iostream>
5  #include <string>
6
7  #define DELIM_DATA " "
8
9  using namespace std;
10
11 class Outpoint {
12     string tx_id;
13     unsigned int idx;
14 public:
15
16     Outpoint();
17     Outpoint(string, unsigned int);
18     Outpoint(const Outpoint &);
19     ~Outpoint();
20
21     Outpoint &operator=(const Outpoint &);
22     bool operator==(const Outpoint &);
23
24     string getTx_id();
25     unsigned int const getIdx();
26     void setTx_id(string);
27     void setIdx(unsigned int);
28
29     friend ostream & operator<< (ostream &, Outpoint);
30 };
```

```
31
32 #endif
```

5.22. output.cpp

```
1
2  #include "output.h"
3  #include "utility.h"
4  #include "messages.h"
5  #include "array.h"
6
7  using namespace std;
8
9  // Constructores.
10
11 Output::Output(){
12     value = 0;
13 }
14
15 Output::Output(float val, string add){
16     value = val;
17     addr = add;
18 }
19
20 Output::Output(const Output &out){
21     value = out.value;
22     addr = out.addr;
23 }
24
25 // Destructor.
26
27 Output::~Output(){
28     value = 0;
29     addr.clear();
30 }
31
32 // Sobrecarga de operadores.
33
34 Output &Output::operator=(const Output &out){
35     value = out.value;
36     addr = out.addr;
37     return *this;
38 }
39
40 bool Output::operator==(const Output &out){
41     return (value == out.value && addr == out.addr);
42 }
43
44 ostream & operator<< (ostream &os, Output outp){
45     os<< outp.value << DELIM_DATA << outp.addr;
46     return os;
47 }
48
49 // Métodos de instancia.
50
51 float const Output::getValue(){
52     return value;
53 }
54
55 string const Output::getAddr(){
56     return addr;
57 }
58
59 void Output::setValue(float val){
60     value = val;
61 }
62
63 void Output::setAddr(string add){
64     addr = add;
```

```
}

// Función que recibe una cadena de caracteres con los valores del output, separa esos valores dependiendo el
// delimitador utilizado y genera un vector con los valores de los campos, luego setea en los campos
// correspondientes del output dichos valores.

void Output:: setOutput(string str){
    Array<string> str_field(0);
    Output newoutput;

    str_field = splitStr(str, DELIM_TX);
    setValue(stof(str_field[0]));
    setAddr(str_field[1]);
}

string Output::toString(){
  string output_str;
    string value_to_str = to_string(getValue());

    value_to_str.erase(value_to_str.find_last_not_of('0') + 1, std::string::npos);
    if(value_to_str.back() == '.'){
        value_to_str.erase(value_to_str.find_last_not_of('.') + 1, std::string::npos);
    }
  output_str += value_to_str;
  output_str += " ";
  output_str += addr;
  return output_str;
}
```

5.23. output.h

```
#ifndef _OUTPUT_H_INCLUDED_
#define _OUTPUT_H_INCLUDED_

#include <iostream>
#include <string>

using namespace std;

class Output {
    float value;
    string addr;

public:

    Output();
    Output(float, string);
    Output(const Output &);
    ~Output();

    Output &operator=(const Output &);
    bool operator==(const Output &);

    float const getValue();
    string const getAddr();

    void setOutput(string);
    void setValue (float);
    void setAddr (string);
    string toString();

    friend ostream & operator<< (ostream &, Output);
};

#endif
```

5.24. sha256.cpp

```
#include <cstring>
#include <fstream>
#include "sha256.h"

const unsigned int SHA256::sha256_k[64] = //UL = uint32
            {0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
             0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
             0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
             0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
             0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
             0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
             0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
             0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
             0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
             0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
             0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
             0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
             0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
             0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
             0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
             0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2};

void SHA256::transform(const unsigned char *message, unsigned int block_nb)
{
    uint32 w[64];
    uint32 wv[8];
    uint32 t1, t2;
    const unsigned char *sub_block;
    int i;
    int j;
    for (i = 0; i < (int) block_nb; i++) {
        sub_block = message + (i << 6);
        for (j = 0; j < 16; j++) {
            SHA2_PACK32(&sub_block[j << 2], &w[j]);
        }
        for (j = 16; j < 64; j++) {
            w[j] =  SHA256_F4(w[j -  2]) + w[j -  7] + SHA256_F3(w[j - 15]) + w[j - 16];
        }
        for (j = 0; j < 8; j++) {
            wv[j] = m_h[j];
        }
        for (j = 0; j < 64; j++) {
            t1 = wv[7] + SHA256_F2(wv[4]) + SHA2_CH(wv[4], wv[5], wv[6])
                + sha256_k[j] + w[j];
            t2 = SHA256_F1(wv[0]) + SHA2_MAJ(wv[0], wv[1], wv[2]);
            wv[7] = wv[6];
            wv[6] = wv[5];
            wv[5] = wv[4];
            wv[4] = wv[3] + t1;
            wv[3] = wv[2];
            wv[2] = wv[1];
            wv[1] = wv[0];
            wv[0] = t1 + t2;
        }
        for (j = 0; j < 8; j++) {
            m_h[j] += wv[j];
        }
    }
}

void SHA256::init()
{
    m_h[0] = 0x6a09e667;
    m_h[1] = 0xbb67ae85;
    m_h[2] = 0x3c6ef372;
    m_h[3] = 0xa54ff53a;
    m_h[4] = 0x510e527f;
```

```
    m_h[5] = 0x9b05688c;
    m_h[6] = 0x1f83d9ab;
    m_h[7] = 0x5be0cd19;
    m_len = 0;
    m_tot_len = 0;
}

void SHA256::update(const unsigned char *message, unsigned int len)
{
    unsigned int block_nb;
    unsigned int new_len, rem_len, tmp_len;
    const unsigned char *shifted_message;
    tmp_len = SHA224_256_BLOCK_SIZE - m_len;
    rem_len = len < tmp_len ? len : tmp_len;
    memcpy(&m_block[m_len], message, rem_len);
    if (m_len + len < SHA224_256_BLOCK_SIZE) {
        m_len += len;
        return;
    }
    new_len = len - rem_len;
    block_nb = new_len / SHA224_256_BLOCK_SIZE;
    shifted_message = message + rem_len;
    transform(m_block, 1);
    transform(shifted_message, block_nb);
    rem_len = new_len % SHA224_256_BLOCK_SIZE;
    memcpy(m_block, &shifted_message[block_nb << 6], rem_len);
    m_len = rem_len;
    m_tot_len += (block_nb + 1) << 6;
}

void SHA256::final(unsigned char *digest)
{
    unsigned int block_nb;
    unsigned int pm_len;
    unsigned int len_b;
    int i;
    block_nb = (1 + ((SHA224_256_BLOCK_SIZE - 9)
                     < (m_len % SHA224_256_BLOCK_SIZE)));
    len_b = (m_tot_len + m_len) << 3;
    pm_len = block_nb << 6;
    memset(m_block + m_len, 0, pm_len - m_len);
    m_block[m_len] = 0x80;
    SHA2_UNPACK32(len_b, m_block + pm_len - 4);
    transform(m_block, block_nb);
    for (i = 0 ; i < 8; i++) {
        SHA2_UNPACK32(m_h[i], &digest[i << 2]);
    }
}

std::string sha256(std::string input)
{
    unsigned char digest[SHA256::DIGEST_SIZE];
    memset(digest,0,SHA256::DIGEST_SIZE);

    SHA256 ctx = SHA256();
    ctx.init();
    ctx.update( (unsigned char*)input.c_str(), input.length());
    ctx.final(digest);

    char buf[2*SHA256::DIGEST_SIZE+1];
    buf[2*SHA256::DIGEST_SIZE] = 0;
    for (unsigned int i = 0; i < SHA256::DIGEST_SIZE; i++)
        sprintf(buf+i*2, "%02x", digest[i]);
    return std::string(buf);
}
```

5.25. sha256.h

```
#ifndef SHA256_H
```

```
#define SHA256_H
#include <string>

class SHA256
{
protected:
    typedef unsigned char uint8;
    typedef unsigned int uint32;
    typedef unsigned long long uint64;

    const static uint32 sha256_k[];
    static const unsigned int SHA224_256_BLOCK_SIZE = (512/8);
public:
    void init();
    void update(const unsigned char *message, unsigned int len);
    void final(unsigned char *digest);
    static const unsigned int DIGEST_SIZE = ( 256 / 8);

protected:
    void transform(const unsigned char *message, unsigned int block_nb);
    unsigned int m_tot_len;
    unsigned int m_len;
    unsigned char m_block[2*SHA224_256_BLOCK_SIZE];
    uint32 m_h[8];
};

std::string sha256(std::string input);

#define SHA2_SHFR(x, n)    (x >> n)
#define SHA2_ROTR(x, n)   ((x >> n) | (x << ((sizeof(x) << 3) - n)))
#define SHA2_ROTL(x, n)   ((x << n) | (x >> ((sizeof(x) << 3) - n)))
#define SHA2_CH(x, y, z)  ((x & y) ^ (~x & z))
#define SHA2_MAJ(x, y, z) ((x & y) ^ (x & z) ^ (y & z))
#define SHA256_F1(x) (SHA2_ROTR(x,  2) ^ SHA2_ROTR(x, 13) ^ SHA2_ROTR(x, 22))
#define SHA256_F2(x) (SHA2_ROTR(x,  6) ^ SHA2_ROTR(x, 11) ^ SHA2_ROTR(x, 25))
#define SHA256_F3(x) (SHA2_ROTR(x,  7) ^ SHA2_ROTR(x, 18) ^ SHA2_SHFR(x,  3))
#define SHA256_F4(x) (SHA2_ROTR(x, 17) ^ SHA2_ROTR(x, 19) ^ SHA2_SHFR(x, 10))
#define SHA2_UNPACK32(x, str)                 \
{                                             \
    *((str) + 3) = (uint8) ((x)      );       \
    *((str) + 2) = (uint8) ((x) >>  8);       \
    *((str) + 1) = (uint8) ((x) >> 16);       \
    *((str) + 0) = (uint8) ((x) >> 24);       \
}
#define SHA2_PACK32(str, x)                   \
{                                             \
    *(x) =   ((uint32) *((str) + 3)      )    \
           | ((uint32) *((str) + 2) <<  8)    \
           | ((uint32) *((str) + 1) << 16)    \
           | ((uint32) *((str) + 0) << 24);   \
}
#endif
```

5.26. transaction.cpp

```
#include "transaction.h"
#include "messages.h"
#define BREAK_LINE '\n'
using namespace std;

// Constructores.

Transaction::Transaction(){
    n_tx_in = 0;
    n_tx_out = 0;
}

Transaction::Transaction(const Transaction & transac){
    input_arr = transac.input_arr;
```

```
    output_arr = transac.output_arr;
    n_tx_in = transac.n_tx_in;
    n_tx_out = transac.n_tx_out;
}

Transaction::Transaction(int tx_in, int tx_out){
    n_tx_in = tx_in;
    n_tx_out = tx_out;
    Array<Input> input_arr(tx_in);
    Array<Output> output_arr(tx_out);
}

Transaction::Transaction(Array<Input> input_array, Array<Output> output_array){
    input_arr = input_array;
    output_arr = output_array;
    n_tx_in = input_arr.getSize();
    n_tx_out = output_arr.getSize();
}

// Destructor.

Transaction::~Transaction(){
}

// Sobrecarga de operadores.

ostream & operator<< (ostream &os, Transaction t){
  os << t.n_tx_in<< endl;
    if(t.n_tx_in){
        os << t.input_arr<< endl;
    }
  os << t.n_tx_out;
  if(t.n_tx_out){
        cout << endl;
        os << t.output_arr;
    }

  return os;
}

bool Transaction::operator==(const Transaction &t){
  return (n_tx_in == t.n_tx_in && n_tx_out== t.n_tx_out && input_arr == t.input_arr &&
    output_arr == t.output_arr);
}

// Métodos de instancia.

unsigned int Transaction::getN_tx_in()const{
    return n_tx_in;
}

unsigned int Transaction::getN_tx_out()const{
    return n_tx_out;
}

Array<Input> Transaction::getInput_arr(){
    return input_arr;
}

Array<Output> Transaction::getOutput_arr(){
    return output_arr;
}

void Transaction::setN_tx_in(unsigned int tx_in){
    n_tx_in = tx_in;
}

void Transaction::setN_tx_out(unsigned int tx_out){
    n_tx_out = tx_out;
```

```
}

void Transaction::setInput_arr(Array<Input> input_array){
    input_arr = input_array;
}

void Transaction::setOutput_arr(Array<Output> output_array){
    output_arr = output_array;
}

// Función que recibe un arreglo con todos los valores levantados del archivo .txt y luego procede a
// crear tanto los inputs como los outputs. Finalmente los agrupa en sus vectores correspondientes
// y procede a cargar los campos en la transaccion.

void Transaction::setValues(Array <string> fvalues){
    Array <Input> v_inputs;
    Array <Output> v_outputs;

    size_t i,j;
    for (i=0; i<(size_t)stoi(fvalues[0]); i++){
      Input input_aux;
      input_aux.setInput(fvalues[i+1]);
      v_inputs.addValueEnd(input_aux);
    }
    for(i=i+1, j=0; j<(size_t)stoi(fvalues[i]);j++){
    Output output_aux;
    output_aux.setOutput(fvalues[j+i+1]);
    v_outputs.addValueEnd(output_aux);
  }
  setN_tx_in(stoi(fvalues[0]));
  setN_tx_out(stoi(fvalues[i]));
  setInput_arr(v_inputs);
  setOutput_arr(v_outputs);
}

string Transaction::toString(){
    string tx_str;
    size_t i;

    tx_str += to_string(getN_tx_in()) + BREAK_LINE;
    for (i = 0; i < getInput_arr().getSize(); i++){
        tx_str += getInput_arr()[i].toString() + BREAK_LINE;
    }
    tx_str += to_string(getN_tx_out()) + BREAK_LINE;
    for (i = 0; i < getOutput_arr().getSize() - 1; i++){

        tx_str += getOutput_arr()[i].toString() + BREAK_LINE;
    }
    tx_str += getOutput_arr()[i].toString();
    return tx_str;
}

string Transaction::getTxHash(){
    return sha256( sha256(((*this).toString() + BREAK_LINE)) );
}
```

5.27. transaction.h

```
#ifndef _TRANSACTION_H_INCLUDED_
#define _TRANSACTION_H_INCLUDED_

#include "array.h"
#include "input.h"
#include "output.h"
#include "sha256.h"

using namespace std;
```

```
10
11 class Transaction{
12     unsigned int n_tx_in;
13     unsigned int n_tx_out;
14     Array<Input> input_arr;
15     Array<Output> output_arr;
16
17 public:
18
19     Transaction();
20     Transaction(const Transaction &);
21     Transaction(int tx_in, int tx_out);
22     Transaction(Array<Input>, Array<Output>);
23     ~Transaction();
24
25     unsigned int getN_tx_in()const;
26     unsigned int getN_tx_out()const;
27     Array<Input> getInput_arr();
28     Array<Output> getOutput_arr();
29
30     string getTxHash();
31
32     void setN_tx_in(unsigned int);
33     void setN_tx_out(unsigned int);
34     void setInput_arr(Array<Input>);
35     void setOutput_arr(Array<Output>);
36     void setValues(Array <string>);
37     string toString();
38     bool operator==(const Transaction &);
39
40     friend ostream & operator<< (ostream &os, Transaction t);
41 };
42
43 #endif
```

5.28. unspent.cpp

```
1  #include "unspent.h"
2  #include "utility.h"
3  #include "messages.h"
4  #include "array.h"
5  #include "outpoint.h"
6
7  using namespace std;
8
9  // Constructores.
10
11 Unspent::Unspent(){
12     value = 0;
13 }
14
15 Unspent::Unspent(float val, Outpoint outpnt){
16     value = val;
17     outpoint = outpnt;
18 }
19
20 Unspent::~Unspent(){
21     value = 0;
22 }
23
24 // Sobrecarga de operadores.
25
26 Unspent &Unspent::operator=(const Unspent &uns){
27     value = uns.value;
28     outpoint = uns.outpoint;
29     return *this;
30 }
31
32 bool Unspent::operator==(const Unspent &unsp){
```

```
    return (value == unsp.value && outpoint == unsp.outpoint);
}


// Getters

float Unspent::getValue(){
    return value;
}

Outpoint Unspent::getOutpoint(){
    return outpoint;
}


//Setters

void Unspent::setValue(float val){
    value = val;
}
void Unspent::setOutpoint (Outpoint outpnt){
  outpoint=outpnt;
}

ostream & operator<< (ostream &os, Unspent unsp){
    os<< unsp.getValue() << DELIM_DATA << unsp.getOutpoint();
    return os;
}
```

5.29. unspent.h

```
#ifndef _UNSPENT_H_INCLUDED_
#define _UNSPENT_H_INCLUDED_


#include <iostream>
#include <string>
#include "outpoint.h"

using namespace std;

class Unspent{
    float value;
    Outpoint outpoint;

public:

    Unspent();
    Unspent(float, Outpoint);
    ~Unspent();

    Unspent &operator=(const Unspent &);
    bool operator==(const Unspent &);

    float getValue();
    Outpoint getOutpoint();

    void setValue (float);
    void setOutpoint(Outpoint outpoint);
    friend ostream & operator<< (ostream &, Unspent);

};

#endif
```

5.30. utility.cpp

```cpp
#include <string>
#include <map>

#include "validation.h"
#include "array.h"
#include "utility.h"
#include "transaction.h"
#include "block.h"
#include "header.h"
#include "body.h"
#include "list.h"
#include "unspent.h"
#include "outpoint.h"
#include "input.h"


// Función que recibe una cadena de caracteres y un delimitador y devuelve un vector donde
// se encuentran las subcadenas obtenidas.

Array<string> splitStr(string str_, char delim){
  Array <string> arr(0);
  string aux;
  istringstream iss;

  iss.str(str_);
  while(getline(iss, aux, delim))
      arr.addValueEnd(aux);
  return arr;
}

//  Función que crea un vector de vectores de strings donde cada vector
//  de strings corresponde a una transacción.

Array<Array<string>> txArr2Vec(Array<string> str_vec){
  Array<Array<string>> str_2vec(0);
  size_t i = 0, j = 0, k;
  // i se utiliza para iterar sobre el vector de strings (parámetro str_vec) que
  // representa lo leído en el archivo de entrada. Cuando el valor de i alcanza el tamaño
  // de str_vec, significa que se recorrió todo el vector. k se utiliza para iterar sobre
  // cada conjunto de inputs y outputs (por separado) representando cuántos inputs y outputs
  // hay por transacción. Por último, j itera sobre el vector de vectores de cadenas, str_2vec
  // representando cuántas transacciones hay.


  while(i < str_vec.getSize()){
    Array<string> empty_str_vec(0);
    str_2vec.addValueEnd(empty_str_vec);

    // Carga de inputs

    for (k = 0; k < (size_t)stoi(str_vec[i]) + 1; k++){
        str_2vec[j].addValueEnd(str_vec[i + k]);
    }
    i += k;

    // Carga de outputs

    for (k = 0; k < (size_t)stoi(str_vec[i]) + 1; k++){
      str_2vec[j].addValueEnd(str_vec[i + k]);
    }
    i += k;
    j++;
  }
  return str_2vec;
}

```

```
// Función que crea el vector de transacciones a partir del vector
// de vectores de strings, o sea, toma los valores y se los asigna
// a los campos de las instancias de Transaction.

Array<Transaction> txArrSet(Array<Array<string>> arr_str_arr){
  Array<Transaction> transaction_vec(0);
  size_t i;

  for (i = 0; i < arr_str_arr.getSize(); i++){
    Transaction transaction;
    transaction.setValues(arr_str_arr[i]);
    transaction_vec.addValueEnd(transaction);
  }
  return transaction_vec;
}


// Función que crea un map representando cuántos bits 0 tiene a la izquierda cada
// uno de los caracteres hexadecimales. Servirá de referencia para validar
// la dificultad pretendida del hash del bloque.

map<char,int> zeroBitsMap(){
  map<char,int> zeros;
  zeros['0'] = 4;
  zeros['1'] = 3;
  zeros['2'] = 2;
  zeros['3'] = 2;
  zeros['4'] = 1;
  zeros['5'] = 1;
  zeros['6'] = 1;
  zeros['7'] = 1;
  zeros['8'] = 0;
  zeros['9'] = 0;
  zeros['a'] = 0;
  zeros['b'] = 0;
  zeros['c'] = 0;
  zeros['d'] = 0;
  zeros['e'] = 0;
  zeros['f'] = 0;
  zeros['A'] = 0;
  zeros['B'] = 0;
  zeros['C'] = 0;
  zeros['D'] = 0;
  zeros['E'] = 0;
  zeros['F'] = 0;
  return zeros;
}

// Función que recibe el fvalues y devuelve un array de array de strings
// donde cada array de strings representa un bloque.

Array<Array<string>> fileToBlock(Array<string> fvalues){
  size_t i = 0, j = 0, k = 0;
  Array<Array<string>> block_vec(0);

  while (i < fvalues.getSize() - 1){
    Array<string> curr_block(0);
    for (j = 0; j < 4; j++){
      curr_block.addValueEnd(fvalues[i+j]);
    }
    i += j;
    for (k = 0; k < (fvalues.getSize() - i) && !(isHash(fvalues[k+i])); k++){
      curr_block.addValueEnd(fvalues[i+k]);
    }
    block_vec.addValueEnd(curr_block);
    i += k;
  }
  return block_vec;
}
```

```
136
137 Block stringToBlock(Array<string> string_block){
138   Block ret_block;
139   Header aux_header;
140   Body aux_body;
141   Array<string> txs_str_vec(0);
142   Array<Array<string>> tx_vec_vec_aux(0);
143   Array<Transaction> tx_array_aux(0);
144
145   aux_header.setPrevBlock(string_block[0]);
146   aux_header.setTxnsHashManually(string_block[1]);
147   aux_header.setBits(stoul(string_block[2]));
148   aux_header.setNonce(stoul(string_block[3]));
149
150   aux_body.setTxn_count(stoul(string_block[4]));
151
152   for (size_t i = 5; i < string_block.getSize(); i++){
153     txs_str_vec.addValueEnd(string_block[i]);
154   }
155
156   tx_vec_vec_aux = txArr2Vec(txs_str_vec);
157   tx_array_aux = txArrSet(tx_vec_vec_aux);
158   aux_body.setTx_arr(tx_array_aux);
159
160   ret_block.setHeader(aux_header);
161   ret_block.setBody(aux_body);
162
163   return ret_block;
164
165 }
166
167 float sumValue(Array<string> cmd_arr){
168   float sumatory = 0;
169   for(size_t i=3; i<cmd_arr.getSize();i+=2)
170     sumatory+=stof(cmd_arr[i]);
171   return sumatory;
172
173 }
174
175 void userUnspent(string hash_user, List <Unspent> *unspent_list, List <Block> * algochain
      ){
176   Node <Unspent> *it_unspent = unspent_list->getFirst();
177   Node <Block> *it_algochain = algochain->getFirst();
178
179   while (it_algochain){
180     Array<Transaction> txn_arr(0);
181     unsigned int txn_count;
182
183     txn_arr = it_algochain->getData().getBody().getTx_arr();
184     txn_count = it_algochain->getData().getBody().getTxn_count();
185     for (size_t i = 0; i < txn_count; i++){
186
187       Array<Output> output_arr(0);
188       unsigned int output_count;
189
190       output_arr = txn_arr[i].getOutput_arr();
191       output_count = txn_arr[i].getN_tx_out();
192
193       for (size_t j = 0; j < output_count; j++){
194
195         if (hash_user == output_arr[j].getAddr()){
196           Outpoint outpoint(txn_arr[i].getTxHash(), j);
197           Unspent new_unspent(output_arr[j].getValue(), outpoint);
198           Node<Unspent> unspent_node(new_unspent);
199           unspent_list->addNodeEnd(unspent_node);
200         }
201       }
202       Array<Input> input_arr(0);
203       unsigned int input_count;
```

```
204
205       input_arr = txn_arr[i].getInput_arr();
206       input_count = txn_arr[i].getN_tx_in();
207       for (size_t j = 0; j < input_count; j++){
208         if (hash_user == input_arr[j].getAddr()){
209           bool flag = false;
210           it_unspent = unspent_list->getFirst();
211
212           while (it_unspent){
213             if (it_unspent->getData().getOutpoint() == input_arr[j].getOutpoint()){
214               it_unspent->remove(unspent_list);
215               it_unspent = unspent_list->getFirst();
216               flag = true;
217             }else
218               it_unspent = it_unspent->getNext();
219           }
220
221           if(!flag)
222             cerr << "No hay una entrada de dinero que justifique esta salida de dinero."
      << endl;
223         }
224       }
225     }
226     it_algochain = it_algochain->getNext();
227   }
228   return;
229 }
230
231
232 float totalUnspent(List<Unspent> *unspent_list, Array<Transaction> *mempool, string
      hash_user){
233   Node <Unspent> *it_unspent = unspent_list->getFirst();
234
235   if((*mempool).getSize()==0)
236     return sumUnspent(unspent_list);
237   for(size_t i= 0; i<(*mempool).getSize();i++)
238   {
239     Array<Output> output_arr(0);
240     unsigned int output_count;
241     output_arr = (*mempool)[i].getOutput_arr();
242     output_count = (*mempool)[i].getN_tx_out();
243     for (size_t j = 0; j < output_count; j++){
244       if (hash_user == output_arr[j].getAddr()){
245         Outpoint outpoint((*mempool)[i].getTxHash(), j);
246         Unspent new_unspent(output_arr[j].getValue(), outpoint);
247         Node<Unspent> unspent_node(new_unspent);
248         unspent_list->addNodeEnd(unspent_node);
249       }
250     }
251     Array<Input> input_arr(0);
252     unsigned int input_count;
253     input_arr = (*mempool)[i].getInput_arr();
254     input_count = (*mempool)[i].getN_tx_in();
255     for (size_t j = 0; j < input_count; j++){
256       if (hash_user == input_arr[j].getAddr()){
257         bool flag = false;
258         it_unspent = unspent_list->getFirst();
259         while (it_unspent){
260           if (it_unspent->getData().getOutpoint() == input_arr[j].getOutpoint()){
261             it_unspent->remove(unspent_list);
262             it_unspent = unspent_list->getFirst();
263             flag = true;
264           }else
265             it_unspent = it_unspent->getNext();
266         }
267         if(!flag)
268           cerr << "No hay una entrada de dinero que justifique esta salida de dinero." <<
      endl;
269       }
```

```
    }
  }
  return sumUnspent(unspent_list);
}


float sumUnspent(List<Unspent> *unspent_list)
{
  float unspent;
  Node <Unspent> *it_unspent(unspent_list->getFirst());
  while (it_unspent){
    unspent += it_unspent->getData().getValue();
    it_unspent = it_unspent->getNext();
  }
  return unspent;
}
```

5.31. utility.h

```
#ifndef UTILITY__H
#define UTILITY__H

#include <iostream>
#include <string>
#include <map>
#include "transaction.h"
#include "array.h"
#include "list.h"
#include "block.h"
#include "unspent.h"

Array<string> splitStr(string str_, char delim);
Array<Array<string>> txArr2Vec(Array<string> str_vec);
Array<Transaction> txArrSet(Array<Array<string>> str_vec_vec);
map<char,int> zeroBitsMap();

Array<Array<string>> fileToBlock(Array<string> fvalues);
Block stringToBlock(Array<string> string_block);
float sumValue(Array<string> cmd_arr);

void userUnspent(string hash_user, List <Unspent> * unspent_list, List <Block> *
    algochain);
float sumUnspent(List<Unspent> *unspent_list);
float totalUnspent(List<Unspent> *unspent_list, Array<Transaction> *mempool, string
    hash_user);
#endif
```

5.32. validation.cpp

```
#include <iostream>

#include "validation.h"
#include "header.h"
#include "messages.h"
#include "block.h"
#include "options.h"
#define HASH_SIZE 64


// Función que valida el formato de las transacciones recibidas.

bool validateTxFormat(Array<string> fvalues, int *amount){
  size_t i = 0, k;

  // i se utiliza para iterar sobre el vector de strings fvalues (que contiene todas las
    líneas de las
  // transacciones recibidas). k se utiliza para iterar sobre cada conjunto de inputs y
    outputs
```

```
18    // (por separado) representando cuántos inputs y outputs hay por transacción.
19
20
21    // La función se ejecuta mientras haya elementos en la posición i del vector.
22
23    while(i < fvalues.getSize()){
24      (*amount)++;
25      // Se verifica que el primer elemento leído de la transacción sea un número (por
        medio de isNumber),
26      // ya que representa el número de inputs que tiene esa transacción. De ser así, se
        itera desde 0
27      // hasta el número de inputs y por cada iteración, se valida el formato de cada input
        , mediante
28      // la función validateInputFormat. De encontrarse algún formato inválido o cantidad
        errónea de inputs,
29      // se retorna false, luego de enviar el correspondiente mensaje de error y se termina
         el programa.
30
31      if(isNumber(fvalues[i])){
32        if((size_t)stoi(fvalues[i]) <= fvalues.getSize() - i -1){
33          for (k = 0; k < (size_t)stoi(fvalues[i]); k++){
34            if (!validateInputFormat(fvalues[i+k+1])==1){
35              return false;
36            }
37          }
38        }
39        else{
40          cerr << ERR_INPUTS_QUANT << endl;
41          return false;
42        }
43      }
44      else{
45        cerr << ERR_FORMAT_QUANT_INPUT << endl;
46        return false;
47      }
48
49        // Se actualiza el valor de i para que quede en la posición siguiente del vector
        fvalues
50      // que representa el número de outputs de la misma transacción.
51
52      i += k + 1;
53
54      // Si i es mayor al tamaño de fvalues, significa que el archivo no tiene más líneas y
         el
55      // formato es erróneo, o sea faltaría la información que representa cuántos outputs
        tiene
56      // la transacción y los mismos outputs.
57
58      if(i > fvalues.getSize() - 1){
59        cerr << ERR_FILE_INCOMPLETE << endl;
60        return false;
61      }
62
63      // Se verifica que el elemento leído de la transacción sea un número (por medio de
        isNumber),
64      // ya que representa el número de outputs que tiene esa transacción. De ser así, se
        itera desde 0
65      // hasta el número de outputs y por cada iteración, se valida el formato de cada
        output, mediante
66      // la función validateOutputFormat. De encontrarse algún formato inválido o cantidad
        errónea de outputs,
67      // se retorna false, luego de enviar el correspondiente mensaje de error y se termina
         el programa.
68
69      if(isNumber(fvalues[i])){
70        if((size_t)stoi(fvalues[i]) <= fvalues.getSize() - i -1){
71          for (k = 0; k < (size_t)stoi(fvalues[i]); k++){
72            if(!validateOutputFormat(fvalues[k+i+1]))
73              return false;
```

```
74          }
75        }else{
76          cerr << ERR_OUTPUTS_QUANT << endl;
77          return false;
78        }
79      }
80      else{
81        cerr << ERR_FORMAT_QUANT_OUTPUT << endl;
82        return false;
83      }
84
85      // Se actualiza la posición de i. Si no hay más transacciones, no se entrará de
        vuelta
86      // al while. Caso contrario, se sigue iterando por cada transacción
87
88      i += k + 1;
89    }
90    return true;
91  }
92
93
94
95
96  bool validateInputFormat(string input){
97
98    // Un input está compuesto por 3 términos, que se declaran en orden debajo:
99
100   string tx_id_hash;  // el hash de la transacción de donde este input toma fondos
101   string idx;     // Int no negativo, índice sobre la secuencia de outputs de la
        transaccion con hash tx id,
102   string addr;    // la dirección de origen de los fondos (debe coincidir con la direcció
        n del output referenciado)
103
104   string delim = TX_DELIM; // delimitador para cada uno de los campos del input
105   size_t pos;          // la posicion del string "input", o sea, el que valido.
106
107   // Se verifica que lleguen 3 términos.
108
109     // el método "find" devuelve la posición de la primera aparición de "delim" o "npos"
        si
110   // "delim" no se encuentra en la cadena.
111
112   if((pos = input.find(delim)) != string::npos){
113     tx_id_hash = input.substr(0, pos); // se guarda como tx_id desde el principio de la
        cadena hasta el 1er "delim"
114     input = input.substr(pos + delim.length()); // se actualiza input, cortando desde "
        pos" hasta el final
115
116     // A continuación, se valida lo asignado a tx_id_hash (debe ser un hash de 64
        caracteres hexadecimales).
117
118     if(!isHexa(tx_id_hash)){
119       cerr << ERR_INPUT_TX_ID_HEXA << endl;
120       return false;
121     }
122     if(tx_id_hash.length() != 64){
123       cerr << ERR_INPUT_TX_ID_LENGTH << endl;
124       return false;
125     }
126   }
127   else{
128     cerr << ERR_INPUT_ONE_ARG <<endl;
129     return false;
130   }
131   if((pos = input.find(delim)) != string::npos){  // se actualiza "pos" hasta el
        siguiente "delim"
132     idx = input.substr(0, pos); // se guarda como idx desde el principio de la cadena
        hasta el "delim"
```

```
133     input = input.substr(pos + delim.length()); // se actualiza input desde el "delim"
        inclusive hasta el final
134     // Se utiliza la función isNumber para validar el formato del campo idx (debe ser un
        número entero positivo).
135     if(!isNumber(idx)){
136       cerr << ERR_INPUT_IDX << endl;
137       return false;
138     }
139   }
140   else{
141     cerr << ERR_INPUT_TWO_ARGS <<endl;
142     return false;
143   }
144
145   // Por último, para que el formato sea correcto, NO se debe encontrar un nuevo "delim"
        (ya que,
146   // de encontrarse, significaría que hay 4 términos o más, lo cual no es un formato vá
        lido de input).
147
148   if((pos = input.find(delim)) == string::npos){
149     addr = input; // aquí, addr es lo que resta de input.
150
151     // Addr se valida de la misma forma que se valida tx_id_hash.
152
153     if(!isHexa(addr)){
154       cerr << ERR_INPUT_ADDR_HEXA << endl;
155       return false;
156     }
157     if(addr.length() != 64){
158       cerr << ERR_INPUT_ADDR_LENGTH << endl;
159       return false;
160     }
161   }
162   else{
163     cerr << ERR_INPUT_TOO_MANY_ARGS <<endl;
164     return false;
165   }
166   return true;
167 }
168
169
170 bool validateOutputFormat(string output){
171   // Un output está compuesto por 2 términos, que se declaran en orden debajo:
172   string value;   // el valor del output (en algocoins), tiene que ser un float positivo.
173   string addr;    // la dirección de destino de los fondos (hash de 64 caracteres
        hexadecimales)
174
175   string delim = TX_DELIM;
176   size_t pos;
177
178   // Para los outputs, se deberá verificar que haya 2 términos, el procedimiento es aná
        logo a la
179   // función validateInputFormat. Teniendo en cuenta que para validar el formato de "
        value" se
180   // utiliza la función isFloat. Addr se valida de la misma manera que en
        validateInputFormat.
181
182   if((pos = output.find(delim)) != string::npos){
183     value = output.substr(0, pos);
184     output = output.substr(pos + delim.length());
185     if(!isFloat(value)){
186       cerr << ERR_OUTPUT_VALUE << endl;
187       return false;
188     }
189   }
190   else{
191     cerr << ERR_OUTPUT_ONE_ARG << endl;
192     return false;
193   }
```

```
194
195   if((pos = output.find(delim)) == string::npos){
196     addr = output;
197     if(!isHexa(addr)){
198       cerr << ERR_OUTPUT_ADDR_HEXA << endl;
199       return false;
200     }
201     if(addr.length() != 64){
202       cerr << ERR_OUTPUT_ADDR_LENGTH << endl;
203       return false;
204     }
205   }
206   else{
207     cerr << ERR_OUTPUT_TO_MANY_ARGS << endl;
208     return false;
209   }
210   return true;
211
212 }
213
214
215
216 bool validateBlockFormat(Array<string> block)
217 {
218   int n_tx;
219   Array <string> tx_aux;
220
221   if(!isHash(block[0]) || !isHash(block[1]) || !isNumber(block[2]) || !isNumber(block[3])
        ||!isNumber(block[4]) )
222     return false;
223
224   for(size_t i=5; i<block.getSize();i++)
225     tx_aux.addValueEnd(block[i]);
226   //cout<<tx_aux;
227   if(!validateTxFormat(tx_aux,&n_tx) || n_tx != stoi(block[4]))
228     return false;
229   return true;
230 }
231
232 // Devuelve true si el string es un int válido
233
234 bool isNumber(const string & s){
235     string::const_iterator it = s.begin();
236     while (it != s.end() && isdigit(*it))
237       ++it;
238     return (!s.empty() && it == s.end());
239 }
240
241 // Devuelve true si el string es un hexadecimal válido
242
243 bool isHexa(const string & s){
244     string::const_iterator it = s.begin();
245     while (it != s.end() && isxdigit(*it))
246     ++it;
247     return (!s.empty() && it == s.end());
248 }
249
250 bool isHash(string str){
251   return isHexa(str) && str.length() == HASH_SIZE;
252 }
253
254 // Devuelve true si el string es un float válido
255
256 bool isFloat(const string& s){
257     string::const_iterator it = s.begin();
258     bool decimalPoint = false;
259     long unsigned int minSize = 0;
260     while(it != s.end()){
261     if(*it == '.'){
```

```
262     if(!decimalPoint)
263       decimalPoint = true;
264     else break;
265     }else if(!isdigit(*it)){
266     break;
267     }
268     ++it;
269     }
270     return s.size() > minSize && it == s.end();
271 }
272 
273 // Función que recibe el hash, el map contenedor zero_bits, la dificultad pretendida y
274 // retorna true cuando el hash cumple con la dificultad.
275 
276 bool validateDifficulty(string hash, map<char,int> zero_bits, int diff){
277   string::const_iterator it = hash.begin();
278   bool isValid = false;
279 
280   // Se itera sobre el hash mientras el string tenga caracteres y mientras isValid sea
       False.
281   // Se lee el primer caracter hexadecimal del hash y, por medio de la función zero_bits,
       se verifica
282   // cuántos bits cero contiene a la izquierda dicho número hexadecimal.
283 
284   // Si dicha cantidad de ceros supera la dificultad pretendida, entonces la dificultad
       queda verificada,
285   // isValid es True y no se re-ingresa al while.
286 
287   // Si dicha cantidad de ceros, en cambio, NO supera la dificultad pretendida,
288   // se valida si dicho número es distinto de 0, ya que de ser así, la dificultad no
       queda validada.
289 
290   // Si dicha cantidad de ceros NO supera la dificultad pretendida y ES un 0, se itera
       nuevamente sobre
291   // el hash para validar el siguiente caracter.
292 
293   while (it != hash.end() && !isValid){
294     if(zero_bits[(*it)] >= (int)diff){
295       isValid = true;
296     }
297     else if((*it) != '0'){
298       break;
299     }
300     diff = diff - zero_bits[(*it)];
301     ++it;
302   }
303   return isValid;
304 }
305 
306 bool validateInitFormat(Array<string> cmd_arr){
307 
308     if(cmd_arr.getSize() != 4){
309         return false;
310     }
311 
312 
313     if(!validateUserFormat(cmd_arr[1])){
314         return false;
315     }
316 
317     if(!validateValueFormat(cmd_arr[2])){
318         return false;
319     }
320 
321     if(!validateBitsFormat(cmd_arr[3])){
322         return false;
323     }
324 
325     return true;
```

```
}

bool validateUserFormat(string user){
    size_t pos;
    string delim = TX_DELIM;

    if(user.empty()){
        return false;
    }

    // el método "find" devuelve la posición de la primera aparición de "delim" o "npos" si
    // "delim" no se encuentra en la cadena.
    if((pos = user.find(delim)) != string::npos){
        return false;
    }

    return true;
}

bool validateValueFormat(string value){

    if(value.empty()){
        return false;
    }
    if(!isFloat(value)){
        return false;
    }

    return true;
}

bool validateBitsFormat(string bits){

    if(bits.empty()){
        return false;
    }
    if(!isNumber(bits)){
        return false;
    }

    return true;
}


bool validateSaveFormat(Array <string> cmd_arr)
{
  if (cmd_arr.getSize()!= 2)
    return false;
  else return true;
}

// La función transfer aceptará cualquier combinación de caracteres como user (source).
bool validateTransferFormat(Array<string> cmd_arr){
  if(cmd_arr[1].empty() || cmd_arr.getSize() < 3 || cmd_arr.getSize() % 2 != 0)
    return false;
  for(size_t i = 2; i<cmd_arr.getSize();i+=2){
    if(cmd_arr[i].empty() || !isFloat((cmd_arr[i+1])))
      return false;
  }
  return true;
}



bool validateMineFormat(Array<string> cmd_arr){
  return cmd_arr.getSize()==2 && isNumber(cmd_arr[1]);
}

```

```
394 bool validateBalanceFormat(Array<string> cmd_arr){
395   return cmd_arr.getSize()==2;
396 }
397
398 bool validateTxnFormat(Array<string> cmd_arr){
399   return isHash(cmd_arr[1]) && cmd_arr.getSize() == 2;
400 }
401
402 bool validateLoadFormat(Array<string> cmd_arr){
403   return (cmd_arr.getSize()==2);
404 }
405
406 bool validateOptBlockFormat(Array<string> cmd_arr){
407   return isHash(cmd_arr[1]) && cmd_arr.getSize() == 2;
408 }
409
410 bool validateLoadedAlgochain(Array<Block> block_array){
411   size_t bk_it;
412
413   if(!validateGenesis(block_array[0]))
414     return false;
415
416   for (bk_it = 0; bk_it < block_array.getSize(); bk_it++){
417     if(!(block_array[bk_it].getHeader().validateHeaderDifficulty())){
418       cerr << "The " << bk_it << " block does not verifies the given difficulty." << endl
      ;
419       return false;
420     }
421     if(bk_it > 0){
422       if(!(block_array[bk_it].getHeader().getPrev_block() == block_array[bk_it - 1].
      getBlockHash())){
423         cerr << "The prev_block of the block " << bk_it << " doesn't match the hash of
      the block " << bk_it - 1 << endl;
424         return false;
425       }
426     }
427     if(!(block_array[bk_it].getHeader().getTxns_hash() == block_array[bk_it].
      getMerkleHash())){
428       cerr << "The txn_hash of the block's header " << bk_it << " doesn't match the Mekle
       hash of it's transactions." << endl;
429       return false;
430     }
431     if(!validateDoubleSpending(block_array, bk_it)){
432       return false;
433     }
434     if(bk_it > 0){
435       if(!validateTxnFunds(block_array, bk_it)){
436         return false;
437       }
438     }
439   }
440
441   return true;
442 }
443
444 bool validateDoubleSpending(Array<Block> block_array, size_t bk_it){
445   size_t txn_it, bk_it2, txn_it2, out_it, inp_it;
446
447
448   for (txn_it = 0; txn_it < block_array[bk_it].getBody().getTx_arr().getSize(); txn_it++)
      {
449     for(out_it = 0; out_it < block_array[bk_it].getBody().getTx_arr()[txn_it].
      getOutput_arr().getSize(); out_it++){
450       Outpoint outpoint_ref(block_array[bk_it].getBody().getTx_arr()[txn_it].getTxHash(),
       out_it);
451       int output_appears = 0;
452
453       txn_it2 = txn_it + 1;
```

```
454       for ( ; txn_it2 < block_array[bk_it].getBody().getTx_arr().getSize() &&
      output_appears < 2; txn_it2++){
455         for(inp_it = 0; inp_it < block_array[bk_it].getBody().getTx_arr()[txn_it2].
      getInput_arr().getSize(); inp_it++){
456           if(block_array[bk_it].getBody().getTx_arr()[txn_it2].getInput_arr()[inp_it].
      getOutpoint() == outpoint_ref){
457             output_appears += 1;
458           }
459         }
460       }
461       if(output_appears >= 2){
462         cerr << "There's double spending of the output " << out_it << " the block " <<
      bk_it << endl;
463         return false;
464       }
465       bk_it2 = bk_it + 1;
466       for( ; bk_it2 < block_array.getSize(); bk_it2++){
467         for (txn_it2 = 0 ; txn_it2 < block_array[bk_it2].getBody().getTx_arr().getSize()
      && output_appears < 2; txn_it2++){
468           for(inp_it = 0; inp_it < block_array[bk_it2].getBody().getTx_arr()[txn_it2].
      getInput_arr().getSize(); inp_it++){
469             if(block_array[bk_it2].getBody().getTx_arr()[txn_it2].getInput_arr()[inp_it].
      getOutpoint() == outpoint_ref){
470               output_appears += 1;
471             }
472           }
473         }
474       }
475       if(output_appears >= 2){
476         cerr << "There's double spending of the output " << out_it << " the block " <<
      bk_it << endl;
477         return false;
478       }
479     }
480   }
481   return true;
482
483 }
484
485 bool validateGenesis(Block block){
486   if(!(block.getHeader().getPrev_block() == NULL_HASH)){
487     cerr << "The first block is not a valid genesis block. The prev block of the genesis
      block is not null." << endl;
488     return false;
489   }
490   if(!(block.getBody().getTxn_count() == 1)){
491     cerr << "The first block is not a valid genesis block. The genesis block must have
      one and only one transaction." << endl;
492     return false;
493   }
494   if(!(block.getBody().getTx_arr()[0].getN_tx_in() == 1)){
495     cerr << "The first block is not a valid genesis block. The genesis block must have
      one and only one input." << endl;
496     return false;
497   }
498   if(!(block.getBody().getTx_arr()[0].getN_tx_out() == 1)){
499     cerr << "The first block is not a valid genesis block. The genesis block must have
      one and only one output." << endl;
500     return false;
501   }
502   Input input = block.getBody().getTx_arr()[0].getInput_arr()[0];
503
504   if(!(input.getOutpoint().getTx_id() == NULL_HASH)){
505     cerr << "The first block is not a valid genesis block. The tx_id of the genesis'
      input must be a null hash." << endl;
506     return false;
507   }
508   if(!(input.getOutpoint().getIdx() == 0)){
```

```
509       cerr << "The first block is not a valid genesis block. The idx of the genesis' input
          must be 0." << endl;
510       return false;
511     }
512     if(!(input.getAddr() == NULL_HASH)){
513       cerr << "The first block is not a valid genesis block. The addr of the genesis' input
           must be a null hash." << endl;
514       return false;
515     }
516     return true;
517   }
518
519   bool validateTxnFunds(Array<Block> block_array, size_t bk_it){
520     size_t txn_it, inp_it, bk_it2, out_it, txn_it2;
521
522     for(txn_it = 0; txn_it < block_array[bk_it].getBody().getTx_arr().getSize(); txn_it++){
523       float input_funds = 0;
524       float output_funds = 0;
525
526       for(inp_it = 0; inp_it < block_array[bk_it].getBody().getTx_arr()[txn_it].
          getInput_arr().getSize(); inp_it++){
527         Outpoint outpoint_ref = block_array[bk_it].getBody().getTx_arr()[txn_it].
          getInput_arr()[inp_it].getOutpoint();
528         string addr_ref = block_array[bk_it].getBody().getTx_arr()[txn_it].getInput_arr()[
          inp_it].getAddr();
529         for(bk_it2 = 0; bk_it2 <= bk_it; bk_it2++){
530           for(txn_it2 = 0; txn_it2 < block_array[bk_it2].getBody().getTx_arr().getSize();
          txn_it2++){
531             if(bk_it2 == bk_it && txn_it2 == txn_it){
532               cerr << "Funds of the input not found" << endl;
533               return false;
534             }
535             if(outpoint_ref.getTx_id() == block_array[bk_it2].getBody().getTx_arr()[txn_it2
          ].getTxHash()){
536               if(block_array[bk_it2].getBody().getTx_arr()[txn_it2].getOutput_arr()[
          outpoint_ref.getIdx()].getAddr() == addr_ref){
537                 input_funds += block_array[bk_it2].getBody().getTx_arr()[txn_it2].
          getOutput_arr()[outpoint_ref.getIdx()].getValue();
538               }else
539               {
540                 cerr << "The addrs of the input and it's referenced output don't match" <<
          endl;
541                 return false;
542               }
543               bk_it2 = bk_it + 1;
544               break;
545             }
546           }
547         }
548       }
549       for(out_it = 0; out_it < block_array[bk_it].getBody().getTx_arr()[txn_it].
          getOutput_arr().getSize(); out_it++){
550         output_funds += block_array[bk_it].getBody().getTx_arr()[txn_it].getOutput_arr()[
          out_it].getValue();
551       }
552       if(!(input_funds == output_funds)){
553         cerr << "The funds in the inputs don't match the funds in the outputs of the
          transaction " << txn_it << " (block " << bk_it << ")" << endl;
554         return false;
555       }
556     }
557     return true;
558   }
```

5.33. validation.h

```
1  #ifndef _VALIDATION_H_INCLUDED_
2  #define _VALIDATION_H_INCLUDED_
3
```

```
#include "array.h"
#include "block.h"
#include <string>
#include <map>

#define TX_DELIM " "

using namespace std;

bool validateTxFormat(Array<string> fvalues, int *amount);
bool validateInputFormat(string input);
bool validateOutputFormat(string output);
bool validateBlockFormat(Array<string> block);


bool isNumber(const string & s);
bool isHexa(const string & s);
bool isFloat(const string& s);
bool isHash(string str);

map<char,int> zeroBitsMap();
bool validateDifficulty(string, map<char,int>, int);
bool validateInitFormat(Array<string> cmd_arr);
bool validateUserFormat(string user);
bool validateValueFormat(string value);
bool validateBitsFormat(string bits);
bool validateSaveFormat(Array <string> cmd_arr);
bool validateTransferFormat(Array<string> cmd_arr);
bool validateMineFormat(Array<string> cmd_arr);
bool validateBalanceFormat(Array<string> cmd_arr);
bool validateTxnFormat(Array<string> cmd_arr);
bool validateLoadFormat(Array<string> cmd_arr);
bool validateOptBlockFormat(Array<string> cmd_arr);
bool validateLoadedAlgochain(Array<Block> block_array);
bool validateGenesis(Block block);
bool validateDoubleSpending(Array<Block> block_array, size_t bk_it);
bool validateTxnFunds(Array<Block> block_array, size_t bk_it);

#endif
```