



ALGORITMOS Y PROGRAMACIÓN II (75.04/95.12)

---

## TRABAJO PRÁCTICO N° 1

---

### Alumnos:

Galván, Sergio Daniel	sdgalvan@fi.uba.ar	#51290
Vera Guzmán, Ramiro	rverag@fi.uba.ar	#95887
Dreszman, Alan	adreszman@fi.uba.ar	#92351

Fecha de entrega: Jueves 03/12/2020

---

---

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Diseño</b>	<b>2</b>
<b>3. Implementación</b>	<b>2</b>
3.1. UML	3
<b>4. Clases</b>	<b>4</b>
4.1. Clase BlockchainBookkeeper	4
4.1.1. Acciones por comando para BlockchainBookkeeper	4
4.2. Clase BlockchainHistoryBook	8
4.2.1. Comandos en los que participa BlockchainHistoryBook	8
4.3. Clase Mempool	12
4.3.1. Acciones por comando para Mempool	12
4.4. Clase Queue	14
4.5. Clase Cuentas	16
4.6. Clase BlockchainManager	24
4.7. Clase BlockchainFileManager	31
4.8. Clase BlockchainBuilder	51
4.9. Clase Block	57
4.10. Clase Transaction	61
4.11. Clase TransactionInput	64
4.12. Clase TransactionOutput	66
4.13. Clases sha256 cmdlime	67
<b>5. Main</b>	<b>67</b>
<b>6. Compilación</b>	<b>69</b>
<b>7. Pruebas sobre el programa</b>	<b>71</b>
7.1. Pruebas de fuga de memoria con Vlagrind	71
7.2. Pruebas de errores forzados	73
7.3. Pruebas de funcionamiento normal	74
<b>8. Conclusión</b>	<b>75</b>
<b>9. Anexo I</b>	<b>75</b>
9.1. Enunciado	75

## 1. Introducción

En el presente informe se detalla el trabajo práctico que tiene como objetivo el diseño e implementación de un programa en C++, con el cual se busca ejercitar los conceptos vistos en la materia. El programa será una implementación de *Blockchain* (bajo el pseudónimo *Algochain*). El programa está preparado para recibir, al ejecutarse, argumentos por la línea de comandos. A los mismos le siguen ciertas acciones que tienen que ver con la creación, modificación, carga y consulta de un sistema de *Blockchain*. El flujo de consecuencias de cada comando se desarrolla en la descripción de las clases asociadas con los cada uno de ellos. El desarrollo presentado se ha realizado como complemento del presentado en el **TP0**, en el cual se implemento la funcionalidad de armar un *Bloque* (unidad básica de la *Blockchain*) a partir de (un archivo de texto con) una transacción. Dicha funcionalidad se mantiene como parte del engranaje, pero resulta invisible durante la ejecución del actual programa.

**COMENTARIO SOBRE EL TRABAJO:** Debido a la falta de tiempo y a la gran complejidad del mismo, el programa no tiene implementados los métodos de **balance**. Esto se debe a que no se completo la integración de la clase Cuentas. La idea de la misma era llevar un registro intermedio de los usuarios así como sus saldos confirmados y no confirmados. Esto permitía validar rápidamente al usar balance, además de generar la línea extra de vueltos durante la creación de una transacción con el comando transfer. Sin embargo, el presente informe completa la mayoría de los requerimientos exigidos por la cátedra en cuanto a diseño, robustez, encapsulamiento y programación orientada a objetos.

## 2. Diseño

Dado que no se trata de un programa que siga una única secuencia de pasos (es decir existen varias secuencias independientes entre sí con distintos caminos y distintos resultados) la lógica del mismo está distribuida en varias de sus clases. Dentro del paradigma OOP, se ha optado por un acercamiento que se propone una modularización de las funcionalidades principales del programa y de sus procesos adyacentes o de soporte. Particularmente se observará (más en detalle en la sección correspondiente) que la lógica central del programa se encuentra en las clases llamadas **BlockChainBuilder**, **BlockChainFileManager** y **BlockChainBookkeeper**, mientras que las demás clases realizan tareas de almacenamiento y traslado de la información, entre otras.

## 3. Implementación

Un flujo general del programa sigue la siguiente lógica: Cabe mencionar que la ejecución del programa depende de como este es invocado en la consola. Por un lado, si se le especifica la lista de comandos desde un archivo el programa entenderá cada línea como una entrada del usuario y ejecutara en pos a ello, siguiendo una lectura secuencial. Puede resumirse la ejecución de la misma en las tareas descriptas a continuación, entendiéndose IDLE, como el estado previo a la lectura del comando.

1. Inicio, estado IDLE (a la espera de la introducción de algún comando)
2. Introducción de algún comando
3. Validación de argumentos
4. Parseo de la información pasada en los argumentos
5. Interpretación de la información parseada (lo que determina la acción a ejecutar)
6. Validación de la información parseada
7. Ejecución de la acción interpretada
8. Vuelta al estado IDLE

### 3.1. UML

A continuación, se esquematiza un diagrama UML de clases para observar gráficamente la jerarquización y las clases implementadas.

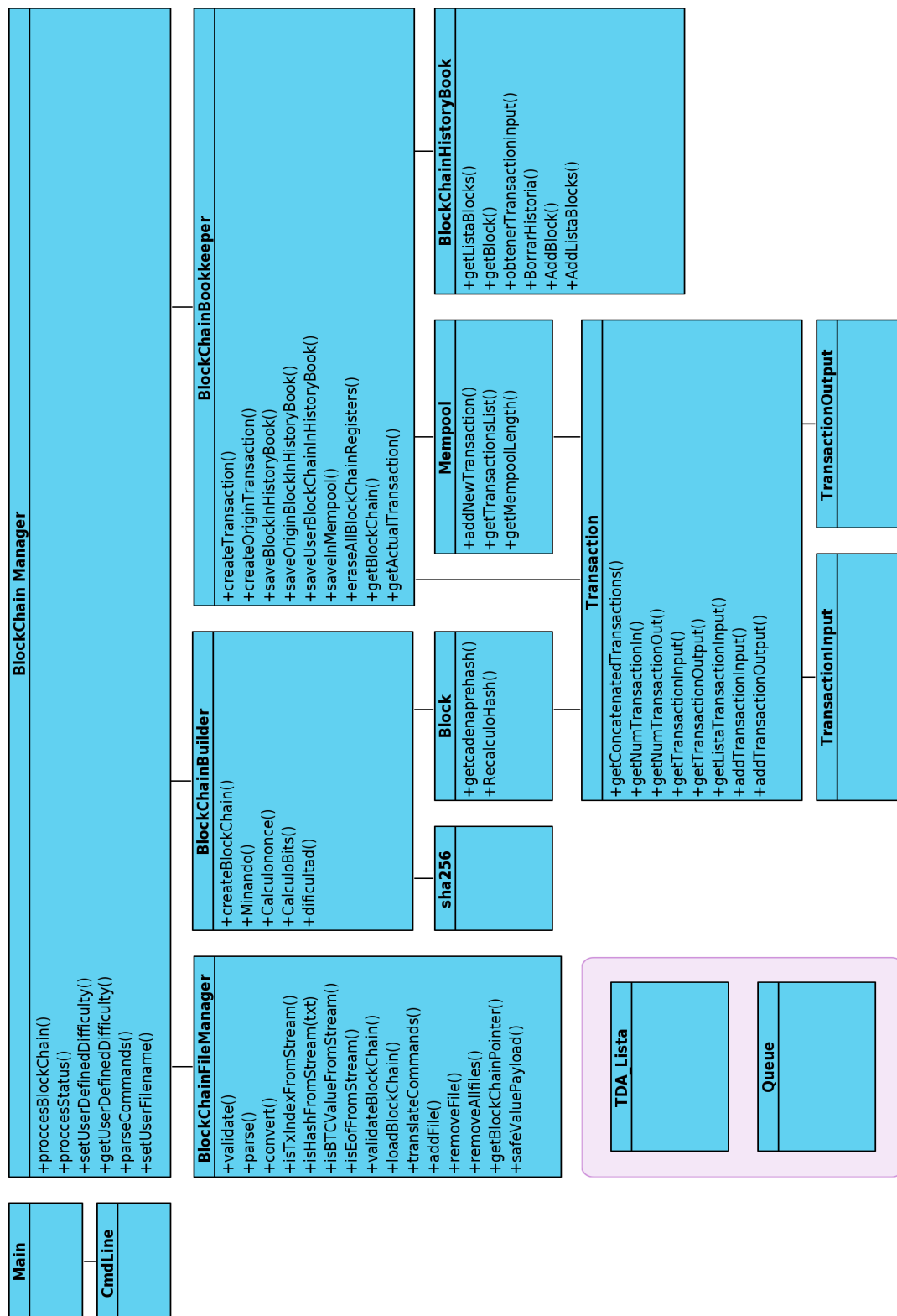


Figura 3.1: Diagrama de Clases. Obs: Se obviaron métodos constructores, destructores, *getters* y *setters*

## 4. Clases

### 4.1. Clase BlockchainBookkeeper

La clase **BlockchainBookkeeper** es una interfaz entre los comandos que requieren alguna interacción con la Mempool (almacenamiento temporal de transacciones que no han sido minadas) y/o con **BlockchainHistoryBook** (historial de transacciones, es decir, la mismísima Blockchain). La clase Bookkeeper viene del ingles contador. La idea es que que esta clase sepa de la historia contable y de las transacciones no confirmadas y que pueda operar con ellas y guardar de ser necesario. Se mantuvo la privacidad de las clases BlockchainHistoryBook y la clase Mempool poniendo todos sus metodos como privados y estáticos y declarando a la clase Bookkeeper como *friend*. Esta idea refuerza el concepto de encapsulamiento dado que ambas clases no deberían ser accedidas por ninguna otra clase, sin que el bookkeeper se entere.

#### 4.1.1. Acciones por comando para BlockchainBookkeeper

A continuación se detalla como interviene la clase Bookkeeper en la ejecución de los distintos comandos.

##### **init**

La clase interviene utilizando los datos correspondientes para crear el bloque de origen. Los datos previamente parseados por **BlockchainFileManager** son utilizados en los métodos **createOriginTransaction()** y **saveOriginBlockInHistoryBook()** para tal fin.

##### **transfer**

La clase interviene utilizando los datos correspondientes para grabar una transferencia en la Mempool. Los datos previamente parseados por **BlockchainFileManager** son utilizados en el método **saveInMempool()** para tal fin.

**mine** En el minado, esta clase extrae la información almacenada en Mempool y la dispone a BlockchainBuilder para el posterior ensamblaje de un bloque.

**balance** Esta clase consulta en Mempool las transacciones vinculadas con el usuario en cuestión. Se utiliza el método **getBlockChain()**.

**block** Esta clase busca en la historia BlockchainHistoryBook el hash id del bloque en cuestion, si lo encuentra lo imprime sino, devuelve un estado de hash no encontrado.

##### **txn**

**load** Para el comando load se instancia a la clase FileManager principalmente para abrir el archivo en tiempo de ejecucion y una vez, validada y cargada esta se la da al bookkeeper para que la guarde en la historia oficial (HistoryBook) como la nueva blockchain a consultar

**save** En este comando se instancia principalmente a FileManager para abrir el archivo en tiempo de ejecucion. Una vez abierto el bookkeeper accedera a la historia oficial para generar una copia y darsela al FileManager para que la convierta a txt.

```
1
2 #include "BlockchainBookkeeper.h"
3
4 BlockchainBookkeeper::BlockchainBookkeeper() {
5     this->ActualTransaction = NULL;
6 }
7
8 BlockchainBookkeeper::~BlockchainBookkeeper() {
9     if(this->ActualTransaction !=NULL)
10         delete ActualTransaction;
11
12     if ( ! this->TransactionList.vacia() ) {
13         lista <Transaction *>::iterador it( this->TransactionList );
14         it = this->TransactionList.primer();
15         while ( ! this->TransactionList.isEmpty() ) {
16             delete it.dato();
17             this->TransactionList.eliminar_nodo(it);
18         }
19     }
20     if ( ! this->BlockList.vacia() ) {
```

```

21         lista <Block *>::iterador it( this->BlockList );
22         it = this->BlockList.primer();
23         while ( ! this->BlockList.isEmpty() ) {
24             delete it.dato();
25             this->BlockList.eliminar_nodo(it);
26         }
27     }
28 }
29
30 status_t BlockChainBookkeeper::createOriginTransaction(payload_t & payload){
31     this->ActualTransaction = new Transaction(0,1);
32     this->ActualTransaction->getTransactionOutput(1)->setAddr(sha256(sha256(payload.user)));
33     this->ActualTransaction->getTransactionOutput(1)->setValue(payload.value);
34     return STATUS_FINISH_CONVERT_SUCCESSFULLY;
35 }
36
37 Transaction * & BlockChainBookkeeper::getActualTransaction(void){
38     return this->ActualTransaction;
39 }
40
41 Block * & BlockChainBookkeeper::getActualBlock(void){
42     return this->ActualBlock;
43 }
44
45 status_t BlockChainBookkeeper::saveOriginBlockInHistoryBook(Block *& block){
46     if (! BlockChainHistoryBook::AlgoChain.isEmpty() ) BlockChainHistoryBook::BorrarHistoria()
47     ;
48     return saveBlockInHistoryBook(block);
49 }
50
51 status_t BlockChainBookkeeper::saveBlockInHistoryBook(Block* &block){
52     //TODO falta actualiza lista de usuarios
53     if (BlockChainHistoryBook::AddBlock(block) ) return STATUS_OK;
54     else return STATUS_BAD_ALLOC;
55 }
56
57 status_t BlockChainBookkeeper::saveUserBlockChainInHistoryBook(lista<Block*> &listaBlock){
58     if (! BlockChainHistoryBook::AlgoChain.isEmpty() ) BlockChainHistoryBook::BorrarHistoria()
59     ;
60     lista<Block*>::iterador it(listaBlock);
61     it = listaBlock.ultimo();
62     while(!it.extremo()){
63         if (! BlockChainHistoryBook::AddBlock(it.dato() ) ) return STATUS_BAD_ALLOC;
64         it.retroceder();
65     }
66     return STATUS_OK;
67 }
68
69 status_t BlockChainBookkeeper::createTransaction(payload_t payload){
70     std::string _user_ = payload.ArgTranfer->dequeue();
71     const string hashUser= sha256(sha256(_user_));
72
73     //TODO Buscar en la lista de usuario a ver si tiene saldo
74
75     // Busco en la historia la transaccion asociado al usuario pasado por hash
76     Transaction * tr = BlockChainHistoryBook::getTransactionByTransactionOutputUser(hashUser);
77     if(tr == NULL){
78         tr = Mempool::getTransactionsFromMempool(hashUser);
79         if(tr == NULL) return STATUS_ERROR_HASH_NOT_FOUND;
80     }
81
82     // Mirando como es la estructura de la transaccion completo el outpoint
83
84     // Contando con txIn encuentro el valor de indice del outpoint
85     unsigned int txIn = 0;
86     lista <TransactionOutput *> tOutput;
87     tOutput =tr->getTransactionOutputList();
88     lista <TransactionOutput *>::iterador itTransOutput( tOutput);

```

```

89     itTransOutput = tOutput.primer();
90     do {
91         if ( hashUser.compare(itTransOutput.dato()->getAddr()) == 0 ) {
92             break;
93         }
94         txIn++;
95         itTransOutput.avanzar();
96     } while ( ! itTransOutput.extremo() );
97
98     // Con el doble hash de la transaccion obtengo el valor de Txid
99     std::string Txid = sha256(sha256(tr->getConcatenatedTransactions()));
100
101     this->ActualTransaction = new Transaction();
102     this->ActualTransaction->addTransactionInput();
103     this->ActualTransaction->getTransactionInput(1)->setTxId(Txid);
104     this->ActualTransaction->getTransactionInput(1)->setIdx(txIn);
105     this->ActualTransaction->getTransactionInput(1)->setAddr(hashUser);
106
107     unsigned int OutputNumber = 1 ;
108     while( ! payload.ArgTranfer->isEmpty()){
109         this->ActualTransaction->addTransactionOutput();
110         this->ActualTransaction->getTransactionOutput(OutputNumber)->setAddr(sha256(sha256(
111         payload.ArgTranfer->dequeue())));
112         this->ActualTransaction->getTransactionOutput(OutputNumber)->setValue(std::stof(
113         payload.ArgTranfer->dequeue()));
114         OutputNumber++;
115     }
116
117     // @TODO falta agregar como output el vuelto
118     return STATUS_OK;
119 }
120
121 lista<Transaction *> & BlockchainBookkeeper::getMempool(){
122     if (Mempool::getMempoolLength()){
123         lista<Transaction *> mempool = Mempool::getTransactionsList();
124         lista<Transaction *> ::iterador itMempool(mempool);
125         while(! itMempool.extremo()){
126             Transaction actualTran;
127             actualTran = *(itMempool.dato());
128             Transaction * copyTrans = new Transaction(actualTran);
129             this->TransactionList.insertar(copyTrans) ;
130             itMempool.avanzar();
131         }
132         Mempool::BorrarMempool();
133     }else{
134         this->ActualTransaction = new Transaction(0,0);
135         this->TransactionList.insertar(this->ActualTransaction );
136     }
137     return this->TransactionList;
138 }
139
140 status_t BlockchainBookkeeper::saveInMempool(Transaction * trans){
141     // @TODO falta actualizar lista de usuarios
142     Mempool::addNewTransaction(trans);
143     return STATUS_OK;
144 }
145
146 status_t BlockchainBookkeeper::searchInHistoryBook(HashIdType type, std::string hashId){
147     switch(type){
148     case HashIdType::blockId:{
149         Block * B = BlockchainHistoryBook::searchBlock(hashId);
150         if(B == NULL) return STATUS_ERROR_HASH_NOT_FOUND;
151         Block * newBlock = new Block(*B);
152         this->BlockList.insertar(newBlock);
153         return STATUS_OK;
154     }
155     break;
156     }

```

```

157     case HashIdType::txnId:{
158         const lista <Transaction *> * T = BlockchainHistoryBook::searchTransaction(hashId);
159         if(T == NULL) return STATUS_ERROR_HASH_NOT_FOUND;
160         lista <Transaction *>::iterador it(*T);
161         while(!it.extremo()){
162             Transaction * newTrans = new Transaction(*(it.dato()));
163             this->TransactionList.insertar(newTrans) ;
164             it.avanzar();
165         }
166         return STATUS_OK;
167         break;
168     }
169 }
170 return STATUS_ERROR_HASH_NOT_FOUND;
171 }
172
173
174 const lista<Block *> & BlockchainBookkeeper::getBlockChain(void){
175     return BlockchainHistoryBook::getListBlocks();
176 }
177
178 lista<Block *> & BlockchainBookkeeper::getBlockList(void){
179     return this->BlockList;
180 }
181
182 lista<Transaction *> & BlockchainBookkeeper::getTransactionList(void){
183     return this->TransactionList;
184 }
185
186 status_t BlockchainBookkeeper::eraseAllBlockchainRegisters(void){
187     BlockchainHistoryBook::BorrarHistoria();
188     Mempool::BorrarMempool();
189     return STATUS_OK;
190 }
191
192 std::string BlockchainBookkeeper::getLastBlockHash(void){
193     lista<Block *>::iterador AlgoChain( BlockchainHistoryBook::AlgoChain);
194     AlgoChain = BlockchainHistoryBook::AlgoChain.ultimo();
195     return AlgoChain.dato()->getBlockHash();
196 }
197
198 std::string BlockchainBookkeeper::getTransactionHash(){
199     string debugString = this->ActualTransaction->getConcatenatedTransactions();
200     return sha256(sha256(debugString));
201 }

```

```

1
2 #ifndef BLOCKCHAINBOOKKEEPER_H_
3 #define BLOCKCHAINBOOKKEEPER_H_
4
5 #include "BlockchainHistoryBook.h"
6 #include "BlockchainDataTypes.h"
7 #include "BlockchainStatus.h"
8 #include "Mempool.h"
9 #include "Transaction.h"
10 #include "Block.h"
11
12 #include "sha256.h"
13
14 // #include "lista.h"
15
16 class BlockchainBookkeeper {
17
18 private:
19     Block * ActualBlock;
20     Transaction * ActualTransaction;
21     lista<Block * > BlockList;
22     lista<Transaction *> TransactionList;

```



```

23
24 public:
25     BlockchainBookkeeper();
26     ~BlockchainBookkeeper();
27
28     status_t createTransaction(payload_t payload);
29     status_t createOriginTransaction(payload_t & payload);
30     status_t saveBlockInHistoryBook(Block * &block);
31     status_t saveOriginBlockInHistoryBook(Block* &block);
32     status_t saveUserBlockchainInHistoryBook(lista<Block*> &listaBlock);
33     status_t saveInMempool(Transaction * trans);
34     status_t eraseAllBlockchainRegisters(void);
35     status_t searchInHistoryBook(HashIdType type, std::string hashId);
36
37     std::string getLastBlockHash(void);
38     std::string getTransactionHash(void);
39     const lista<Block *> & getBlockchain(void);
40     Block * &getActualBlock(void);
41     Transaction * &getActualTransaction(void);
42     lista<Block *> &getBlockList(void);
43     lista<Transaction *> &getTransactionList(void);
44     lista<Transaction *> & getMempool(void );
45
46 };
47
48
49 #endif /* BLOCKCHAINBOOKKEEPER_H_ */

```

## 4.2. Clase BlockchainHistoryBook

Es el registro dinámico de toda la cadena de bloques, es decir, la *Blockchain* en si misma se encuentra alojada en su atributo **AlgoChain**. AlgoChain es entonces una lista de instancias de **Block**. Puesto que no existen diversas AlgoChains en el programa se decidió que la clase **BlockchainHistoryBook** sea una clase **estática**, es decir, al igual que Mempool o BlockchainManager son clases que no pueden instanciarse. A diferencia de Manager, y en concordancia con Mempool ambas tienen todos sus metodos privados, puesto que son clases importantes en donde no debería haber una corrupcion de la informacion. La unica forma de acceder a los metodos es a traves de la clase Bookkeeper.

### 4.2.1. Comandos en los que participa BlockchainHistoryBook

#### **init**

Para el comando init, existe un trabajo en conjunto de todas las clases. Primero FileManager, valida y parsea los datos ingresado. Como los datos ingresados sirven para crear una Transferencia pero no lo es en si, es Bookkeeper le encargada de generarla. Por ultimo, Builder mina el bloque devolviendoselo a Bookkeeper este accede a la HistoryBook y guarda el bloque origen. Se utiliza el método AddBlock() para la insercion. Cabe destacar que este metodo borra la historia anterior.

#### **mine**

Al igual que en init, se mina un nuevo bloque que termina siendo encadenado al último bloque presente de la AlgoChain. Se utiliza el método AddBlock()

#### **balance**

**block** Este comando solicita la información contenida en uno de los bloques almacenados por esta clase. Se ejecuta el método getBlock()

#### **txn**

**load** El proceso de carga de la información correspondiente una *Blockchain* previa a partir de un archivo *.txt*, impacta directamente en esta clase. Dicha información es almacenada ejecutando el método AddListaBlocks().

**save** La información presente como cadena de bloques en esta clase es lo que se guarda en el archivo *.txt* correspondiente.

```

2 #include "BlockChainHistoryBook.h"
3
4 lista<Block*> BlockChainHistoryBook::AlgoChain;
5
6 // Para usar x l nea de comandos block <id>
7 Block * BlockChainHistoryBook::searchBlock( const std::string HashBlock ) {
8     Block * B = NULL;
9     std::string b_prev = "";
10
11     // EL HASH YA ES VALIDADO EN FILEMANAGER
12     // Checks
13     //if ( txns_hash.empty() ) {
14     //    return B;
15     //}
16     //else if ( ! Block::CheckHash( txns_hash, TiposHash::clavehash256 ) ) {
17     //    return B;
18     //}
19     // End Checks
20
21     for ( size_t i = 0; i < (size_t) LargoHash::LargoHashEstandar; i++) { b_prev += '0'; } //
22     Block Zero
23     if ( ! AlgoChain.vacia() ) {
24         lista <Block *>::iterador it( AlgoChain );
25         it = AlgoChain.primer();
26         // if ( ! ( it.dato()->getpre_block() == b_prev ) ) {
27         //     // Mal definido el Block Zero
28         //     return B;
29         // }
30         do {
31             if ( ! ( it.dato()->getpre_block() == HashBlock ) ) {
32                 B = it.dato();
33                 break;
34             }
35             it.avanzar();
36         } while ( ! it.extremo() );
37     }
38     return B;
39 }
40
41
42 const lista<Transaction *> * BlockChainHistoryBook::searchTransaction(const std::string
43 txns_hash ){
44     const lista<Transaction *> * T = NULL;
45     std::string b_prev;
46
47     // EL HASH YA ES VALIDADO EN FILEMANAGER
48     // Checks
49     //if ( txns_hash.empty() ) {
50     //    return B;
51     //}
52     //else if ( ! Block::CheckHash( txns_hash, TiposHash::clavehash256 ) ) {
53     //    return B;
54     //}
55     // End Checks
56
57     if ( ! AlgoChain.vacia() ) {
58         lista <Block *>::iterador it( AlgoChain );
59         it = AlgoChain.primer();
60         do {
61             if ( ! ( it.dato()->gettxns_hash() == txns_hash ) ) {
62                 T = &(it.dato()->getListaTran());
63                 break;
64             }
65             it.avanzar();
66         } while ( ! it.extremo() );
67     }
68     return T;
69 }

```

```

70
71
72 TransactionInput * BlockchainHistoryBook::obtenerTransactionInput( const std::string tx_id )
73 {
74     std::string b_prev;
75     TransactionInput * TI = NULL;
76
77     // Checks
78     // if ( tx_id.empty() ) {
79     //     return TI;
80     // }
81     // else if ( ! Block::CheckHash( tx_id, TiposHash::clavehash256 ) ) {
82     //     return TI;
83     // }
84     // End Checks
85
86     //for ( size_t i = 0; i < (size_t) LargoHash::LargoHashEstandar; i++) { b_prev += '0'; }
87     // Block Zero
88     if ( ! AlgoChain.vacia() ) {
89         lista <Block *>::iterador itBlock( AlgoChain );
90         itBlock = AlgoChain.primerero();
91         do {
92
93             // Si el primer elemento no es el bloque origen hubo corrupcion de Algochain
94             // Esl problema es que la implementacion de listas dejan el primer elemento
95             // al final por eso genera error al usar AlgoChain.primerero()
96             //if ( ! ( it.dato()->getpre_block() == b_prev ) ) { return TI; }
97
98             //b_prev = it.dato()->getpre_block();
99             // Debo abrir un segundo bloque de iteraciones sobre la lista de TI
100             lista <Transaction *> trns;
101             trns = itBlock.dato()->getListatran();
102             lista <Transaction *>::iterador itTrans( trns );
103             // La lista a iterar es la de TransactionInput
104             itTrans = trns.primerero();
105             if ( ! trns.vacia() ) {
106                 do {
107                     // De las dos listas, itero las de TI
108                     lista <TransactionInput *> tInput;
109                     tInput = itTrans.dato()->getTransactionInputList();
110                     lista <TransactionInput *>::iterador itTransInput( tInput );
111                     itTransInput = tInput.primerero();
112                     if ( ! tInput.vacia() ) {
113                         do {
114                             if ( tx_id.compare(itTransInput.dato()->getTxId()) ) {
115                                 return itTransInput.dato();
116                             }
117                             itTransInput.avanzar();
118                         } while ( ! itTransInput.extremo() );
119                     }
120                     itTrans.avanzar();
121                 } while ( ! itTrans.extremo() );
122             }
123             itBlock.avanzar();
124         } while ( ! itBlock.extremo() );
125     }
126     return TI;
127 }
128
129 Transaction * BlockchainHistoryBook::getTransactionByTransactionOutputUser( const std::string
130 user ) {
131     //std::string b_prev;
132     //TransactionOutput * TI = NULL;
133     Transaction * Trans = NULL;
134
135     // Checks
136     // if ( tx_id.empty() ) {
137     //     return TI;
138     // }

```

```

137 // else if ( ! Block::CheckHash( tx_id, TiposHash::clavehash256 ) ) {
138 //     return TI;
139 // }
140 // End Checks
141
142 //for ( size_t i = 0; i < (size_t) LargoHash::LargoHashEstandar; i++) { b_prev += '0'; }
143 // Block Zero
144 if ( ! AlgoChain.vacia() ) {
145     lista <Block *>::iterador itBlock( AlgoChain );
146     itBlock = AlgoChain.primer();
147     do {
148
149         // Si el primer elemento no es el bloque origen hubo corrupcion de Algochain
150         // Es el problema es que la implementacion de listas dejan el primer elemento
151         // al final por eso genera error al usar AlgoChain.primer()
152         //if ( ! ( it.dato()->getpre_block() == b_prev ) ) { return TI; }
153
154         //b_prev = it.dato()->getpre_block();
155         // Debo abrir un segundo bloque de iteraciones sobre la lista de TI
156         lista <Transaction *> trns;
157         trns = itBlock.dato()->getListTran();
158         lista <Transaction *>::iterador itTrans( trns );
159         // La lista a iterar es la de TransactionInput
160         itTrans = trns.primer();
161         if ( ! trns.vacia() ) {
162             do {
163                 // De las dos listas, itero las de TI
164                 lista <TransactionOutput *> tOutput;
165                 tOutput = itTrans.dato()->getTransactionOutputList();
166                 lista <TransactionOutput *>::iterador itTransOutput( tOutput );
167                 itTransOutput = tOutput.primer();
168                 if ( ! tOutput.vacia() ) {
169                     do {
170                         // Si encuentro al usuario en el output
171                         // devuelvo el bloque para analizarlo
172                         // afuera
173                         if ( user.compare(itTransOutput.dato()->getAddr()) == 0 ) {
174                             return itTrans.dato();
175                         }
176                         itTransOutput.avanzar();
177                     } while ( ! itTransOutput.extremo() );
178                 }
179                 itTrans.avanzar();
180             } while ( ! itTrans.extremo() );
181         }
182         itBlock.avanzar();
183     } while ( ! itBlock.extremo() );
184 }
185 return Trans;
186 }
187 void BlockchainHistoryBook::BorrarHistoria(void){
188     // AlgoChain se autodestruye, antes debo liberar la memoria asignada en cada elemento *
189     // AlgoChain de la lista
190     // El compilador ejecuta antes los destructores de las clases hijas que liberan su memoria
191     // dinámica.
192     if ( ! AlgoChain.vacia() ) {
193         lista <Block *>::iterador it( AlgoChain );
194         it = AlgoChain.primer();
195         while ( ! AlgoChain.isEmpty() ) {
196             delete it.dato();
197             AlgoChain.eliminar_nodo(it);
198         }
199     }
200 }
201 bool BlockchainHistoryBook::AddBlock( Block *& B ){
202     Block * newBlock = new Block(*B);
203     AlgoChain.insertar(newBlock);

```

```

204     return true;
205 }

```

```

1
2 #ifndef BLOCKCHAINHISTORYBOOK_H_
3 #define BLOCKCHAINHISTORYBOOK_H_
4
5 #include <string>
6 #include "lista.h"
7 #include "Block.h"
8
9 enum class HashIdType{
10     blockId,
11     txnId,
12 };
13
14 class BlockChainHistoryBook {
15 private:
16     friend class BlockChainBookkeeper;
17     static lista<Block*> AlgoChain;
18     //---Getters---//
19     static const lista <Block *> & getListaBlocks() {return AlgoChain;};
20     // Para usar x l nea de comandos block <id>
21     static Block * searchBlock( const std::string txns_hash ); //
22     // Ante cualquier error devuelve NULL
23     // Para usar x l nea de comandos txn <id>
24     static const lista<Transaction *> * searchTransaction(const std::string txns_hash );
25     static Transaction * getTransactionByTransactionOutputUser( const std::string user );
26     static TransactionInput * obtenerTransactionInput( const std::string tx_id ); // Ante
27     // cualquier error devuelve NULL
28
29     //---Setters---//
30     //---Otros---//
31     static void BorrarHistoria( void );
32     static bool AddBlock( Block * & B );
33     static bool AddListaBlocks( lista <Block *> & lista );
34 public:
35 };
36 #endif /* BLOCKCHAINHISTORYBOOK_H_ */

```

### 4.3. Clase Mempool

Es una clase est tica en la cual se almacena la informaci n de todas las transferencias introducidas. El atributo transList, como su nombre representa, es una lista de objetos Transaction.

#### 4.3.1. Acciones por comando para Mempool

##### transfer

La clase interviene almacenando los datos correspondientes. Los datos previamente parseados por **Block-ChainFileManager** son utilizados en el m todo **addNewTransaction()** para tal fin.

**mine** Cuando se mina, Mempool recibe una consulta de parte de BlockChainBookkeeper. A trav s de BlockChainBookkeeper se devuelve el atributo transList a partir del cual se extrae la informaci n necesaria para minar un bloque. Se utiliza el m todo getTransactionsList().

**balance** Se consultan las transacciones que correspondan al usuario en cuesti n.

```

1
2 #include "Mempool.h"
3 #include "lista.h"
4

```

```

5 lista <Transaction *> Mempool::transList;
6
7 void Mempool::addNewTransaction(Transaction * & new_txn){
8
9     Transaction * newTx = new Transaction(*new_txn);
10    transList.insertar( newTx );
11 }
12
13
14 lista <Transaction *> & Mempool::getTransactionsList(){
15     return transList;
16     //Se deberia borrar la mempool cuando se trae toda la lista...
17 }
18
19 size_t Mempool::getMempoolLength(){
20     return transList.tamano();
21 }
22
23 void Mempool::BorrarMempool(void) {
24     if ( ! transList.vacia() ) {
25         lista <Transaction *>::iterador it( transList );
26         it = transList.primer();
27         while ( ! transList.isEmpty() ) {
28             delete it.dato();
29             transList.eliminar_nodo(it);
30         }
31     }
32 }
33
34
35 Transaction * Mempool::getTransactionsFromMempool(std::string hashUser){
36     if ( ! transList.vacia() ) {
37         lista <Transaction *>::iterador itTrans( transList );
38         // La lista a iterar es la de TransactionInput
39         itTrans =transList.primer();
40         if ( !transList.vacia() ) {
41             do {
42                 // De las dos listas, itero las de TI
43                 lista <TransactionOutput *> tOutput;
44                 tOutput = itTrans.dato()->getTransactionOutputList();
45                 lista <TransactionOutput *>::iterador itTransOutput( tOutput );
46                 itTransOutput = tOutput.primer();
47                 if ( ! tOutput.vacia() ) {
48                     do {
49                         // Si encuentro al usuario en el output
50                         // devuelvo el bloque para analizarlo
51                         // afuera
52                         if ( hashUser.compare(itTransOutput.dato()->getAddr()) == 0 ) {
53                             return itTrans.dato();
54                         }
55                         itTransOutput.avanzar();
56                     } while ( ! itTransOutput.extremo() );
57                 }
58                 itTrans.avanzar();
59             } while ( ! itTrans.extremo() );
60         }
61     }
62     return NULL;
63 }

```

```

1
2 #ifndef MEMPOOL_H_
3 #define MEMPOOL_H_
4
5
6 using namespace std;
7 // #include "MempoolUnit.h"
8 #include "Transaction.h"

```

```

9  #include "lista.h"
10 // static lista <Transaction *> Mempool::Mempool;
11
12
13 class Mempool{
14
15     private:
16
17         //friend class Transaction;
18         friend class BlockchainBookkeeper;
19         static lista <Transaction *> transList;
20         static void addNewTransaction(Transaction *& new_txn);    // Usa metodos de LISTA
21         para agregar un nuevo nodo con una transaccion
22         //static Transaction & get_transaction_n(int n);    // DEVUELVE LA N-ESIMA
23         TRANSACCION DE LA MEMPOOL (LA MEMPOOL SE ORGANIZA EN ORDEN DESCENDENTE)
24         static lista <Transaction *> & getTransactionsList();    // Devuelve un puntero a la
25         lista de transacciones
26         static Transaction * getTransactionsFromMempool(std::string hashUser);
27         static size_t getMempoolLength();
28         static void BorrarMempool(void);
29
30     public:
31
32 };
33
34 #endif /* MEMPOOL_H_ */

```

#### 4.4. Clase Queue

La clase Queue es una clase soporte utilizada como cola en el FileManager. La utilización de esta clase evito grandes secciones de código en donde se validaba, se pedía memoria y luego se cargaba puesto que al validar el dato, ese se encola, logrando que el proceso de pedida de memoria sea invisible para el que la use. Además al desacolarse los datos, se liberaban los bloques pedidos, por lo que usar esta clase como contenedor fue una elección inteligente.

```

1  #include "Queue.h"
2  #include <iostream>
3  using namespace std;
4
5
6  template<class T>
7  Queue<T>::Queue(): frontPtr(NULL), backPtr(NULL), count(0)
8  {
9  }
10
11 template<class T>
12 Queue<T>::~~Queue()
13 {
14     if (frontPtr != NULL)
15     {
16         while(! isEmpty()){
17             dequeue();
18         }
19     }
20 }
21
22 template<class T>
23 bool Queue<T>::isEmpty(){
24     return(count == 0);
25 }
26
27 template<class T>
28 void Queue<T>::enqueue(T data){
29     Node *newOne = new Node;

```

```

30     newOne->data = data;
31     newOne->next = NULL;
32     if(isEmpty()){
33         frontPtr = newOne;
34     }
35     else{
36         backPtr->next = newOne;
37     }
38     backPtr = newOne;
39     count++;
40 }
41
42 template<class T>
43 T Queue<T>::dequeue() {
44     if(isEmpty()){
45         T emptyData{};
46         cout << "Nothing inside" << endl;
47         return emptyData;
48     }
49     else{
50         T data;
51         Node *temp = frontPtr;
52         if(frontPtr == backPtr){
53             frontPtr = NULL;
54             backPtr = NULL;
55         }
56         else{
57             frontPtr = frontPtr->next;
58         }
59         data = temp->data;
60         delete temp;
61         count--;
62         return data;
63     }
64 }
65
66 #include "Transaction.h"
67
68 //template class Queue<int>;
69 template class Queue<size_t>;
70 //template class Queue<char>;
71 //template class Queue<float>;
72 template class Queue<string>;
73 template class Queue<Transaction>;
74
75
76 // QUEUE_H

```

```

1
2 #ifndef QUEUE_H
3 #define QUEUE_H
4 #include <cstdlib>
5
6 template<typename T>
7 class Queue
8 {
9     private:
10         struct Node{
11             T data;
12             Node *next;
13         };
14
15         Node *frontPtr;
16         Node *backPtr;
17         int count;
18
19     public:
20         Queue();

```



```

21     ~Queue();
22     bool isEmpty();
23     void enqueue(T data);
24     T dequeue();
25 };
26
27 #endif // QUEUE_H

```

## 4.5. Clase Cuentas

La clase Cuentas se trata de una lista de cuentas donde la *address* de la cuenta se encuentra en formato hexadecimal de 60 bytes. Su utilidad es complementaria a Mempool, ya que todo movimiento que se efectúe en dicha clase reciba una consulta en Cuentas para validar los fondos del usuario en una determinada operación. Es decir, mantiene un registro de las transacciones pendientes previas al minado.

```

1
2  /*
3   * Cuentas.cpp
4   */
5
6  #include "Cuentas.h"
7
8
9  //---Constructores---//
10
11  Cuentas::Cuentas() {
12      this->cantidad = 0;
13  }
14
15  //---Destructor---//
16
17  Cuentas::~Cuentas() {
18      // lista->listadocuentas se elimina en el ambito del destructor de lista.h
19      // Solo hay que liberar los punteros dentro de cada dato.
20      if ( ! this->listadocuentas.vacia() ) {
21          lista <cuentas_t *>::iterador it( listadocuentas );
22          it = this->listadocuentas.primer();
23          do {
24              delete it.dato();
25              it.avanzar();
26          } while ( ! it.extremo() );
27      }
28  }
29
30  //---Getters---//
31
32  size_t Cuentas::getcantidad() {
33      return this->cantidad;
34  }
35
36  std::string Cuentas::getalias( const std::string addr ) {
37      std::string alias = "";
38      if ( addr.empty() ) {
39          return alias;
40      }
41      else if ( ! BlockchainBuilder::CheckHash( addr, TiposHash::clavehash256 ) ) {
42          return alias;
43      }
44
45      if ( ! this->listadocuentas.vacia() ) {
46          lista <cuentas_t *>::iterador it( listadocuentas );
47          it = this->listadocuentas.primer();
48          do {
49              if ( it.dato()->addr == addr ) {
50                  alias = it.dato()->alias;
51                  break;

```

```

52         }
53         it.avanzar();
54     } while ( ! it.extremo() );
55 }
56 return alias;
57 }
58
59 size_t Cuentas::iscuenta( const std::string addr ) {
60     size_t numero = 0;
61     std::string alias = "";
62     if ( addr.empty() ) {
63         return numero;
64     }
65     else if ( ! BlockchainBuilder::CheckHash( addr, TiposHash::clavehash256 ) ) {
66         return numero;
67     }
68
69     if ( ! this->listadocuentas.vacia() ) {
70         lista <cuentas_t *>::iterador it( listadocuentas );
71         it = this->listadocuentas.primer();
72         do {
73             if ( it.dato()->addr == addr ) {
74                 numero = it.dato()->numerocuenta;
75                 break;
76             }
77             it.avanzar();
78         } while ( ! it.extremo() );
79     }
80     return numero;
81 }
82
83 float Cuentas::getsaldo( const std::string addr ) {
84     float saldo = -1;
85     if ( addr.empty() ) {
86         return saldo;
87     }
88     else if ( ! BlockchainBuilder::CheckHash( addr, TiposHash::clavehash256 ) ) {
89         return saldo;
90     }
91
92     if ( ! this->listadocuentas.vacia() ) {
93         lista <cuentas_t *>::iterador it( listadocuentas );
94         it = this->listadocuentas.primer();
95         do {
96             if ( it.dato()->addr == addr ) {
97                 saldo = it.dato()->saldo;
98                 break;
99             }
100             it.avanzar();
101         } while ( ! it.extremo() );
102     }
103     return saldo;
104 }
105
106 float Cuentas::getpendiente( const std::string addr ) {
107     float pendiente = 0;
108     if ( addr.empty() ) {
109         return pendiente;
110     }
111     else if ( ! BlockchainBuilder::CheckHash( addr, TiposHash::clavehash256 ) ) {
112         return pendiente;
113     }
114
115     if ( ! this->listadocuentas.vacia() ) {
116         lista <cuentas_t *>::iterador it( listadocuentas );
117         it = this->listadocuentas.primer();
118         do {
119             if ( it.dato()->addr == addr ) {
120                 pendiente = it.dato()->pendiente;
121                 break;

```

```

122         }
123         it.avanzar();
124     } while ( ! it.extremo() );
125 }
126 return pendiente;
127 }
128
129 const cuentas_t * Cuentas::getdetallecuenta( const std::string addr, lista <Block *> &
130     AlgoChain ) {
131     cuentas_t * C = NULL;
132     if ( addr.empty() ) {
133         return C;
134     }
135     else if ( ! BlockchainBuilder::CheckHash( addr, TiposHash::clavehash256 ) ) {
136         return C;
137     }
138     if ( ! this->listadocuentas.vacia() ) {
139         lista <cuentas_t *>::iterador it( listadocuentas );
140         it = this->listadocuentas.primer();
141         do {
142             if ( it.dato()->addr == addr ) {
143                 Extracto E(addr);
144                 C->addr = addr;
145                 C->alias = it.dato()->alias;;
146                 C->saldo = it.dato()->saldo;
147                 C->numerocuenta = it.dato()->numerocuenta;
148                 C->pendiente = it.dato()->pendiente;           // No lo tengo
149                 // lista <movimientos_t *> detalle;           // Extracto de la cuenta, necesito
150                 la AlgoChain
151                 //lista <movimientos_t *> Extracto::obtenerdetalle( const lista <Block *> &
152                 AlgoChain, std::string addr ) {
153                     C->detalle = E.obtenerdetalle( AlgoChain, addr );
154                     break;
155                 }
156                 it.avanzar();
157             } while ( ! it.extremo() );
158         }
159         return C;
160     }
161 }
162
163 size_t Cuentas::getnumerocuenta( const std::string addr ) {
164     // Checks
165     if ( addr.empty() ) { return 0; }
166     else if ( ! BlockchainBuilder::CheckHash( addr, TiposHash::clavehash256 ) ) {
167         return 0;
168     }
169     if ( ! this->listadocuentas.vacia() ) {
170         lista <cuentas_t *>::iterador it( listadocuentas );
171         it = this->listadocuentas.primer();
172         do {
173             if ( it.dato()->addr == addr ) {
174                 return it.dato()->numerocuenta;
175             }
176             it.avanzar();
177         } while ( ! it.extremo() );
178     }
179     return 0;
180 }
181
182 size_t Cuentas::getnumerocuenta( const std::string addr, const std::string alias ) {
183     // Checks
184     if ( ! addr.empty() ) { return Cuentas::getnumerocuenta(addr); }
185     if ( alias.empty() ) { return 0; }
186     if ( ! this->listadocuentas.vacia() ) {
187         lista <cuentas_t *>::iterador it( listadocuentas );

```

```

189         it = this->listadocuentas.primer();
190         do {
191             if ( it.dato()->alias == alias ) {
192                 return it.dato()->numerocuenta;
193             }
194             it.avanzar();
195         } while ( ! it.extremo() );
196     }
197     return 0;
198 }
199
200
201 //---Setters---//
202 bool Cuentas::setalias( const std::string addr, const std::string alias ) {
203
204     // Checks
205     if ( addr.empty() ) { return false; }
206     else if ( ! BlockchainBuilder::CheckHash( addr, TiposHash::clavehash256 ) ) {
207         return false;
208     }
209
210     if ( ! this->listadocuentas.vacia() ) {
211         lista <cuentas_t *>::iterador it( listadocuentas );
212         it = this->listadocuentas.primer();
213         do {
214             if ( it.dato()->addr == addr ) {
215                 it.dato()->alias = alias;
216                 break;
217             }
218             it.avanzar();
219         } while ( ! it.extremo() );
220         return ( ! it.extremo() );
221     }
222     else { return false; }
223 }
224
225 bool Cuentas::setpendiente( const std::string addr, const float monto ) {
226     // Checks
227     if ( addr.empty() ) { return false; }
228     else if ( ! BlockchainBuilder::CheckHash( addr, TiposHash::clavehash256 ) ) {
229         return false;
230     }
231     if ( monto == 0 ) { return false; }
232
233     if ( ! this->listadocuentas.vacia() ) {
234         lista <cuentas_t *>::iterador it( listadocuentas );
235         it = this->listadocuentas.primer();
236         do {
237             if ( it.dato()->addr == addr ) {
238                 it.dato()->pendiente = monto;
239                 break;
240             }
241             it.avanzar();
242         } while ( ! it.extremo() );
243         return ( ! it.extremo() );
244     }
245     else { return false; }
246 }
247
248 bool Cuentas::setsaldo( const std::string addr, const float monto ) {
249
250     // Checks
251     if ( addr.empty() ) { return false; }
252     if ( monto < 0 ) { return false; }
253
254     if ( ! this->listadocuentas.vacia() ) {
255         lista <cuentas_t *>::iterador it( listadocuentas );
256         it = this->listadocuentas.primer();
257         do {
258             if ( it.dato()->addr == addr ) {

```

```

259         it.dato()->saldo = monto;
260         break;
261     }
262     it.avanzar();
263 } while ( ! it.extremo() );
264 return ( ! it.extremo() );
265 }
266 else { return false; }
267 }
268
269 //---Otros---//
270
271 size_t Cuentas::NuevoNumero() {
272     // Genera un autom tico.
273     static size_t Id = 1;    // Arranca de 1, 0 -> error
274     return Id++;
275 }
276
277 bool Cuentas::addcuenta( const std::string addr, const std::string alias, const float monto )
278 {
279     cuentas_t * C = NULL;
280
281     // Checks
282     if ( addr.empty() ) {
283         return false;
284     }
285     else if ( ! BlockchainBuilder::CheckHash( addr, TiposHash::clavehash256 ) ) {
286         return false;
287     }
288
289     // Verificar si esta duplicada
290     if ( ! this->listadocuentas.vacia() ) {
291         lista <cuentas_t *>::iterador it( listadocuentas );
292         it = this->listadocuentas.primer();
293         do {
294             if ( it.dato()->addr == addr ) {
295                 // Actualizar el monto y/o el alias
296                 if ( !alias.empty() ) {
297                     it.dato()->alias = alias;
298                     cout << "datos actualizados: Alias -> " << it.dato()->alias << endl;
299                 }
300                 it.dato()->saldo = monto;
301                 cout << "datos actualizados: Monto -> " << it.dato()->saldo << endl;
302                 return false;
303             }
304             it.avanzar();
305         } while ( ! it.extremo() );
306     }
307
308     try {
309         C = new cuentas_t;
310         C->addr = addr;
311         C->saldo = monto;
312         C->alias = alias;
313         C->numerocuenta = Cuentas::NuevoNumero();
314         this->listadocuentas.insertar( C );
315         cantidad++;
316     }
317     catch (std::bad_alloc& ba)
318     {
319         std::cerr << "bad_alloc caught: " << ba.what() << '\n';
320     }
321
322     return true;
323 }
324
325 bool Cuentas::addcuenta( const std::string addr, const std::string alias ) {
326     return this->addcuenta( addr, alias, 0 /* Para eliminar ambigüedades en la sobrecarga */ );
327 }

```

```

327 bool Cuentas::addcuenta( const std::string addr, const float monto ) {
328     return this->addcuenta( addr, "", monto );
329 }
330
331 bool Cuentas::deposito( const std::string addr, const float monto ) {
332
333     // Checks
334     if ( addr.empty() ) { return false; }
335     if ( monto <= 0 ) { return false; }
336
337     if ( ! this->listadocuentas.vacia() ) {
338         lista <cuentas_t *>::iterador it( listadocuentas );
339         it = this->listadocuentas.primer();
340         do {
341             if ( it.dato()->addr == addr ) {
342                 it.dato()->saldo += monto;
343                 break;
344             }
345             it.avanzar();
346         } while ( ! it.extremo() );
347         return ( ! it.extremo() );
348     }
349     else { return false; }
350 }
351
352 bool Cuentas::extraccion( const std::string addr, const float monto ) {
353
354     // Checks
355     if ( addr.empty() ) { return false; }
356     if ( monto > 0 ) { return false; }
357
358     if ( ! this->listadocuentas.vacia() ) {
359         lista <cuentas_t *>::iterador it( listadocuentas );
360         it = this->listadocuentas.primer();
361         do {
362             if ( it.dato()->addr == addr ) {
363                 if ( it.dato()->saldo > monto ) {
364                     it.dato()->saldo -= monto;
365                     return true;
366                 }
367                 break;
368             }
369             it.avanzar();
370         } while ( ! it.extremo() );
371         return false;
372     }
373     else { return false; }
374 }
375
376 bool Cuentas::openlista( const std::string file ) {
377     std::string linea = "", alias = "", hashcuenta = "";
378     if ( file.empty() ) return false;
379
380     ifstream archivoClientes( file, ios::in );
381     if ( !archivoClientes ) {
382         cerr << "Error al abrir: " << file << endl;
383         return false;
384     }
385
386     while ( ! archivoClientes.eof() ) {
387         getline( archivoClientes, linea );
388         if ( linea.empty() ) break;
389         if ( linea.length() >= (size_t) LargoHash::LargoHashEstandar ) {
390             size_t pos = 0;
391             try {
392                 // Formato clavehash + ", " + alias
393                 hashcuenta = linea.substr( 0, (size_t) LargoHash::LargoHashEstandar );
394                 if ( linea.length() > (size_t) LargoHash::LargoHashEstandar ) {
395                     alias = linea.substr( (size_t) LargoHash::LargoHashEstandar );
396                     pos = alias.find( ", " );

```

```

397         if ( pos != std::string::npos ) {
398             alias = alias.substr( pos + 1 );
399         }
400         if ( alias.substr(0, 1) == " " ) {
401             alias = alias.substr( 1 );
402         }
403     }
404     else {
405         alias = "";
406     }
407     if ( BlockchainBuilder::CheckHash( hashcuenta, TiposHash::clavehash256 ) ) {
408         // Ver si ya está en la lista
409         if ( ! iscuenta( hashcuenta ) ) {
410             cuentas_t * cuenta;
411             cuenta = new cuentas_t;
412             cuenta->addr = hashcuenta;
413             cuenta->alias = alias;
414             cuenta->saldo = 0;
415             cuenta->pendiente = 0;
416             cuenta->numerocuenta = NuevoNumero();
417             this->listadocuentas.insertar( cuenta );
418             cantidad++;
419         }
420     }
421     else {}
422 }
423 catch ( const std::length_error& e ) {
424     std::cerr << e.what() << '\n';
425 }
426 catch ( std::bad_alloc& ba ) {
427     std::cerr << "bad_alloc caught: " << ba.what() << '\n';
428 }
429 }
430 else {}
431 }
432 archivoClientes.close();
433
434 return true;
435 }
436
437 bool Cuentas::savelista( const std::string file ) {
438
439     if ( file.empty() ) return false;
440     if ( this->listadocuentas.vacia() ) return false;
441
442     ofstream archivoClientes( file, ios::out );
443     if ( !archivoClientes ) {
444         cerr << "Error al abrir: " << file << endl;
445         return false;
446     }
447
448     if ( ! this->listadocuentas.vacia() ) {
449         std::string linea = "", alias = "", hashcuenta = "";
450         lista <cuentas_t *>::iterador it( listadocuentas );
451         it = this->listadocuentas.primer();
452         do {
453             if ( it.dato()->alias.empty() ) {
454                 archivoClientes << it.dato()->addr << endl;
455             }
456             else {
457                 archivoClientes << it.dato()->addr << " " << it.dato()->alias << endl;
458             }
459             it.avanzar();
460         } while ( ! it.extremo() );
461         return ( it.extremo() ); // Toda la lista grabada
462     }
463     else { return false; }
464
465     archivoClientes.close();
466

```

```

467     return true;
468 }

```

```

1
2  /*
3   * Cuentas.h
4   */
5
6  #ifndef CUENTAS_H_
7  #define CUENTAS_H_
8
9  #include<string>
10 #include<iostream>
11 #include<fstream>
12
13 #include "Extracto.h"
14 #include "TiposHash.h"
15
16 #include "lista.h"
17
18 class Cuentas {
19 private:
20     lista <cuentas_t *> listadocuentas;          // Lista de cuentas_t
21     size_t cantidad;                          // Total de cuentas_t
22     size_t NuevoNumero();
23
24 public:
25     //---Constructores---//
26     Cuentas();
27     //---Destructor---//
28     ~Cuentas();
29     //---Getters---//
30     float getsaldo( const std::string addr );
31     float getpendiente( const std::string addr );
32     std::string getalias( const std::string addr );
33     size_t getnumerocuenta( const std::string addr );
34     size_t getnumerocuenta( const std::string addr, const std::string alias );
35     const cuentas_t * getdetallecuenta( const std::string addr, lista <Block *> & AlgoChain );
36     size_t getcantidad();
37
38     //---Setters---//
39     bool setalias( const std::string addr, const std::string alias );
40     bool setsaldo( const std::string addr, const float monto );
41     bool setpendiente( const std::string addr, const float monto );
42
43     //---Otros---//
44     size_t iscuenta( const std::string addr );
45     bool addcuenta( const std::string addr, const std::string alias = "", const float monto =
46     0 );
47     bool addcuenta( const std::string addr, const std::string alias = "" );
48     bool addcuenta( const std::string addr, const float monto );
49     bool deposito( const std::string addr, const float monto );
50     bool extraccion( const std::string addr, const float monto );
51     bool depositopendiente( const std::string addr, const float monto );
52     bool extraccionpendiente( const std::string addr, const float monto );
53     void listadototal( const float saldominimo = 0 );
54
55     // Persistencia del Objeto //
56     bool openlista( const std::string file );
57     bool savelista( const std::string file );
58 };
59
60 #endif /* CUENTAS_H_ */

```



## 4.6. Clase BlockChainManager

La clase **BlockChainManager** sigue con la idea de gestionar el proyecto. Durante esta implementación fue fácil agregar estados de error o estados validos puesto que se contaba con la implementación del TP0 teniendo en cuenta que el código incrementaría en complejidad. Podrían inferirse algunas funciones elementales :

1. Gestionar el flujo del programa a través de estados : Cada proceso de las clases FileManager y Builder son gestionadas según el Manager y en base a que tipo de estado estas indican, se decide la lógica del código.(Como un jefe decide sobre el destino de un proyecto según lo que le indican sus empleados)
2. Instancia y opera con las clases *FileManager* y *Builder*: Si bien en un concepto redundante, permite tener localizado el código en un solo lado, dado que no se instancian en ningún otro lugar las clases que hacen el trabajo. (Como un jefe contrata y ordena a los empleados a realizar el trabajo)
3. Opera de interfaz entre el *main* y todo lo relacionado a *BlockChain*: Este concepto esta ligado a POO puesto que trata de incitar el concepto de encapsulamiento. Toda información que provenga de por fuera de los límites del programa pasa por el **BlockChainManager**, y todo lo relacionado a *BlockChain* queda contenido en de su entorno (así como un jefe hace el contacto con el cliente, pero el cliente no sabe *hacia dentro* cómo se realiza su trabajo)

Se decidió mantener la implementación de esta clase de manera estática. En particular se reforzó el encapsulamiento de la *BlockChain* dentro del main. Esta decisión fue tomada puesto que fue necesario hacer un *refactor* sobre la clase FileManager. Por ende, aprovechando el cambio de paradigma de los argumentos de línea de comando del presente TP, se implementó un método (setUserFilename) capaz de cargar los datos del archivo del usuario en una estructura denominada *file*, *logrando así desacoplar el main del manejo de archivos*.

En síntesis, la clase **BlockChainManager** es la *estructura jefe*. Es decir, es la clase desde la cuál se desprende el funcionamiento del resto de las clases y la gestión del hilo del programa.

Los métodos principales de la clase **BlockChainManager** son:

1. **processBlockChain()**: gestiona todo el flujo del programa a través de un llamado secuencial a los métodos correspondientes a cada paso (detallados más abajo)
2. **procesStatus()**: Se encarga de procesar y comunicar los diversos mensajes de estados que maneja la aplicación (tanto los informativos como los de error)

Otro cambio que se vio reflejado en Manager fue tener que cambiar la lógica de ProcessBlockChain puesto que cambio radicalmente el enfoque con respecto a TP0. La idea es utilizar diagrama de estados con memoria, dependiendo de los comandos que se ingresan. En primer lugar se instancia al FileManager, clase que opera con archivos y hace que genere un nuevo tipo de dato denominado payload. *La idea es que FileManager realice la validación de archivos y guarde los datos*

```
1 //----- BlockChainManager.cpp -----//
2
3
4
5 #include "BlockChainManager.h"
6
7 status_t BlockChainManager::state = STATUS_OK;
8 status_t BlockChainManager::command = STATUS_READING_COMMANDS;
9
10 // Descripcion: procesBlockChain es un metodo que encapsula toda la logica interna de
11 // en este metodo, dependiendo el comando de entrada, se instanciaran las clases
12 // BlockChainBuilder,
13 // BlockChainBookkeeper y BlockChainFileManager que realizaran la tarea seleccionada.
14 // Precondicion: Toda logica relacionada a blockchain debe estar invocada desde esta zona
15 // Postcondicion:
16 void BlockChainManager::procesBlockChain() {
17     BlockChainManager::procesStatus(STATUS_READING_COMMANDS);
18     BlockChainFileManager fileManager;
19     std::string command;
20     payload_t payload;
21     fileManager.safeValuePayload(payload);
```

```

22 while (BlockchainManager::command != STATUS_NO_MORE_COMMANDS ) {
23     BlockchainManager::proccesStatus( fileManager.translateCommands(payload) );
24     std::cout<< "Begin Parse Command ...";
25     switch(payload.command)
26     {
27         case Commands::init:
28             {
29                 std::cout<< "Done"<< std::endl;
30                 //-----//
31                 // Datos del payload que sirven en este caso.
32                 // std::string usr = payload.user;
33                 // float value = payload.value;
34                 // unsigned int bits = payload.bits;
35                 //-----//
36
37                 BlockchainBookkeeper bookkeeper;
38                 BlockchainBuilder builder(payload.bits);
39
40                 // A partir del payload bookkeeper crea una transaccion
41                 BlockchainManager::proccesStatus( bookkeeper.createOriginTransaction(
payload) );
42
43                 // Builder crea un bloque origen con los datos suministrados por
bookkeeper
44                 BlockchainManager::proccesStatus( builder.createOriginBlock( *(bookkeeper.
getActualTransaction()) ) );
45                 fileManager << FileTypes::userCommandResponseFiles << builder.
getObtainedHash()<<"\\n";
46
47                 // Bookkeeper guarda ese bloque en la historia y actualiza su lista de
usuarios
48                 BlockchainManager::proccesStatus( bookkeeper.saveOriginBlockInHistoryBook(
builder.getBlocklActual() ) );
49             }
50             break;
51         case Commands::transfer:
52             {
53                 std::cout<< "Done"<< std::endl;
54                 //-----//
55                 // Datos del payload que sirven en este caso.
56                 // Queue<std::string> * ArgTransfer = payload.ArgTransfer
57
58                 BlockchainBookkeeper bookkeeper;
59
60                 // Bookkeeper intenta armar una Transaccion, funciona como validacion
61                 // puesto que
62                 // si no lo logra completar, el usuario no existe en la historia
63                 BlockchainManager::proccesStatus( bookkeeper.createTransaction(payload));
64                 fileManager << FileTypes::userCommandResponseFiles << bookkeeper.
getTransactionHash()<<"\\n";
65
66                 // Bookkeeper guarda ese bloque en la mempool y actualiza su lista de
usuarios
67                 BlockchainManager::proccesStatus( bookkeeper.saveInMempool(bookkeeper.
getActualTransaction()));
68             }
69             break;
70         case Commands::mine:
71             {
72                 std::cout<< "Done"<< std::endl;
73                 //-----//
74                 // Datos del payload que sirven en este caso.
75                 // unsigned int bits = payload.bits;
76                 //-----//
77                 BlockchainBookkeeper bookkeeper;
78                 BlockchainBuilder builder(payload.bits);
79
80                 // Bookkeeper indaga en la historia buscando el hash del ultimo bloque
81

```

```

82         std::string prevoiusBlockHash = bookkeeper.getLastBlockHash();
83
84         // Builder mina a partir de la mempool y del hash del bloque anterior
85         BlockChainManager::procesStatus( builder.createBlock( bookkeeper.
getMempool(),prevoiusBlockHash) );
86         fileManager << FileTypes::userCommandResponseFiles << builder.
getObtainedHash() <<"\\n";
87
88         // Bookkeeper guarda ese bloque en la historia y actualiza su lista de
usuarios
89         BlockChainManager::procesStatus( bookkeeper.saveBlockInHistoryBook(
builder.getBlocklActual() ) );
90     }
91     break;
92     case Commands::block:
93     {
94         std::cout<< "Done"<< std::endl;
95         //-----//
96         // Datos del payload que sirven en este caso.
97         // std::string id = payload.id;
98         //-----//
99         BlockChainBookkeeper bookkeeper;
100
101         // Blockkeeper busca en su libro contable el bloque requerido.
102         // 'Block' es un tipo enumerativo interno de bookkeeper para
103         // buscar solo mirando los hash de bloques.
104         BlockChainManager::procesStatus( bookkeeper.searchInHistoryBook(
HashIdType::blockId, payload.id ) );
105
106         // Imprime la transaccion en el archivo de respuesta
107         BlockChainManager::procesStatus( fileManager.convert( FileTypes::
userCommandResponseFiles,bookkeeper.getBlockList() ) );
108
109     }
110     break;
111     case Commands::balance:
112     {
113         std::cout<< "Done"<< std::endl;
114         //-----//
115         // Datos del payload que sirven en este caso.
116         // std::string usr = payload.usr;
117         //-----//
118         // BlockChainBookkeeper bookkeeper;
119
120         // Blockkeeper busca en su libro lista de usuarios al usr correspondiente
121         // BlockChainManager::procesStatus( bookkeeper.searchInUserList( payload.
usr );
122
123     }
124     break;
125     case Commands::txn:
126     {
127         std::cout<< "Done"<< std::endl;
128         //-----//
129         // Datos del payload que sirven en este caso.
130         // std::string id = payload.id;
131         //-----//
132         BlockChainBookkeeper bookkeeper;
133
134         // Blockkeeper busca en su libro contable la transaccion requerida.
135         // 'Transaction' es un tipo enumerativo interno de bookkeeper para
136         // buscar solo mirando los hash de Transaction.
137         BlockChainManager::procesStatus( bookkeeper.searchInHistoryBook(
HashIdType::txnId, payload.id ) );
138
139         // Imprime la transaccion en el archivo de respuesta
140         BlockChainManager::procesStatus( fileManager.convert( FileTypes::
userCommandResponseFiles,bookkeeper.getTransactionList() ) );
141
142     }

```

```

143         break;
144     case Commands::load:
145     {
146         std::cout<< "Done"<< std::endl;
147         //-----//
148         // Datos del payload que sirven en este caso.
149         // std::string filename = payload.filename;
150         //-----//
151         // El filemanager ya esta intanciado antes.
152         BlockchainBookkeeper bookkeeper;
153
154         // Filemanager abre el archivo pasado como argumento dentro del payload.
155         file_t newFile;
156         newFile.fileID = payload.filename;
157         newFile.type = FileTypes::loadBlockchainFile;
158         newFile.mode = ios::in;
159         newFile.isStandard = false;
160         BlockchainManager::proccesStatus( fileManager.addFile(newFile) );
161
162         // Filemanager valida que la estructura de la blockchain este
163     correctamente
164         BlockchainManager::proccesStatus( fileManager.validateBlockChain() );
165
166         // FileManager parsea los datos de la blockchain en la lista de bloques
167         BlockchainManager::proccesStatus( fileManager.loadBlockChain() );
168         // Bookkeeper guarda en la historia
169         BlockchainManager::proccesStatus( bookkeeper.
saveUserBlockChainInHistoryBook( fileManager.getBlockChainPointer() ));
170
171         // Filemanager cierra el archivo pasado como argumento dentro del payload.
172         BlockchainManager::proccesStatus( fileManager.removeFile( newFile.type) );
173         fileManager<<FileTypes::userCommandResponseFiles<< "OK \n";
174     }
175     break;
176     case Commands::save:
177     {
178         std::cout<< "Done"<< std::endl;
179         //-----//
180         // Datos del payload que sirven en este caso.
181         // std::string filename = payload.filename;
182         //-----//
183         // El filemanager ya esta intanciado antes.
184         BlockchainBookkeeper bookkeeper;
185         file_t newFile;
186         newFile.fileID = payload.filename;
187         newFile.type = FileTypes::saveBlockChainFile;
188         newFile.mode = ios::out;
189         newFile.isStandard = false;
190
191         // Filemanager abre el archivo pasado como argumento dentro del payload.
192         BlockchainManager::proccesStatus( fileManager.addFile(newFile) );
193
194         std::cout<< "Begin Converting Block to File ..." << std::endl;
195         BlockchainManager::proccesStatus( fileManager.convert(FileTypes::
saveBlockChainFile,bookkeeper.getBlockChain()) );
196
197         // Filemanager cierra el archivo pasado como argumento dentro del payload.
198         BlockchainManager::proccesStatus( fileManager.removeFile( newFile.type) );
199
200         fileManager<<FileTypes::userCommandResponseFiles<< "OK \n";
201     }
202     break;
203     default:
204         BlockchainManager::command = STATUS_NO_MORE_COMMANDS;
205         std::cout<< "Fail: Error Not Defined"<< std::endl;
206         break;
207 }
208 }
209

```

```

210
211
212     cout << "Finishing Execution " << endl;
213     BlockchainBookkeeper bookkeeper;
214     BlockchainManager::proccesStatus( bookkeeper.eraseAllBlockchainRegisters() );
215     BlockchainManager::proccesStatus(fileManager.removeAllFiles());
216
217 }
218
219 // Descripcion: proccesStatus es un metodo que analiza los estados de salida de las clases
220 // BlockchainBuilder, BlockchainBookkeeper y BlockchainFileManufer que son las que realizan
221 // las tareas. En base a dichos estados BlockchainManager decide el flujo del programa.
222 // Precondicion:
223 // Postcondicion:
224 void BlockchainManager::proccesStatus(status_t status){
225     BlockchainFileManager fileManager;
226     state = status;
227     switch(status){
228         // ----- Terminaciones Correctas-----//
229         case STATUS_OK:
230             //std::cout << "Done" << std::endl;
231             break;
232         case STATUS_READING_COMMANDS:
233             command = STATUS_READING_COMMANDS;
234             //std::cout << "Reading.." << std::endl;
235             break;
236         case STATUS_NO_MORE_COMMANDS:
237             command = STATUS_NO_MORE_COMMANDS;
238             //std::cout << "Reading.." << std::endl;
239             break;
240         case STATUS_OPEN_FILE_SUCCESSFULLY:
241             break;
242         case STATUS_CLOSE_FILE_SUCCESSFULLY:
243             break;
244         case STATUS_FINISH_CONVERT_SUCCESSFULLY:
245             break;
246         //----- Errores -----//
247         case STATUS_ERROR_HASH_NOT_FOUND:
248             fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
249             std::cerr << "Error Comando no conocido" << std::endl;
250             //std::abort();
251             break;
252         case STATUS_ERROR_COMMAND_NOT_FOUND:
253             fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
254             std::cerr << "Error Comando no conocido" << std::endl;
255             //std::abort();
256             break;
257         case STATUS_ERROR_COMMAND_PAYLOAD:
258             fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
259             std::cerr << "Error Comando con argumentos invalidos" << std::endl;
260             //std::abort();
261             break;
262         case STATUS_CORRUPT_FORMAT:
263             fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
264             //std::cout << "Error de Formato: Formato Incorrecto" << std::endl;
265             std::cerr << "Error de Formato: Formato Incorrecto" << std::endl;
266             //std::abort();
267             break;
268         case STATUS_CORRUPT_FORMAT_BAD_BITS:
269             fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
270             //std::cout << "Error de Formato: Hash incorrecto" << std::endl;
271             std::cerr << "Error de Formato: Hash incorrecto" << std::endl;
272             //std::abort();
273             break;
274         case STATUS_CORRUPT_FORMAT_BAD_NONCE:
275             fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
276             //std::cout << "Error de Formato: Hash incorrecto" << std::endl;
277             std::cerr << "Error de Formato: Hash incorrecto" << std::endl;
278             //std::abort();
279             break;

```

```

280 case STATUS_CORRUPT_FORMAT_BAD_HASH:
281     fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
282     //std::cout << "Error de Formato: Hash incorrecto" << std::endl;
283     std::cerr << "Error de Formato: Hash incorrecto" << std::endl;
284     //std::abort();
285     break;
286 case STATUS_CORRUPT_FORMAT_BAD_TXINDEX:
287     fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
288     //std::cout << "Error de Formato: Indice de Tx incorrecto" << std::endl;
289     std::cerr << "Error de Formato: Indice de Tx incorrecto" << std::endl;
290     //std::abort();
291     break;
292 case STATUS_CORRUPT_FORMAT_BAD_TXIN:
293     fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
294     //std::cout << "Error de Formato: Indice Tx In Incorrecto" << std::endl;
295     std::cerr << "Error de Formato: Indice Tx In Incorrecto" << std::endl;
296     //std::abort();
297     break;
298 case STATUS_CORRUPT_FORMAT_BAD_TXOUT:
299     fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
300     //std::cout << "Error de Formato: Indice Tx Out Incorrecto" << std::endl;
301     std::cerr << "Error de Formato: Indice Tx Out Incorrecto" << std::endl;
302     //std::abort();
303     break;
304 case STATUS_CORRUPT_FORMAT_BAD_BTCVALUE:
305     fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
306     //std::cout << "Error de Formato: Valor de Bitcoin Incorrecto" << std::endl;
307     std::cerr << "Error de Formato: Valor de Bitcoin Incorrecto" << std::endl;
308     //std::abort();
309     break;
310 case STATUS_BAD_ALLOC:
311     fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
312     //std::cout << "Error de sistema: Memoria insuficiente" << std::endl;
313     std::cerr << "Error de sistema: Memoria insuficiente" << std::endl;
314     std::abort();
315     break;
316 case STATUS_BAD_READ_INPUT_FILE:
317     fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
318     //std::cout << "Error de Lectura: Archivo de entrada daĩ¿ado" << std::endl;
319     std::cerr << "Error de Lectura: Archivo de entrada daĩ¿ado" << std::endl;
320     std::abort();
321     break;
322 case STATUS_BAD_READ_OUTPUT_FILE:
323     fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
324     //std::cout << "Error de Lectura: Archivo de salida daĩ¿ado" << std::endl;
325     std::cerr << "Error de Lectura: Archivo de salida daĩ¿ado" << std::endl;
326     std::abort();
327     break;
328 case STATUS_ERROR_LIST_OF_TRANSACT_NOT_SUPPORTED:
329     fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
330     //std::cout << "Error de Conversion: No hay nada que convertir" << std::endl;
331     std::cerr << "Error: El sistema no soporta entradas con mas de una transaccion por
bloque" << std::endl;
332     std::abort();
333     break;
334 case STATUS_NO_BLOCKS_TO_CONVERT:
335     fileManager << FileTypes::userCommandResponseFiles << "FAIL\n";
336     //std::cout << "Error de Conversion: No hay nada que convertir" << std::endl;
337     std::cerr << "Error de Conversion: No hay nada que convertir" << std::endl;
338     //std::abort();
339     break;
340 default:
341     std::cout << std::endl;
342     break;
343 }
344 }
345
346
347 lista<file_t *> BlockchainManager::userFiles;
348

```

```

349 // Descripcion: setUserFilename es un metodo utilizado en cmdline en la version 2.1.1
350 // con el objetivo de cargar el archivo del usuario en un archivo file_t y darselo al
351 // filemanager. En este metodo solo se cargaran los campos de filename, mode y type.
352 // - filename: hace referencia al nombre del archivo. En caso de no ser un archivo real, se
353 // denominara como Standard Input o Standard Output.
354 // - mode: hace referencia a si es un archivo de entrada ios::in o de salida ios::out
355 // - type: hace referencia al tipo de archivo que se pasara. En este metodo solo existiran
356 // los tipos FileTypes::userCommandInputFiles y FileTypes::userCommandResponseFiles.
357 // Precondicion: Se espera que los argumentos de este setter esten correctos
358 // Postcondicion: Se da un file_t a fileManager.
359 void BlockchainManager::setUserFilename(ios_base::openmode mode, std::string filename, bool
    isStandard){
360     BlockchainFileManager fileManager;
361     //---Si es un archivo de entrada---//
362     if( mode == ios::in){
363         if(isStandard){
364             filename += " Input";
365             cout << "La direccion del archivo Origen es : Cin (Entrada Standard)" << endl;
366         }
367         else{
368             cout<< "La direccion del archivo Origen es : "<< filename << endl;
369         }
370         file_t usrFile;
371         usrFile.fileID = filename;
372         usrFile.mode = mode;
373         usrFile.type = FileTypes::userCommandInputFiles;
374         usrFile.isStandard = isStandard;
375         BlockchainManager::proccesStatus( fileManager.addFile(usrFile) );
376     }else //---Si es un archivo de salida---//
377     if( mode == ios::out){
378         if(isStandard){
379             filename += " Output";
380             cout<< "La direccion del archivo Destino es: Cout (Salida Standard)" << endl;
381         }
382         else{
383             cout<< "La direccion del archivo Destino es : "<< filename <<endl;;
384         }
385         file_t usrFile;
386         usrFile.fileID = filename;
387         usrFile.mode = mode;
388         usrFile.type = FileTypes::userCommandResponseFiles;
389         usrFile.isStandard = isStandard;
390         BlockchainManager::proccesStatus( fileManager.addFile(usrFile) );
391     }
392 }
393
394 //-----//
395 // OBSOLETO PARA LA VERSION 2.1.1 (Tp1)
396
397 #define DIFFICULTY_DEFAULT_VALUE 3
398 unsigned int BlockchainManager::userDefinedDifficulty = DIFFICULTY_DEFAULT_VALUE;
399
400
401 // Descripcion: Metodo que se utilizo en la version 1.1.1 (Tp0) para cargar el valor de
402 // dificultad (bits) desde la clase cmdline. En la version 2.1.1 (Tp1) ya no se vuelve a usar
403 // Precondicion: Debe llegar un int para verificar el signo
404 // Postcondicion: Debe cargarse un unsined_int en la variable estatica userDefinedDifficulty
405 // con el valor de entrada
406 void BlockchainManager::setUserDefinedDifficulty(int d){
407     if( d < 0 ){
408         std::cout << "Error de Formato: Dificultad debe ser mayor a cero " << std::endl;
409         std::cerr << "Error de Formato: Dificultad debe ser mayor a cero" << std::endl;
410         std::abort();
411     }
412     userDefinedDifficulty = (unsigned int) d;
413 }
414
415 // Descripcion: Metodo que se utilizo en la version 1.1.1 (Tp0) devolver el valor de
416 // userDefinedDifficulty. En la version 2.1.1 (Tp1) ya no se vuelve a usar
417 // Precondicion: Debe haberse cargado userDefinedDifficulty previamente con el metodo setter

```

```

418 // Postcondicion: Devuelve el valor de userDefinedDifficulty
419 unsigned int BlockChainManager::getUserDefinedDifficulty( void ){
420     return userDefinedDifficulty;
421 }

```

```

1 //----- BlockChainManager.h -----//
2
3
4 #ifndef BLOCKCHAINMANAGER_H_
5 #define BLOCKCHAINMANAGER_H_
6
7 #include <iostream>
8 #include "BlockChainStatus.h"
9 #include "BlockChainFileManager.h"
10 #include "BlockChainBuilder.h"
11 #include "BlockChainBookkeeper.h"
12
13 class BlockChainManager {
14     static status_t state;
15     static status_t command;
16     static unsigned int userDefinedDifficulty;
17     static lista<file_t *> userFiles;
18     static void parseCommands(std::string command,payload_t & payload);
19 public:
20
21     static void procesBlockChain();
22     static void procesStatus( status_t status );
23
24     static void setUserDefinedDifficulty(int d);
25     static unsigned int getUserDefinedDifficulty(void);
26
27     static void setUserFilename(ios_base::openmode mode , std::string filename = "Standard",
28     bool isStandar = true);
29 };
30 #endif /* BLOCKCHAINMANAGER_H_ */

```

## 4.7. Clase BlockchainFileManager

La clase *BlockChainFileManager*, como su nombre lo indica, se encarga de centrar todo lo relacionado a archivos. Esta clase oficia de interfaz entre los archivos del usuario y los archivos internos. Dado el cambio de sentido del presente Tp, fue necesario la realización de una refactorización. La idea principal de esta clase sigue siendo la misma, pero ahora los métodos de validación y parseo implementados en el Tp0 ya no fueron usados, la diferencia radicó en la cantidad de distintas entradas escritas por el usuario (lo que maximiza la cantidad de errores) y el aumento de los archivos a utilizar (que pasó de 2, que siempre estaban abiertos, a 4 en donde 2 pueden no abrirse nunca, dependiendo el comando)

Debido a estos cambios fue necesario un trabajo arduo en esta clase, para la validación de una *Blockchain* de entradas, validación de cada comando ingresado por consola. En particular el comando transfer, que no tiene un ancho predeterminado sino que posee un ancho variable (para el mismo se utilizó la clase *Queue* (ver sección ??)). Por otro lado, otro de los cambios mencionados fue la capacidad de tener una lista de archivos. Dado que esta lista debía ser visible para cualquier instancia de FileManager, se optó por declarar dicha lista como atributo estático. Dicho atributo opera como variable global en la clase logrando mantener la misma lista para toda instancia. Este cambio logro también solucionar problemas del Tp0 en donde no era posible correr el algoritmo desde entrada/salida standard. Por otra parte, se comenzó a observar que la estructura de datos *raw<sub>t</sub>* utilizada en el Tp0 ya no era representativa, por lo cual se optó por usar una clase nativa de transacciones, la clase Transaction. Esto permitió tener menor *overhead* y conversión a la hora de utilizar dichas transacciones.

1. Validación: Dado que se trabaja con *hashes* y estos son sensibles a pequeños cambios, se debe hacer una validación estricta antes de procesar. En esta etapa el *FileManager*. verifica el formato especificado del todo el archivo. Implementado en el método *validate()*



2. **Parseo:** Luego de la validación el programa hace una nueva pasada por el archivo y comienza a pedir una determinada cantidad de memoria dinámica para la creación de una estructura, denominada dentro del programa como *raw\_t*, la cual será cargada por los valores del archivo y usada por las clases internas del programa. Éste es el punto final del flujo de entrada, pasada esta función se pierde el concepto de archivo. El parseo está implementado con el método *parse()*
3. **Conversión :** Una vez terminada la lógica principal la clase recibirá una lista de bloques (en este caso, una lista de 1 bloque) y comenzará a pasar todo lo obtenido en la lista al archivo de salida terminando con el objetivo del programa. Implementado con el método *convert()*

El diseño de esta clase trajo un interrogante y una relación de compromiso. Se optó por realizar dos pasadas al archivo, una correspondiente a la validación y otra al *parseo*. Si bien esta solución no es eficiente, puesto que se podrían realizar ambas tareas en una única pasada, existe una cierta ganancia en desacoplar las funciones. Una de ellas se corresponde con la idea de que las listas de transacciones pueden ser (muy) largas y resultaría inconveniente para el minado que se solicite una gran cantidad de memoria (que además no se utilizaría en otros recursos) para luego encontrarse con que no es necesaria puesto que el archivo tiene fallas. En cuyo caso se tendría que comenzar a liberar dicha memoria. Ésto se traduciría en un código si bien eficiente, largo y tedioso. Por ende en pos de la legibilidad, mantenibilidad y de la idea de que sólo se demande memoria dinámica cuando realmente sea necesaria, se optó por realizar dos pasadas.

```

1 #include "BlockChainFileManager.h"
2
3 // BlockChainFileManager::fileList es una lista estatica
4 // visible para cualquier instancia de filemanager.
5 // En esta se encuentran todos los archivos abiertos y operados
6 // por filemanager. BlockChainFileManager::fileList solo puede contener
7 // un archivo de cada type.
8 lista<file_t *> BlockChainFileManager::fileList;
9
10
11 // Descripcion: Constructor de un objeto filemanager
12 // Precondicion:
13 // PostCondicion: Atributos inicializado a valores adecuados.
14 BlockChainFileManager::BlockChainFileManager() {
15     this->pRawData = NULL;
16 }
17
18 // Descripcion: Destructor de un objeto filemanager
19 // Precondicion: Si el objeto pidio memoria dinamica, debe liberarse y
20 // cargarse a valores adecuados.
21 // PostCondicion:
22 BlockChainFileManager::~BlockChainFileManager() {
23     if(this->pRawData != NULL){
24         pRawData->inTx = 0;
25         delete [] pRawData->IN_tableOfTxId;           pRawData->IN_tableOfTxId = NULL;
26         delete [] pRawData->IN_tableOfIndex;           pRawData->IN_tableOfIndex = NULL;
27         delete [] pRawData->IN_tableOfAddr;            pRawData->IN_tableOfAddr = NULL;
28         pRawData->outTx = 0;
29         delete [] pRawData->OUT_tableOfValues;         pRawData->OUT_tableOfValues = NULL;
30         delete [] pRawData->OUT_tableOfAddr;           pRawData->OUT_tableOfAddr = NULL;
31         delete pRawData;
32         pRawData = NULL;
33     }
34     if ( ! this->userBlockChain.vacia() ) {
35         lista <Block *>::iterador it( this->userBlockChain );
36         it = this->userBlockChain.primerero();
37         while ( ! this->userBlockChain.isEmpty() ) {
38             delete it.dato();
39             this->userBlockChain.eliminar_nodo(it);
40         }
41     }
42 }
43
44 // Descripcion: Devuelve en un puntero a fstream pasado como argumento el valor de
45 // fs correspondiente al file_t de la lista de archivos.

```

```

46 // Precondicion: type debe ser pasado correctamente.
47 // PostCondicion: Devuelve true/false si encuentra o no dicho file_t en la lista.
48 bool BlockchainFileManager::getFilefromList(FileTypes type, std::fstream ** fs){
49     lista<file_t *>::iterador it(BlockChainFileManager::fileList);
50     while(! it.extremo())
51     {
52         if(it.dato()->type == type){
53             (*fs) = & (it.dato()->fs);
54             return true;
55         }
56         it.avanzar();
57     }
58     return false;
59 }
60
61 // Descripcion: Devuelve en un puntero a istream pasado como argumento el valor de
62 // iss correspondiente al file_t de la lista de archivos.
63 // Precondicion: type debe ser pasado correctamente.
64 // PostCondicion: Devuelve true/false si encuentra o no dicho file_t en la lista.
65 bool BlockchainFileManager::getIssfromList(FileTypes type, std::istream ** iss){
66     lista<file_t *>::iterador it(BlockChainFileManager::fileList);
67     while(! it.extremo())
68     {
69         if(it.dato()->type == type){
70             (*iss) = it.dato()->iss;
71             return true;
72         }
73         it.avanzar();
74     }
75     return false;
76 }
77
78 // Descripcion: Devuelve en un puntero a ostream pasado como argumento el valor de
79 // oss correspondiente al file_t de la lista de archivos.
80 // Precondicion: type debe ser pasado correctamente.
81 // PostCondicion: Devuelve true/false si encuentra o no dicho file_t en la lista.
82 bool BlockchainFileManager::getOssfromList(FileTypes type, std::ostream ** oss){
83     lista<file_t *>::iterador it(BlockChainFileManager::fileList);
84     while(! it.extremo())
85     {
86         if(it.dato()->type == type){
87             (*oss) = it.dato()->oss;
88             return true;
89         }
90         it.avanzar();
91     }
92     return false;
93 }
94
95
96
97 // Descripcion: Devuelve en un puntero a ostream pasado como argumento el valor de
98 // oss correspondiente al file_t de la lista de archivos.
99 // Precondicion: type debe ser pasado correctamente.
100 // PostCondicion: Devuelve true/false si encuentra o no dicho file_t en la lista.
101 bool BlockchainFileManager::getIsStandarfromList(FileTypes type){
102     lista<file_t *>::iterador it(BlockChainFileManager::fileList);
103     while(! it.extremo())
104     {
105         if(it.dato()->type == type){
106             return it.dato()->isStandard;
107         }
108         it.avanzar();
109     }
110     return false;
111 }
112
113 // Descripcion: Es funcion hace una traduccion del lenguaje de archivo a un tipo de dato
114 // conocido
115 // por el sistema (payload_t). Basicamente sera llamada siempre que haya comandos por ingresar

```

```

    o si hay un txt con comandos
115 // y los parseara en un tipo de dato payload_t. Este tipo de dato permite limitar el alcance
    del archivo solo
116 // al scope de FileManager puesto que pasada esta funcion no existe el concepto de archivo y
    se utiliza el de payload_t.
117 // Precondicion: Se le da un payload que debe ser inicializado a valores seguros previo a la
    conversion.
118 // PostCondicion: Si los comandos son validos debe rellenar el payload.
119 status_t BlockchainFileManager::translateCommands( payload_t & payload){
120     std::istream * iss;
121     //Obtengo el archivo de comandos de la lista de archivos
122     if( ! this->getIssfromList (FileTypes::userCommandInputFiles,&iss)) return
        STATUS_BAD_READ_INPUT_FILE;
123
124     std::string command,line;
125     int subStringNum = 1;
126     bool state;
127     bool eof = false;
128     Commands commandType = Commands::commandNotDefined;
129
130     //Preinicializo el pyaload en valores seguros
131     this->safeValuePayload(payload);
132
133     //Detecto de la primer linea, solo el comando
134     if (std::getline(*iss,line).eof() ) eof = true;
135
136     //Compatibilidad de archivos de Windows
137     if(line.back() == '\r') line.pop_back();
138
139     //Encuentro el primer delimitador de comando
140     command = getSubString(line, ' ', subStringNum++, &state);
141
142     //Valido y obtengo el comando
143     if( !state ) return STATUS_ERROR_COMMAND_PAYLOAD;
144     if( !isOnValidCommandTable(command,commandType) ) return STATUS_ERROR_COMMAND_NOT_FOUND;
145
146     //Completo el payload en base al comando
147     switch(commandType)
148     {
149         case Commands::init:
150             {
151                 //std::cout<< "init" << std::endl;
152                 std::string usr,value,bits;
153
154                 usr = getSubString(line, ' ', subStringNum++, &state); if( !state )
155                 return STATUS_ERROR_COMMAND_PAYLOAD;
156                 value = getSubString(line, ' ',subStringNum++, &state); if( !state )
157                 return STATUS_ERROR_COMMAND_PAYLOAD;
158                 bits = getSubString(line, ' ', subStringNum++, &state); if( !state )
159                 return STATUS_ERROR_COMMAND_PAYLOAD;
160
161                 //-----Debug -----//
162                 //std::cout<< usr << std::endl;
163                 //std::cout<< value << std::endl;
164                 //std::cout<< bits << std::endl;
165                 if (! isPositiveIntNumberFromString(value)) return
166                 STATUS_ERROR_COMMAND_PAYLOAD;
167                 if (! isPositiveIntNumberFromString(bits)) return
168                 STATUS_ERROR_COMMAND_PAYLOAD;
169
170                 payload.command = commandType;
171                 payload.user = usr;
172                 payload.value = std::stof(value);
173                 payload.bits = std::stoi(bits); ;
174             }
175         break;
176         case Commands::transfer:
177             {
178                 //std::cout<< "transfer" << std::endl;
179                 std::string src,dest,value;

```

```

175 //size_t inTx = 1 ;
176 //size_t outTx = 0;
177
178 payload.ArgTransfer = new Queue<string>;
179 this->argBuffer = payload.ArgTransfer;
180
181 bool MoreParameters;
182
183 src = getSubString(line, ' ', subStringNum++, &state); if( !state ) return
STATUS_ERROR_COMMAND_PAYLOAD;
184 payload.ArgTransfer->enqueue(src);
185 // Esta pasada valida, acumula los datos y cuenta la cantidad de destinos
186 // Para luego pedir memoria dinamica con datos veraces.
187 do{
188     dest = getSubString(line, ' ', subStringNum++, &MoreParameters); if (!
MoreParameters) break;
189     value = getSubString(line, ' ', subStringNum++, &state);
190
191     if( !state ) return
STATUS_ERROR_COMMAND_PAYLOAD;
192     if (! isPositiveFloatNumberFromString(value)) return
STATUS_ERROR_COMMAND_PAYLOAD;
193
194     payload.ArgTransfer->enqueue(dest);
195     payload.ArgTransfer->enqueue(value);
196 }while( MoreParameters );
197
198 // Comienzo a rellenar el payload
199
200 //Creo el archivo raw_t en el entorno del filemanager
201 // this->pRawData = new raw_t{0};
202 // if(pRawData == NULL) return STATUS_BAD_ALLOC;
203 // pRawData->inTx = inTx;
204 //
205 // pRawData->IN_tableOfTxId = new std::string[pRawData->inTx];
206 // pRawData->IN_tableOfIndex = new int[pRawData->inTx];
207 // pRawData->IN_tableOfAddr = new std::string[pRawData->inTx];
208 //
209 // if( pRawData->IN_tableOfTxId == NULL ||
210 //     pRawData->IN_tableOfIndex == NULL ||
211 //     pRawData->IN_tableOfAddr == NULL ) return STATUS_BAD_ALLOC;
212 //
213 // for(int i = 0; i < pRawData->inTx; i++)
214 // {
215 //     // El outpoint en esta instancia no puede parsearse puesto que como
comando
216 //     // no se tiene esa informacion. Tampoco se tiene un hash de la direccion
por
217 //     // se envia en el addr el string del nombre.
218 //     pRawData->IN_tableOfTxId[i] = "";
219 //     pRawData->IN_tableOfIndex[i] = -1;
220 //     pRawData->IN_tableOfAddr[i] = argQueue.dequeue();
221 // }
222 //
223 // pRawData->outTx = outTx;
224 // pRawData->OUT_tableOfValues = new float[pRawData->outTx];
225 // pRawData->OUT_tableOfAddr = new std::string[pRawData->outTx];
226 //
227 // if( pRawData->OUT_tableOfValues == NULL ||
228 //     pRawData->OUT_tableOfAddr == NULL ) return STATUS_BAD_ALLOC;
229 //
230 // for(int i = 0; i < pRawData->outTx; i++)
231 // {
232 //
233 //     pRawData->OUT_tableOfAddr[i] = argQueue.dequeue();
234 //     pRawData->OUT_tableOfValues[i] = std::stof( argQueue.dequeue() );
235 // }
236 //
237 // // Como el Raw data no esta completamente relleno puesto
238 // // que no se tiene toda la data necesaria para el minado

```

```

239 //          // se levanta el completeFlag en falso
240 //          pRawData->completeFlag = false;
241          payload.command = commandType;
242          //payload.pRaw = pRawData;
243      }
244      break;
245      case Commands::mine:
246      {
247          //std::cout<< "mine" << std::endl;
248          std::string bits;
249
250          bits = getSubString(line, ' ', subStringNum++, &state);
251
252          if( !state )                      return
STATUS_ERROR_COMMAND_PAYLOAD;
253          if (!isPositiveIntNumberFromString(bits)) return
STATUS_ERROR_COMMAND_PAYLOAD;
254
255          //-----Debug -----//
256          //std::cout<< bits << std::endl;
257          payload.command = commandType;
258          payload.bits = std::stoi(bits);
259
260      }
261      break;
262      case Commands::block:
263      {
264          //std::cout<< "block" << std::endl;
265          std::string id;
266          id = getSubString(line, ' ', subStringNum++, &state);
267
268          //-----Debug -----//
269          //std::cout<< id << std::endl;
270          if( !state )                      return STATUS_ERROR_COMMAND_PAYLOAD;
271          if( ! this->isHashFromString(id) ) return STATUS_ERROR_COMMAND_PAYLOAD;
272
273          payload.command = commandType;
274          payload.id = id;
275      }
276      break;
277      case Commands::balance:
278      {
279          //std::cout<< "balance" << std::endl;
280          std::string usr;
281          usr = getSubString(line, ' ', subStringNum++, &state);
282
283          if( !state )                      return STATUS_ERROR_COMMAND_PAYLOAD;
284
285          //-----Debug -----//
286          //std::cout<< usr << std::endl;
287
288          payload.command = commandType;
289          payload.user = usr;
290      }
291      break;
292      case Commands::txn:
293      {
294          //std::cout<< "txn" << std::endl;
295          std::string id;
296          id = getSubString(line, ' ', subStringNum++, &state);
297
298          //-----Debug -----//
299          //std::cout<< id << std::endl;
300          if( !state )                      return STATUS_ERROR_COMMAND_PAYLOAD;
301          if( !isHashFromString(id) )      return STATUS_ERROR_COMMAND_PAYLOAD;
302
303          payload.command = commandType;
304          payload.id = id;
305      }
306      break;

```

```

307     case Commands::load:
308     {
309         //std::cout<< "load" << std::endl;
310         std::string filename;
311         filename = getSubString(line, ' ', subStringNum++, &state);
312
313         //-----Debug -----//
314         //std::cout<< filename << std::endl;
315
316         payload.command = commandType;
317         payload.filename = filename;
318     }
319     break;
320     case Commands::save:
321     {
322         //std::cout<< "save" << std::endl;
323         std::string filename;
324         filename = getSubString(line, ' ', subStringNum++, &state);
325
326         //-----Debug -----//
327         //std::cout<< filename << std::endl;
328
329         payload.command = commandType;
330         payload.filename = filename;
331     }
332     break;
333     default:
334         break;
335 }
336
337 // Evitar problema de ultima linea con \n y getline en archivos
338 if (!this->getIsStandarfromList (FileTypes::userCommandInputFiles)){
339     static unsigned int lastCharpos;
340     static bool hasEndOfLine = this->hasNewLineAtTheEnd(iss,&lastCharpos);
341     if( hasEndOfLine){
342         // cout<<lastCharpos<<endl;
343         // cout<<iss->tellg()<<endl;
344         if(abs(lastCharpos - iss->tellg() ) == 1){
345             return STATUS_NO_MORE_COMMANDS;
346         }
347     }
348 }
349 if( eof == true)
350     return STATUS_NO_MORE_COMMANDS;
351 return STATUS_READING_COMMANDS;
352 }
353
354 bool BlockChainFileManager::hasNewLineAtTheEnd(std::istream *iss,unsigned int * pos){
355     unsigned int prevPos = iss->tellg();
356     iss->seekg(-1,ios_base::end);    // go to one position before the EOF
357     if(pos) *pos = iss->tellg();
358     char c;
359     iss->get(c);                    // Read current character
360     iss->seekg(prevPos,ios_base::beg);
361     if(c =='\n'){
362         return true;
363     }else
364         return false;
365 }
366
367
368
369 // Descripcion: Verifica si el comando esta en la tabla especificada de comandos y si lo
370 // encuentra
371 // devuelve en commandType el tipo enumerativo asociado a ese comando.
372 // Precondicion: CommandType se debe precargar con algo no definido antes de buscar.
373 // PostCondicion: Devuelve verdadero o falso
374 bool BlockChainFileManager::isOnValidCommandTable(std::string command, Commands & commandType)
375 {
376     size_t i;

```

```

376     size_t size = BlockchainFileManager::getNumberOfValidFunctions();
377     commandType = Commands::commandNotDefined;
378     bool functionIsOnTable = false;
379     for(i = 0 ; i < size ; ++i)
380     {
381         if( command.compare(_commands_[i]) == 0 )
382         {
383             functionIsOnTable = true;
384             commandType = static_cast<Commands>(i);
385             break;
386         }
387     }
388
389     return functionIsOnTable;
390 }
391
392 // Descripcion:
393 // Precondicion:
394 // PostCondicion:
395 unsigned int BlockchainFileManager::getNumberOfValidFunctions()
396 {
397     return sizeof(_commands_)/sizeof(std::string);
398 }
399
400 // Descripcion: Metodo para cargar los valores por defecto del tipo de dato Payload_t
401 // Precondicion:
402 // PostCondicion: Todos los campos de Payload_t inicializados a valores correctos.
403 void BlockchainFileManager::safeValuePayload(payload_t & payload){
404     static bool firstTime = true;
405     if (!firstTime){
406         if(payload.ArgTranfer!= NULL){
407             delete payload.ArgTranfer;
408         }
409     }
410     else
411         firstTime = false;
412     payload.ArgTranfer = NULL;
413     payload.command = Commands::commandNotDefined;
414     //payload.pRaw = NULL;
415     payload.filename = "";
416     payload.id = "";
417     payload.user = "";
418     payload.value = 0;
419     payload.bits = 0;
420 }
421
422
423 // Descripcion: Este metodo recibe un tipo de dato file_t, abre el archivo asociado y lo
424 // agrega a la lista
425 // de file_t de FileManager. Debe ser usado en conjunto con removeFile o removeAllFiles en el
426 // momento de
427 // terminar el trabajo de fileManager.
428 // Precondicion: Tiene que recibir un file_t incompleto pero con los campos filename, type y
429 // mode completos
430 // PostCondicion: Archivo abierto con los campos del file_t correctamente inicializados iss,
431 // oss, fs
432 status_t BlockchainFileManager::addFile( file_t & newFile ){
433
434     file_t * pFile = new file_t;
435     pFile->fileID = newFile.fileID;
436     pFile->mode = newFile.mode;
437     pFile->type = newFile.type;
438     pFile->isStandard = newFile.isStandard;
439
440     if( pFile->mode == ios::in){
441         if(pFile->isStandard ){
442             pFile->iss = & cin;
443         }
444     }

```

```

442     else{
443         pFile->fs.exceptions ( std::ifstream::failbit | std::ifstream::badbit );
444         try {
445             pFile->fs.open(pFile->fileID,pFile->mode);
446         }catch (std::system_error& e) {
447             std::cerr << "Exception opening/reading/closing file error: " << e.code().
message() << "\n";
448             delete pFile;
449             return STATUS_BAD_READ_INPUT_FILE;
450         }
451         pFile->iss = & pFile->fs;
452     }
453     pFile->oss = NULL;
454 }
455 else if( pFile->mode == ios::out){
456     if(pFile->isStandard){
457         pFile->oss = & cout;
458     }
459     else{
460         pFile->fs.exceptions ( std::ifstream::failbit | std::ifstream::badbit );
461         try {
462             pFile->fs.open(pFile->fileID,pFile->mode);
463         }catch (std::system_error& e) {
464             std::cerr << "Exception opening/reading/closing file error: " << e.code().
message() << "\n";
465             delete pFile;
466             return STATUS_BAD_READ_INPUT_FILE;
467         }
468         pFile->oss = & pFile->fs;
469     }
470     pFile->iss = NULL;
471 }
472 }
473
474
475 fileList.insertar(pFile);
476 return STATUS_OPEN_FILE_SUCCESSFULLY;
477 }
478
479 // Descripcion: Destruye el elemento type de la lista de archivos, liberando memoria
480 // y cerrando el archivo asociado con el tipo type.
481 // Este metodo debe invocarse siempre que se uso el metodo AddNewFile.
482 // Precondicion: Debe usarse al final de utilizar AddNewFile.
483 // PostCondicion: Elemento de la lista liberado, archivo del del elemento cerrado
484 // y variables post inicializadas.
485
486 status_t BlockChainFileManager::removeFile(FileTypes type){
487     lista<file_t *>::iterador it(BlockChainFileManager::fileList);
488     while(!it.extremo()){
489         if(it.dato()->type == type){
490             it.dato()->fileID = "";
491             it.dato()->iss = NULL;
492             it.dato()->oss = NULL;
493             it.dato()->isStandard = NULL;
494             it.dato()->type = FileTypes::undefinedFile;
495             if (! it.dato()->isStandard ){
496                 it.dato()->fs.close();
497             }
498             delete it.dato();
499             fileList.eliminar_nodo(it);
500             break;
501         }
502         it.avanzar();
503     }
504     // TODO IMPLEMENTAR UN METODO MEJOR DE BORRADO PUESTO QUE ESTE NO
505     // OBEDECE LA DIRECCION DEL DATO
506
507     return STATUS_CLOSE_FILE_SUCCESSFULLY;
508 }
509

```



```

510 // Descripcion: Destruye la lista de file_t del FileManager, liberando toda la memoria pedida
511 // cerrando los archivos abiertos y postinicializando los valores del file_t a valores nulos.
512 // Este metodo debe invocarse siempre que se uso el metodo AddNewFile.
513 // Precondicion: Debe usarse al final de utilizar AddNewFile.
514 // PostCondicion: Lista completamente liberada, todos los archivos que fueron abiertos
    cerrados
515 // y variables post inicializadas.
516 status_t BlockChainFileManager::removeAllFiles(){
517     lista<file_t *>::iterador it(BlockChainFileManager::fileList);
518     it = fileList.primer();
519     while(!fileList.isEmpty()){
520         it.dato()->fileID = "";
521         it.dato()->iss = NULL;
522         it.dato()->oss = NULL;
523         it.dato()->type = FileTypes::indefinedFile;
524         if (! it.dato()->isStandard ){
525             it.dato()->fs.close();
526         }
527         delete it.dato();
528         fileList.eliminar_nodo(it);
529     }
530     return STATUS_CLOSE_FILE_SUCCESSFULLY;
531 }
532
533 static FileTypes GlobalType = FileTypes::userCommandResponseFiles;
534
535
536 // Descripcion: Sobrecarga del operador << para tipos FileTypes. Este metodo
537 // carga la variable global GlobalType con el tipo de archivo donde se desea
538 // imprimir el mensaje y luego junto la sobrecarga de << para strings imprime
539 // el mensaje.
540 // Precondicion:
541 // PostCondicion:Variable global cargada con el valor type pasada como argumentos
542 BlockChainFileManager& BlockChainFileManager::operator<<(FileTypes type){
543     GlobalType = type;
544     return *this;
545 }
546
547
548 // Descripcion: Sobrecarga del operador << para strings. Este metodo imprime el mensaje
549 // en el archivo determinado por la variable global GlobalType. Por defecto los mensajes
550 // siempre se imprimen en el archivo con asociado al tipo FileTypes::userCommandResponseFiles.
551 // Sin embargo, puede cambiarse usando el metodo de sobrecarga << de Filetypes para cambiar
552 // el archivo de salida.
553 // Precondicion: Si no encuentra el archivo especificado no imprime nada.
554 // PostCondicion: Mensaje impreso en el archivo asociado al tipo definido por GlobalType
555 BlockChainFileManager& BlockChainFileManager::operator<<(std::string message){
556     std::ostream * oss;
557     if(! this->getOssfromList(GlobalType, &oss)) return *this;
558     *oss << message;
559     return *this;
560 }
561
562 // Descripcion:
563 // Precondicion:
564 // PostCondicion:
565 status_t BlockChainFileManager::convert(FileTypes type, const lista <Block *> & BlockChain){
566     ostream * oss;
567     switch(type){
568     case FileTypes::saveBlockChainFile:
569         if( !this->getOssfromList(type,&oss) ) return STATUS_BAD_READ_OUTPUT_FILE;
570         return this->convert(oss, BlockChain);
571         break;
572     case FileTypes::userCommandResponseFiles:
573         if( !this->getOssfromList(type,&oss) ) return STATUS_BAD_READ_OUTPUT_FILE;
574         return this->convert(oss, BlockChain);
575         break;
576     case FileTypes::userCommandInputFiles:
577     case FileTypes::loadBlockChainFile:
578     default:

```

```

579         return STATUS_NO_BLOCKS_TO_CONVERT;
580         break;
581     }
582     return STATUS_BAD_READ_OUTPUT_FILE;
583 }
584
585 status_t BlockChainFileManager::convert(FileTypes type, const lista <Transaction *> &
586     listaTran){
587     switch(type){
588     case FileTypes::saveBlockChainFile:
589         break;
590     case FileTypes::userCommandResponseFiles:
591         ostream * oss;
592         if( !this->getOssfromList( type,&oss) ) return STATUS_BAD_READ_OUTPUT_FILE;
593         return this->convert(oss, listaTran);
594         break;
595     case FileTypes::userCommandInputFiles:
596     case FileTypes::loadBlockChainFile:
597     default:
598         return STATUS_NO_BLOCKS_TO_CONVERT;
599         break;
600     }
601     return STATUS_BAD_READ_OUTPUT_FILE;
602 }
603
604 // Descripcion:
605 // Precondicion:
606 // PostCondicion:
607 status_t BlockChainFileManager::convert(std::ostream * oss, const lista <Block *> & BlockChain
608     ){
609     lista <Block *> ::iterador it(BlockChain);
610     it = BlockChain.ultimo();
611
612     if(!oss->good()) return STATUS_BAD_READ_OUTPUT_FILE;
613     if( BlockChain.vacia() ) return STATUS_NO_BLOCKS_TO_CONVERT;
614     while(!it.extremo()){
615         *oss << it.dato()->getpre_block() << '\n';
616         *oss << it.dato()->gettxns_hash() << '\n';
617         *oss << it.dato()->getbits( ) << '\n';
618         *oss << it.dato()->getnonce() << '\n';
619         *oss << it.dato()->gettxn_count() << '\n';
620         *oss << it.dato()->getcadenaprehash();
621         it.retroceder();
622     }
623     return STATUS_FINISH_CONVERT_SUCCESSFULLY;
624 }
625
626 status_t BlockChainFileManager::convert(std::ostream * oss, const lista <Transaction *> &
627     listaTran){
628     lista <Transaction *> ::iterador it(listaTran);
629     it = listaTran.ultimo();
630
631     if(!oss->good()) return STATUS_BAD_READ_OUTPUT_FILE;
632     if( listaTran.vacia() ) return STATUS_NO_BLOCKS_TO_CONVERT;
633     while(!it.extremo()){
634         *oss << it.dato()->getConcatenatedTransactions();
635         it.retroceder();
636     }
637     return STATUS_FINISH_CONVERT_SUCCESSFULLY;
638 }
639
640 #define BLOCK_1_LINE_PREVIOUS_HASH 1
641 #define BLOCK_2_LINE_BLOCK_HASH 2
642 #define BLOCK_3_LINE_BITS 3
643 #define BLOCK_4_LINE_NONCE 4
644 #define BLOCK_5_LINE_CANT_TX 5
645 #define BLOCK_TRANSAC_TX_INPUT_INDEX 6
646 #define BLOCK_TRANSAC_INPUTS 7
647 #define BLOCK_TRANSAC_TX_OUTPUT_INDEX 8
648 #define BLOCK_TRANSAC_OUTPUTS 9

```

```

646
647 status_t BlockchainFileManager::validateBlockchain(void){
648     std::istream * iss;
649     if(! this->getIssfromList(FileTypes::loadBlockchainFile, &iss)) return
        STATUS_BAD_READ_INPUT_FILE;
650
651     std::string line,aux;
652     unsigned int blockLineCounter = 0, trasnferCounter = 0;
653     unsigned int inTx,outTx,cantTx;
654     bool isCompleteBlock = false;
655     bool isEof = false;
656
657
658     while( std::getline(*iss, line)){
659         // Verifico si llegue al fin de archivo
660         if ((*iss).eof()) {
661             isEof = true;
662         }
663         //Compatibilidad de archivos de Windows
664         if(line.back() == '\r')
665             line.pop_back();
666
667         blockLineCounter++;
668         switch(blockLineCounter){
669             case BLOCK_1_LINE_PREVIOUS_HASH:
670                 isCompleteBlock = false;
671                 if( this->isHashFromString(line) == false){ return
STATUS_CORRUPT_FORMAT_BAD_HASH;} break;
672             case BLOCK_2_LINE_BLOCK_HASH:
673                 if( this->isHashFromString(line) == false){ return
STATUS_CORRUPT_FORMAT_BAD_HASH;} break;
674             case BLOCK_3_LINE_BITS:
675                 if( this->isPositiveIntNumberFromString(line) == false){return
STATUS_CORRUPT_FORMAT_BAD_BITS;} break;
676             case BLOCK_4_LINE_NONCE:
677                 if( this->isPositiveIntNumberFromString(line) == false){return
STATUS_CORRUPT_FORMAT_BAD_NONCE;}break;
678             case BLOCK_5_LINE_CANT_TX:
679                 if( this->isPositiveIntNumberFromString(line) == false){return
STATUS_CORRUPT_FORMAT_BAD_NONCE;}
680                 cantTx = getPositiveNumberFromString(line);
681                 break;
682             case BLOCK_TRANSAC_TX_INPUT_INDEX:
683                 if( this->isPositiveIntNumberFromString(line) == false){return
STATUS_CORRUPT_FORMAT_BAD_TXIN;}
684                 inTx = getPositiveNumberFromString(line);
685                 if(inTx)
686                     break;
687                 blockLineCounter++;
688             case BLOCK_TRANSAC_INPUTS:
689                 if(inTx){
690                     trasnferCounter ++;
691                     if( this->isHashFromString(this->getSubString(line,' ', 1)) == false){
692                         return STATUS_CORRUPT_FORMAT_BAD_HASH;}
693                     if( this->isPositiveIntNumberFromString(this->getSubString(line,' ', 2)) ==
false){
694                         return STATUS_CORRUPT_FORMAT_BAD_TXINDEX;}
695                     if( this->isHashFromString(this->getSubString(line,' ', 3)) == false){
696                         return STATUS_CORRUPT_FORMAT_BAD_HASH;}
697                 }
698                 if (trasnferCounter != inTx) blockLineCounter--;
699                 else trasnferCounter = 0;
700                 break;
701             case BLOCK_TRANSAC_TX_OUTPUT_INDEX:
702                 if( this->isPositiveIntNumberFromString(line) == false){ return
STATUS_CORRUPT_FORMAT_BAD_TXOUT;}
703                 outTx = getPositiveNumberFromString(line);
704                 if (outTx) break;
705             case BLOCK_TRANSAC_OUTPUTS:
706                 if (outTx){

```

```

707         tranferCounter++;
708         if( this->isPositiveFloatNumberFromString(this->getSubString(line, ' ', 1)) ==
false){
709             return STATUS_CORRUPT_FORMAT_BAD_BTCVALUE;
710             if( this->isHashFromString(this->getSubString(line, ' ', 2)) == false){
711                 return STATUS_CORRUPT_FORMAT_BAD_HASH;
712             }
713             if (tranferCounter != outTx) blockLineCounter--;
714             else{
715                 if (cantTx ==1){
716                     tranferCounter = 0;
717                     blockLineCounter = 0;
718                     isCompleteBlock = true;
719                 }else{
720                     tranferCounter = 0;
721                     blockLineCounter = BLOCK_5_LINE_CANT_TX;
722                     cantTx--;
723                     return STATUS_ERROR_LIST_OF_TRANSACT_NOT_SUPPORTED;
724                 }
725             }
726             break;
727         }
728
729
730         // Evitar problema de ultima linea con \n y getline en archivos
731         if (!this->getIsStandarfromList(FileTypes::loadBlockChainFile)){
732             static unsigned int lastCharpos;
733             static bool hasEndOfLine = this->hasNewLineAtTheEnd(iss, &lastCharpos);
734             if( hasEndOfLine){
735                 // cout<<lastCharpos<<endl;
736                 // cout<<iss->tellg()<<endl;
737                 if(abs(lastCharpos - iss->tellg() ) == 1){
738                     if (isCompleteBlock)
739                         return STATUS_OK;
740                     else
741                         return STATUS_BAD_READ_INPUT_FILE;
742                 }
743             }
744         }
745
746         // Si se leyo un bloque completo y es el fin de archivo
747         if (isEof){
748             if (isCompleteBlock)
749                 return STATUS_OK;
750             else
751                 return STATUS_BAD_READ_INPUT_FILE;
752         }
753     }
754     return STATUS_OK;
755 }
756
757 status_t BlockChainFileManager::loadBlockChain(void){
758     std::istream * iss;
759     if(! this->getIssfromList(FileTypes::loadBlockChainFile, &iss)) return
STATUS_BAD_READ_INPUT_FILE;
760     iss->clear();
761     iss->seekg(0, iss->beg);
762
763     std::string line,aux;
764     unsigned int blockLineCounter = 0, tranferCounter = 0;
765     unsigned int inTx,outTx,cantTx;
766     bool isCompleteBlock = false;
767     bool isEof = false;
768     bool finishLoading = false;
769
770     while (!finishLoading){
771         Block * newBlock = new Block;
772         Transaction * newTrans = new Transaction;
773
774         while( std::getline(*iss, line)){

```

```

775 // Verifico si llegue al fin de archivo
776 if ((*iss).eof()) {
777     isEof = true;
778 }
779 //Compatibilidad de archivos de Windows
780 if(line.back() == '\r')
781     line.pop_back();
782
783 blockLineCounter++;
784
785 switch(blockLineCounter){
786 case BLOCK_1_LINE_PREVIOUS_HASH:
787     isCompleteBlock = false;
788     newBlock->setpre_block(line);
789     break;
790 case BLOCK_2_LINE_BLOCK_HASH:
791     newBlock->settxns_hash(line);
792     break;
793 case BLOCK_3_LINE_BITS:
794     newBlock->setbits(std::stoi(line));
795     break;
796 case BLOCK_4_LINE_NONCE:
797     newBlock->setnonce(std::stoi(line));
798     break;
799 case BLOCK_5_LINE_CANT_TX:
800     //newBlock->settxn_count(std::stoi(line));
801     cantTx = getPositiveNumberFromString(line);
802     break;
803 case BLOCK_TRANSAC_TX_INPUT_INDEX:
804     inTx = getPositiveNumberFromString(line);
805     for(unsigned int i = 0; i < inTx ;i++){
806         newTrans->addTransactionInput();
807     }
808     if(inTx)
809         break;
810     blockLineCounter++;
811 case BLOCK_TRANSAC_INPUTS:
812     if(inTx){
813         trasnferCounter ++;
814         newTrans->getTransactionInput(trasnferCounter)->setTxId(this->getSubString
815 (line,' ', 1));
816         newTrans->getTransactionInput(trasnferCounter)->setIdx(std::stoi(this->
817 getSubString(line,' ', 2)));
818         newTrans->getTransactionInput(trasnferCounter)->setAddr(this->getSubString
819 (line,' ', 3));
820     }
821     if (trasnferCounter != inTx) blockLineCounter--;
822     else trasnferCounter = 0;
823     break;
824 case BLOCK_TRANSAC_TX_OUTPUT_INDEX:
825     outTx = getPositiveNumberFromString(line);
826     for(unsigned int i = 0; i < outTx ;i++){
827         newTrans->addTransactionOutput();
828     }
829     if (outTx) break;
830 case BLOCK_TRANSAC_OUTPUTS:
831     if(outTx){
832         trasnferCounter ++;
833         newTrans->getTransactionOutput(trasnferCounter)->setValue(std::stof(this->
834 getSubString(line,' ', 1));
835         newTrans->getTransactionOutput(trasnferCounter)->setAddr(this->
836 getSubString(line,' ', 2));
837     }
838     if (trasnferCounter != outTx) blockLineCounter--;
839     else{
840         trasnferCounter = 0;
841         if (cantTx ==1){
842             blockLineCounter = 0;
843             isCompleteBlock = true;
844             newBlock->settransaction(newTrans);

```

```

840         this->userBlockChain.insertar(newBlock);
841     }else{
842         cantTx--;
843         newBlock->settransaction(newTrans);
844     }
845 }
846 break;
847 }
848 }
849
850 // Evitar problema de ultima linea con \n y getline en archivos
851 if (!this->getIsStandarfromList(FileTypes::loadBlockChainFile)){
852     static unsigned int lastCharpos;
853     static bool hasEndOfLine = this->hasNewLineAtTheEnd(iss,&lastCharpos);
854     if( hasEndOfLine){
855         // cout<<lastCharpos<<endl;
856         // cout<<iss->tellg()<<endl;
857         if(abs(lastCharpos - iss->tellg() ) == 1){
858             if (isCompleteBlock)
859                 return STATUS_OK;
860             else
861                 return STATUS_BAD_READ_INPUT_FILE;
862         }
863     }
864 }
865
866 // Si se leyo un bloque completo y es el fin de archivo
867 if (isEof){
868     if (isCompleteBlock){
869         finishLoading = true;
870         break;
871     }
872     else
873         return STATUS_BAD_READ_INPUT_FILE;
874 }else{
875     if (isCompleteBlock){
876         finishLoading = false;
877         break;
878     }
879 }
880
881 }
882
883 }
884
885 return STATUS_OK;
886 }
887
888
889 std::string BlockChainFileManager::getSubString(std::string line,size_t delim,unsigned int
890 substringNum,bool *pState){
891     //Encuentro el primer delimitador de comando
892     const unsigned int MAX_ALLOWED_SUBSTRING = 50;
893     if (substringNum > MAX_ALLOWED_SUBSTRING){
894         if( pState ) *pState = false;
895         return "";
896     }
897     size_t delimPos = line.find(delim);
898     if ( delimPos == std::string::npos){
899         if( pState ) *pState = false;
900         return "";
901     }
902     if (substringNum == 1){
903         if( pState ) *pState = true;
904         string aux = line.substr(0, delimPos);
905         return aux;
906     }
907     else{
908         for(unsigned int posCounter = 2; posCounter != MAX_ALLOWED_SUBSTRING ; posCounter++){
909             size_t delimPosNext = line.find(delim,delimPos+1);

```

```

909
910     if( posCounter == substringNum)
911     {
912         if ( delimPosNext == std::string::npos){
913             string aux = line.substr(delimPos+1);
914             if( pState ) *pState = true;
915             return aux;
916         }else{
917             string aux = line.substr(delimPos+1,delimPosNext - (delimPos+1));
918             if( pState ) *pState = true;
919             return aux;
920         }
921     }else{
922         if ( delimPosNext == std::string::npos){
923             if( pState ) *pState = false;
924             return "";
925         }
926         delimPos = delimPosNext;
927     }
928 }
929 }
930 if( pState ) *pState = false;
931 return "";
932 }
933
934 // Descripcion:
935 // Precondicion:
936 // PostCondicion:
937 bool BlockchainFileManager::isHashFromString(std::string line){
938 if ( line.size() != (size_t) LargoHash::LargoHashEstandar ) return false;
939 for (unsigned int i = 0; i < line.length(); ++i) {
940     if ( ! isxdigit ( line[i] ) ) return false;
941 }
942 return true;
943 }
944
945 // Descripcion:
946 // Precondicion:
947 // PostCondicion:
948 bool BlockchainFileManager::isPositiveIntNumberFromString(std::string line){
949     int IndexValue;
950     std::string::size_type sz;
951     std::stringstream ss;
952     ss.str(line);
953     if ((ss >> IndexValue).fail()) return false;
954     if (IndexValue < 0) return false;
955     IndexValue = std::stoi(line,&sz);
956     if( sz != line.size() ) return false;
957     return true;
958 }
959
960 // Descripcion:
961 // Precondicion:
962 // PostCondicion:
963 bool BlockchainFileManager::isPositiveFloatNumberFromString(std::string line){
964     float btcValue;
965     std::string::size_type sz;
966     std::stringstream ss;
967     ss.str(line);
968     if ((ss >> btcValue).fail()) return false;
969     if (btcValue < 0) return false;
970     btcValue = std::stof(line,&sz);
971     if( sz != line.size() ) return false;
972     return true;
973 }
974 // Descripcion:
975 // Precondicion:
976 // PostCondicion:
977 unsigned int BlockchainFileManager::getPositiveNumberFromString(std::string line){
978     unsigned int IndexValue;

```

```

979     std::stringstream ss;
980     ss.str(line);
981     ss >> IndexValue;
982     return IndexValue;
983 }
984
985
986 //-----//
987 // OBSOLETO PARA LA VERSION 2.1.1 (TP1)
988
989 // Descripcion:
990 // Precondicion:
991 // PostCondicion:
992 status_t BlockchainFileManager::validate(std::istream * iss){
993     if( ! this->isEmpty(iss)){
994         int inTxTotal,outTxTotal;
995         if( this->isTxIndexFromStream(iss,'\n',&inTxTotal) == false ) return
STATUS_CORRUPT_FORMAT_BAD_TXIN;
996         for(int inTx = 0 ; inTx < inTxTotal ; inTx++){
997             if( this->isHashFromStream(iss,' ') == false ) return
STATUS_CORRUPT_FORMAT_BAD_HASH;
998             if( this->isTxIndexFromStream(iss,' ') == false ) return
STATUS_CORRUPT_FORMAT_BAD_TXINDEX;
999             if( this->isHashFromStream(iss) == false ) return
STATUS_CORRUPT_FORMAT_BAD_HASH;
1000         }
1001         if( this->isTxIndexFromStream(iss,'\n',&outTxTotal) == false ) return
STATUS_CORRUPT_FORMAT_BAD_TXOUT;
1002         for(int outTx = 0 ; outTx < outTxTotal ; outTx++){
1003             if( this->isBTCValueFromStream(iss,' ') == false ) return
STATUS_CORRUPT_FORMAT_BAD_BTCVALUE;
1004             if( this->isHashFromStream(iss) == false ) return
STATUS_CORRUPT_FORMAT_BAD_HASH;
1005         }
1006         if( this->isEofFromStream(iss) == false ) return
STATUS_CORRUPT_FORMAT;
1007     }
1008     return STATUS_OK;
1009 }
1010
1011 // Descripcion:
1012 // Precondicion:
1013 // PostCondicion:
1014 bool BlockchainFileManager::isEmpty(std::istream * iss)
1015 {
1016     // PRECONDICION: ESTA FUNCION SOLO DEBE USARSE ANTES DE HACER
1017     // EL TRABAJO DEL ARCHIVO PUESTO QUE AL TERMINAR DEJA APUNTANDO
1018     // AL PRINCIPIO
1019
1020     bool empty;
1021     //Voy al final de File
1022     iss->seekg(0, ios::end);
1023     empty = (iss->tellg() == 0)? true: false;
1024     //Vuelvo al principio del File
1025     iss->clear();
1026     iss->seekg(0, iss->beg);
1027     return empty;
1028 }
1029
1030
1031 // Descripcion:
1032 // Precondicion:
1033 // PostCondicion:
1034 bool BlockchainFileManager::isTxIndexFromStream(std::istream *iss,char delim , int * pValue)
1035 {
1036     int IndexValue;
1037     std::string line;
1038     std::stringstream ss;
1039
1040     std::getline(*iss, line,delim);

```



```

1041
1042     bool state = this->isPositiveIntNumberFromString(line);
1043     //Debug
1044     //std::cout << line << std::endl;
1045     ss.str(line);
1046     ss >> IndexValue;
1047     if(pValue != NULL) *pValue = IndexValue;
1048     return state;
1049 }
1050
1051 // Descripcion:
1052 // Precondicion:
1053 // PostCondicion:
1054 bool BlockchainFileManager::isHashFromStream(std::istream *iss, char delim, std::string *
1055 pString)
1056 {
1057     std::string line;
1058     std::stringstream ss;
1059     std::getline(*iss, line, delim);
1060     if( line.back() == '\r'){ //Para portabilidad Linux - Window
1061         line.substr(0, line.size()-1);
1062     }
1063     bool state = this->isHashFromString(line);
1064     //Debug
1065     //std::cout << line << std::endl;
1066     if(pString != NULL) *pString = line;
1067     return state;
1068 }
1069
1070 // Descripcion:
1071 // Precondicion:
1072 // PostCondicion:
1073 bool BlockchainFileManager::isBTCValueFromStream(std::istream *iss, char delim, float * pFloat)
1074 {
1075     float floatValue;
1076     std::string line;
1077     std::stringstream ss;
1078
1079     std::getline(*iss, line, delim);
1080     ss.str(line);
1081     if ((ss >> floatValue).fail()) return false;
1082     if (floatValue < 0) return false;
1083     //Debug
1084     //std::cout << line << std::endl;
1085     if(pFloat != NULL) *pFloat = floatValue;
1086     return true;
1087 }
1088
1089 // Descripcion:
1090 // Precondicion:
1091 // PostCondicion:
1092 bool BlockchainFileManager::isEofFromStream(std::istream *iss){
1093     std::string line;
1094     if (iss->eof()) return true;
1095     try{
1096         if (std::getline(*iss, line, '\r').fail() ) return true;
1097     }catch(const std::ios_base::failure& ex) {
1098         //std::cerr << "Caught: std::ios_base::failure" << std::endl;
1099         iss->clear();
1100         return true;
1101     }
1102     if (line.size() != 0 )
1103         return false;
1104     return true;
1105 }
1106
1107 // Descripcion:
1108 // Precondicion:
1109 // PostCondicion:
1110 status_t BlockchainFileManager::parse(std::istream * iss, raw_t * &pBuilderRawData){

```

```

1110
1111 //Creo el archivo raw_t en el entorno del filemanager
1112 this->pRawData = new raw_t{0};
1113 if ( ! this->isEmpty(iss)){
1114     if(pRawData == NULL) return STATUS_BAD_ALLOC;
1115     pRawData->inTx = this->getTxIndexFromStream(iss,'\n');
1116
1117     pRawData->IN_tableOfTxId = new std::string[pRawData->inTx];
1118     pRawData->IN_tableOfIndex = new int[pRawData->inTx];
1119     pRawData->IN_tableOfAddr = new std::string[pRawData->inTx];
1120     if(
1121         pRawData->IN_tableOfTxId == NULL ||
1122         pRawData->IN_tableOfIndex == NULL ||
1123         pRawData->IN_tableOfAddr == NULL ) return STATUS_BAD_ALLOC;
1124
1125     for(int i = 0; i < pRawData->inTx; i++)
1126     {
1127         pRawData->IN_tableOfTxId[i] = this->getHashFromStream(iss,' ');
1128         pRawData->IN_tableOfIndex[i] = this->getTxIndexFromStream(iss,' ');
1129         pRawData->IN_tableOfAddr[i] = this->getHashFromStream(iss);
1130     }
1131
1132     pRawData->outTx = this->getTxIndexFromStream(iss,'\n');
1133     pRawData->OUT_tableOfValues = new float[pRawData->outTx];
1134     pRawData->OUT_tableOfAddr = new std::string[pRawData->outTx];
1135     if(
1136         pRawData->OUT_tableOfValues == NULL ||
1137         pRawData->OUT_tableOfAddr == NULL ) return STATUS_BAD_ALLOC;
1138
1139     for(int i = 0; i < pRawData->outTx; i++)
1140     {
1141         pRawData->OUT_tableOfValues[i] = this->getBTCValueFromStream(iss,' ');
1142         pRawData->OUT_tableOfAddr[i] = this->getHashFromStream(iss);
1143     }
1144     pBuilderRawData = this->pRawData;
1145     return STATUS_OK;
1146 }
1147
1148 // Descripcion:
1149 // Precondicion:
1150 // PostCondicion:
1151 int BlockchainFileManager::getTxIndexFromStream(std::istream *iss,char delim)
1152 {
1153     int IndexValue;
1154     std::string line;
1155     std::stringstream ss;
1156
1157     std::getline(*iss, line,delim);
1158     ss.str(line);
1159     ss >> IndexValue;
1160     return IndexValue;
1161 }
1162
1163 // Descripcion:
1164 // Precondicion:
1165 // PostCondicion:
1166 std::string BlockchainFileManager::getHashFromStream(std::istream *iss,char delim)
1167 {
1168     std::string line;
1169     std::stringstream ss;
1170     std::getline(*iss, line,delim);
1171     return line;
1172 }
1173
1174 // Descripcion:
1175 // Precondicion:
1176 // PostCondicion:
1177 float BlockchainFileManager::getBTCValueFromStream(std::istream *iss,char delim)
1178 {
1179     float floatValue;

```

```

1180     std::string line;
1181     std::stringstream ss;
1182
1183     std::getline(*iss, line, delim);
1184     ss.str(line);
1185     ss >> floatValue;
1186     return floatValue;
1187 }

```

```

1
2  /*
3   * BlockchainFileManager.h
4   *
5   * Created on: 25 oct. 2020
6   * Author: Ramiro
7   */
8
9  #ifndef BLOCKCHAINFILEMANAGER_H_
10 #define BLOCKCHAINFILEMANAGER_H_
11
12 #include <iostream>
13 #include <sstream>
14 #include <ostream>
15 #include <fstream>
16 #include <string>
17 #include "BlockchainStatus.h"
18 #include "BlockchainBuilder.h"
19 #include "BlockchainDataTypes.h"
20 #include "TiposHash.h"
21 #include "Queue.h"
22 #include "lista.h"
23
24
25 class BlockchainFileManager {
26 private:
27
28     raw_t * pRawData;
29     Queue<std::string> * argBuffer;
30     lista<Block *> userBlockChain;
31     static lista<file_t *> fileList;
32
33     //-----Metodos sobre Streams ----- //
34     bool isEmpty(std::istream * iss);
35     bool isTxIndexFromStream(std::istream *iss, char delim = '\n', int * pValue = NULL);
36     bool isHashFromStream(std::istream *iss, char delim = '\n', std::string * pString = NULL);
37     bool isBTCValueFromStream(std::istream *iss, char delim = '\n', float * pFloat = NULL);
38     bool isEofFromStream(std::istream *iss);
39
40     int getTxIndexFromStream(std::istream *iss, char delim = '\n');
41     std::string getHashFromStream(std::istream *iss, char delim = '\n');
42     float getBTCValueFromStream(std::istream *iss, char delim = '\n');
43
44     bool hasNewLineAtTheEnd(std::istream * iss, unsigned int * pos = NULL);
45
46     // ---- Metodos sobre Strings -----//
47     bool isHashFromString(std::string line);
48     bool isPositiveIntNumberFromString(std::string line);
49     bool isPositiveFloatNumberFromString(std::string line);
50
51     unsigned int getPositiveNumberFromString(std::string line);
52
53     std::string getSubString(std::string line, size_t delim, unsigned int substringNum, bool *
54     pState = NULL);
55
56     // --- Metodos sobre lista de Archivos ---//
57     bool getFilefromList(FileTypes type, std::fstream ** fs);
58     bool getIssfromList(FileTypes type, std::istream ** iss);
59     bool getOssfromList(FileTypes type, std::ostream ** oss);

```

```

59     bool getIsStandarfromList(FileTypes type);
60
61     bool isValidCommandTable(std::string command, Commands & commandType);
62     unsigned int getNumberOfValidFunctions( void );
63
64 public:
65     BlockChainFileManager();
66     ~BlockChainFileManager();
67
68     lista <Block *> & getBlockChainPointer(){return userBlockChain;}
69
70     status_t validate(std::istream * iss);
71     status_t validateBlockChain( void );
72     status_t loadBlockChain(void);
73     status_t parse(std::istream * iss, raw_t * &pRawData);
74     status_t convert(std::ostream * oss, const lista <Block *> & BlockChain);
75     status_t convert(std::ostream * oss, const lista <Transaction *> & listaTran);
76     status_t convert(FileTypes type, const lista <Block *> & BlockChain);
77     status_t convert(FileTypes type, const lista <Transaction *> & listaTran);
78     status_t translateCommands(payload_t & payload);
79     status_t addFile( file_t & newFile);
80     status_t removeFile(FileTypes type);
81     status_t removeAllFiles();
82
83     BlockChainFileManager& operator<<(std::string message);
84     BlockChainFileManager& operator<<(FileTypes type);
85
86     void safeValuePayload(payload_t & payload);
87
88 };
89
90 #endif /* BLOCKCHAINFILEMANAGER_H_ */

```

## 4.8. Clase BlockChainBuilder

Esta clase es la encargada de instanciar *bloques* y es la única clase que opera con ellos. Por lo tanto, resulta vital acceder a los bloques a través de *getters* y *setters*. También contiene la lógica del cálculo del *nonce* y la utilización de la clase SHA256.

Dada la decisión de diseño tomada en el Tp0, la clase Builder prácticamente no registró cambios abruptos, sólo correcciones de errores. Se anexa lo escrito en el tp anterior.

*La ventaja de esta separación reside en que si se desea modificar la lógica (lo cuál será así en el siguiente trabajo práctico) se lo hará solamente en esta clase, dado que los conceptos de Bloques y Transacciones se mantendrán intactos.*

Las operaciones principales de *Builder* son la instanciación de los bloques, la carga de información en los mismos y la lógica del *proof of work*, a través del método *minando()*.

Dichas operaciones se implementan, en orden, en los métodos *createBlockChain()* y *Minando()*.

```

1
2
3
4 /*
5  * BlockChainBuilder.cpp
6  *
7  * Created on: 25 oct. 2020
8  * Author: Ramiro
9  */
10
11 #include <time.h>
12
13 #include "BlockChainBuilder.h"
14
15
16 BlockChainBuilder::BlockChainBuilder() :

```

```

17         BlocklActual(),
18         ListaBlocks(),
19         hash_resultado( "" ),
20         bits( ),
21         pRawData(NULL),
22         seconds()
23     {}
24
25     BlockChainBuilder::BlockChainBuilder(size_t d) :
26         BlocklActual(),
27         ListaBlocks(),
28         hash_resultado( "" ),
29         bits( d ),
30         pRawData(NULL),
31         seconds()
32     {}
33
34     BlockChainBuilder::~~BlockChainBuilder() {
35         if ( ! this->ListaBlocks.isEmpty() ) {
36             // lista <Transaction>::iterador it();
37             lista <Block *>::iterador it(ListaBlocks);
38             /* Itero la lista para recuperar todos los strings de la coleccion Transaction
39              que necesito para calcular el Hash.
40             */
41             it = this->ListaBlocks.primer();
42             while ( ! it.extremo() ) {
43                 if(it.dato() != NULL){
44                     delete it.dato();
45                     it.dato() = NULL;
46                 }
47                 it.avanzar();
48             }
49         }
50     }
51 }
52
53 //int BlockChainBuilder::CheckHexa( string value ) {
54 // unsigned int i;
55 // for ( i = 0; i != value.length(); ++i) {
56 //     if ( ! isxdigit ( value[i] ) ) break;
57 // }
58 // if ( i < value.length() ) return i;
59 // return 0;
60 //}
61 //
62 //bool BlockChainBuilder::CheckHash( std::string valor, TiposHash Tipo ) {
63 // if ( valor.empty() ) {
64 //     return false;
65 // }
66 // //else if ( Tipo == TiposHash::clavehash256 && valor.length() != LargoHashEstandar ) {
67 // else if ( Tipo == TiposHash::clavehash256 && valor.length() != (size_t) LargoHash::
68 //     LargoHashEstandar ) {
69 //     return false;
70 // }
71 // //else if ( Tipo == TiposHash::clavefirma && valor.length() != LargoHashFirma ) {
72 // else if ( Tipo == TiposHash::clavefirma && valor.length() != (size_t) LargoHash::
73 //     LargoHashFirma ) {
74 //     return false;
75 // }
76 // else {
77 //     int i = CheckHexa( valor );
78 //     if ( i > 0 ) {
79 //         // Anotar la posiciï¿½n y valor del díç¿gito errï¿½neo
80 //         return false;
81 //     }
82 //     else return true;
83 // }
84 //}
85
86 //unsigned int BlockChainBuilder::Calculononce() {

```

```

85 // static unsigned int contador = 0;
86 // contador++;
87 // return contador;
88
89 //}
90
91 bool BlockChainBuilder::CalculoBits( std::string hash, size_t bits ) {
92     int resultado;
93     if ( hash.length() > 0 ) {
94         std::string hash_hex = "";
95         hash_hex = BlockChainBuilder::hex_str_to_bin_str( hash ); // No lleva this-> porque
96     es static
97         resultado = BlockChainBuilder::CheckDificultadOk( hash_hex, bits );
98         if ( resultado == 1 ) {
99             return true;
100         }
101         else {
102             // Incluye cadena hash vacia y bits == 0
103             // Lo bueno de los booleanos es que siempre estas como mizximo a un bit de
104             acertar.
105             return false;
106         }
107     }
108     else {
109         return false;
110     }
111 }
112
113 bool BlockChainBuilder::Minando() {
114     std::string resultado;
115
116     if ( ! this->ListaBlocks.vacia() ) {
117         lista <Block *>::iterador it;
118         /* Itero la lista para recuperar todos los strings de la coleccion Transaction
119         que necesito para calcular el Hash.
120         */
121         it = this->ListaBlocks.primerero();
122         do {
123             time_t timer1, timer2;
124             time(&timer1); // get current time; same as: timer = time(NULL) */
125             this->BlocklActual = it.dato();
126             do{
127                 resultado = "";
128
129                 resultado += this->BlocklActual->getpre_block();
130                 resultado += '\n';
131                 resultado += this->BlocklActual->gettxns_hash();
132                 resultado += '\n';
133                 resultado += std::to_string(this->BlocklActual->getbits());
134                 resultado += '\n';
135                 resultado += this->BlocklActual->Calculononce();
136                 resultado += '\n';
137                 //std::cout << resultado << std::endl; //DEBUG
138                 //if ( resultado.length() > 0 ) {
139                 this->hash_resultado = sha256 ( sha256( resultado ) );
140                 //std::cout << this->hash_resultado << std::endl; //DEBUG
141                 //}
142                 }while(! CalculoBits( this->hash_resultado, this->bits ) );
143                 time(&timer2);
144                 this->BlocklActual->setseconds( difftime( timer2, timer1 ) );
145
146                 it.avanzar();
147             } while ( ! it.extremo() );
148             return true;
149         }
150         return false;
151     }
152 }
153
154 double BlockChainBuilder::tiempominado() {

```

```

153     return this->seconds;
154 }
155
156 const char* BlockchainBuilder::hex_char_to_bin( char c )
157 {
158     // TODO handle default / error
159     // https://stackoverflow.com/questions/18310952/convert-strings-between-hex-format-and-
160     // binary-format
161     switch( toupper(c) )
162     {
163         case '0': return "0000";
164         case '1': return "0001";
165         case '2': return "0010";
166         case '3': return "0011";
167         case '4': return "0100";
168         case '5': return "0101";
169         case '6': return "0110";
170         case '7': return "0111";
171         case '8': return "1000";
172         case '9': return "1001";
173         case 'A': return "1010";
174         case 'B': return "1011";
175         case 'C': return "1100";
176         case 'D': return "1101";
177         case 'E': return "1110";
178         case 'F': return "1111";
179         default: return "";
180     }
181 }
182
183 std::string BlockchainBuilder::hex_str_to_bin_str( const std::string & hex )
184 {
185     // TODO use a loop from <algorithm> or smth
186     std::string bin;
187     std::string hexbin;
188     for( size_t i = 0; i != hex.length(); ++i ) {
189         hexbin = hex_char_to_bin( hex[i] );
190         if ( hexbin.empty() ) return "";
191         bin += hexbin;
192     }
193     return bin;
194 }
195
196 int BlockchainBuilder::dificultad( const std::string & value, const size_t dif ) {
197     // Se corta el recorrido de la cadena una vez alcanzado el valor dif
198     size_t j = 0;
199
200     if ( value.empty() ) return -1;
201     else if ( dif == 0 ) return -1;
202
203     for ( size_t i = 0; value[ i ]; i++ ) {
204         if ( value[ i ] == '0' ) j++;
205         else if ( value[ i ] == '1' ) break;
206         else return -1;
207         if ( j >= dif ) break;
208     }
209     return j;
210 }
211
212 int BlockchainBuilder::CheckDificultadOk( const std::string & cadenaHexa, const size_t dif ) {
213     int d;
214     if ( cadenaHexa.empty() ) return -3;
215     if ( dif == 0 ) return -2;
216     d = dificultad( cadenaHexa, dif );
217     if ( d < 0 ) return -1;
218     return (size_t) d >= dif ? 1 : 0;
219 }
220
221 //status_t BlockchainBuilder::createBlockChain( void ){

```

```

222 // Block * newBlock = new Block(*pRawData);
223 // newBlock->setpre_block( "ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff"
224 // );
225 // newBlock->settxns_hash(sha256(sha256(newBlock->getcadenaprehash())));
226 // newBlock->setbits(this->bits);
227 // this->ListaBlocks.insertar(newBlock);
228 // this->Minando();
229 // return STATUS_OK;
230 //}
231
232 status_t BlockChainBuilder::createOriginBlock(Transaction& tr){
233     Block * newBlock = new Block(tr);
234     newBlock->setpre_block( "0000000000000000000000000000000000000000000000000000000000000000"
235     );
236     newBlock->settxns_hash( newBlock->gethash_Merkle());
237     newBlock->setbits(this->bits);
238     ListaBlocks.insertar(newBlock);
239     this->Minando();
240     return STATUS_OK;
241 }
242
243 status_t BlockChainBuilder::createBlock(lista<Transaction*> & mempoolList, std::string
244     previousHashBlock){
245     Block * newBlock = new Block(mempoolList);
246     newBlock->setpre_block( previousHashBlock);
247     newBlock->settxns_hash( newBlock->gethash_Merkle());
248     newBlock->setbits(this->bits);
249     ListaBlocks.insertar(newBlock);
250     this->Minando();
251     return STATUS_OK;
252 }

```

```

1
2
3 /*
4  * BlockChainBuilder.h
5  *
6  * Created on: 25 oct. 2020
7  * Author: Ramiro
8  */
9
10 #ifndef BLOCKCHAINBUILDER_H_
11 #define BLOCKCHAINBUILDER_H_
12
13 #include "TiposHash.h"
14 #include "BlockChainDataTypes.h"
15 #include "BlockChainStatus.h"
16 #include "Block.h"
17 #include "lista.h"
18 #include "sha256.h"
19
20
21 class BlockChainBuilder {
22 private: // Redundante pero más legible
23     // Anterior
24     //static int CheckHexa( std::string value ); // <- esta le será más útil a
25     // BlockChainFileManager
26     // Datos privados
27     Block * BlocklActual;
28     lista <Block *> ListaBlocks;
29     std::string hash_resultado;
30     size_t bits; // La dificultad de bits */
31     // Nuevo
32     raw_t * pRawData; // raw_t es el dato raw que devuelve filemanager. De aca builder saca
33     // los datos
34     static bool CalculoBits( std::string hash, size_t bits );
35     bool Minando();
36     static std::string hex_str_to_bin_str( const std::string & hex );

```



```

35     static const char* hex_char_to_bin( char c );
36     static int dificultad( const std::string & value, const size_t dif ); //
    -1 -> Error
37     // Para medir tiempos de minado x Block.
38     double seconds;
39 public:
40     BlockChainBuilder();
41     BlockChainBuilder(size_t d);
42     ~BlockChainBuilder();
43     // Getters
44     // size_t getbits();
45     std::string getObtainedHash(){return hash_resultado;};
46     raw_t *& getRawPointer(){return pRawData;};
47     lista <Block *> getBlockChainPointer(){return ListaBlocks;};
48     Block *& getBlocklActual(){return BlocklActual;};
49     double tiempominado();
50     // Setters
51     bool setbits( size_t valor );
52     // MÃ©todos
53     size_t cantidadBlocks(); // VS me canta que no se usa
54     static int CheckDificultadOk( const std::string & cadenaHexa, const size_t dif ); //
    Error -> < 0, No -> 0, Ok -> 1
55     static std::string Calculononce();
56     //status_t createBlockChain(void);
57     status_t createOriginBlock(Transaction &tr);
58     status_t createBlock(lista<Transaction*> & tr, std::string previousHashBlock);
59     // removidass?
60     //static bool CheckHash( const std::string valor, TiposHash Tipo = TiposHash::clavehash256
    );
61     //static size_t CheckHexa( const std::string value );
62 };
63
64 #endif /* BLOCKCHAINBUILDER_H_ */

```

## 4.9. Clase Block

La clase *Block* representa el nodo de *BlockChain* donde se guarda la información. Como se menciona, consta de métodos para su acceso destacando el método `getcadenaprehash()`, que permite obtener un **string** de todas las cadenas de transacciones concatenadas. Si bien la clase *Block* es en si un contenedor de datos (puesto que contienen una lista de transacciones) también es la única que interactúa con la clase **Transaction**. Con ésto se busca restringir el alcance de dicha clase a los límites de **Block**. Para la implementación de la clase **Block** se utilizó una implementación del TDA lista con iterador, puesto que es necesario que la clase maneje lista de Transacciones. Por otro lado, se trató de implementar bloques Try-Catch dentro de las zonas críticas para lograr tener un manejo de errores más controlado.

```
1
2
3 //Archivo fuente clase Block / AlgoBlock del tp0 para la materia 9512 Algoritmos y
  Programacion 2.
4
5 #include "Block.h"
6
7
8 // Constructores
9 Block::Block()
10 : pre_block(""), txns_hash(""), bits(3 /* El valor por default establecido en el TP0 */),
  nonce(0), eBlock(StatusBlock::BlockSinDatos), txn_count(0), CurTran(NULL)
11 // ver el #define DIFFICULTY_DEFAULT_VALUE 3
12 {
13     //this->ListaTran = NULL;
14     // this->CurTran = NULL;
15     // this->txn_count = 0;
16     // this->eBlock = StatusBlock::BlockSinDatos;
17 }
18
19 Block::Block( const raw_t & raw )
20 : pre_block(""), txns_hash(""), bits( 3 /* El valor por default establecido en el TP0 */),
  nonce(0), eBlock(StatusBlock::BlockSinDatos)
21 {
22     /* BÃ;sicamente:
23         se instancia un objeto Transaction, se asume que se reciben datos consistentes.
24         Se le transfiere en crudo el raw_t, (por ejemplo en el constructor directamente).
25         La clase Transaction luego deberÃ¡a instanciar los TransactionInput y
26         TransactionOutput correspondientes.
27         Y calcular al finalizar la carga de los objetos el string de resultado.
28         Al final se anade el objeto a ListaTran.
29         Dudas:
30         si en el txt se lee un Block que contiene varios Transaction, como los recibe
31         Block ?
32         En una lista lista.h o en un arreglo dinÃ¡mico vector.h raw_t?
33         En este caso se recibe solo un raw_t, igualmente lo cargo en una lista, para
34         hacerlo mÃ¡s genÃ©rico.
35     */
36     try {
37         this->CurTran = new Transaction( raw ); // <- Ojo, nuevo constructor
38         this->ListaTran.insertar( this->CurTran ); // Para el Constructor con un contenedor
39         de raw_t habrÃ¡ que iterar pasando el mismo tipo de parÃ¡metros al constructor de
40         Transaction
41         this->txn_count = 1; // Para el Constructor que recibe un
42         Contenedor, se incrementa en cada instancia nueva de Transaction
43         this->eBlock = StatusBlock::BlockPendienteCadena_prehash;
44         RecalculoHash();
45     }
46     catch (std::bad_alloc& ba)
47     {
48         this->eBlock = StatusBlock::BlockBadAlloc;
49         std::cerr << "bad_alloc caught: " << ba.what() << '\n';
50     }
51 }
52
53 // Destructor
```

```

48 Block::~Block() {
49     // ListaTran se autodestruye, antes debo liberar la memoria asignada en cada elemento *
    ListaTran de la lista
50     if ( ! this->ListaTran.vacia() ) {
51
52         lista <Transaction *>::iterador it(ListaTran);
53         /* Itero la lista para recuperar todos los strings de la coleccion Transaction
54            que necesito para calcular el Hash.
55         */
56         it = this->ListaTran.primer();
57         while ( ! it.extremo() ) {
58             delete it.dato();
59             it.avanzar();
60         }
61     }
62 }
63
64 // Getters
65 unsigned int Block::gettxn_count() {
66     return this->txn_count;
67 }
68
69 std::string Block::getpre_block() {
70     return this->pre_block;
71 }
72
73 std::string Block::gettxns_hash() {
74     return this->txns_hash;
75 }
76
77 unsigned int Block::getbits() {
78     return this->bits;
79 }
80
81 unsigned int Block::getnonce() {
82     return this->nonce;
83 }
84
85 std::string Block::getcadenaprehash() {
86     return this->cadena_prehash;
87 }
88
89 // Setters
90 bool Block::setpre_block( std::string valor ) {
91     if ( valor.empty() ) {
92         this->pre_block = "";
93         // Hay que anotar, en un status ?, el error o disparar un throw
94     }
95     else {
96         /* 1) Debo validar que sea una cadena de 32 bytes o 64 dÃgitos Hexa
97            2) Chequear que cada byte sea un caracter hexa vÃlido.
98            2) Chequear que cada byte sea un caracter hexa vÃlido. Se elimina se supone que
99               vien externamente validado.
100            if ( BlockchainBuilder::CheckHash( valor, TiposHash::clavehash256 ) ) {
101                this->pre_block = valor;
102            }
103            */
104         this->pre_block = valor;
105     }
106     return true;
107 }
108
109 bool Block::settxns_hash( std::string valor ) {
110     if ( valor.empty() ) {
111         this->txns_hash = "";
112         // Hay que anotar, en un status ?, el error o disparar un throw
113     }
114     else {
115         /* 1) Debo validar que sea una cadena de 32 bytes o 64 dÃgitos Hexa
116            2) Chequear que cada byte sea un caracter hexa vÃlido. Se elimina se supone que
117               viene externamente validado.

```

```

115         if ( BlockchainBuilder::CheckHash( valor, TiposHash::clavehash256 ) ) {
116             this->txns_hash = valor;
117         }
118         /*
119         this->txns_hash = valor;
120     }
121     return true;
122 }
123
124 bool Block::setbits( unsigned int valor ) {
125     if ( !valor ) {
126         this->bits = 0;
127         // Hay que anotar, en un status ?, el error o disparar un throw
128     }
129     else {
130         this->bits = valor;
131     }
132     return true;
133 }
134
135 bool Block::setnonce( int valor ) {
136     if ( valor < 0 ) {
137         this->nonce = 0;
138         // Hay que anotar, en un status ?, el error o disparar un throw
139     }
140     else {
141         /* No se valida nada, puede ser cualquier dato */
142         this->nonce = (unsigned int) valor;
143     }
144     return true;
145 }
146
147 bool Block::settransaction( const raw_t & raw ) {
148     try {
149         this->CurTran = new Transaction( raw ); // <- Ojo, nuevo constructor
150         this->ListaTran.insertar( this->CurTran ); // Para el Constructor con un contenedor
151         // de raw_t habr ; que iterar pasando el mismo tipo de par ;metros al constructor de
152         // Transaction
153         this->txn_count = 1; // Para el Constructor que recibe un
154         // Contenedor, se incrementa en cada instancia nueva de Transaction
155         this->eBlock = StatusBlock::BlockPendienteCadena_prehash;
156         RecalculoHash();
157         return true;
158     }
159     catch (std::bad_alloc& ba)
160     {
161         this->eBlock = StatusBlock::BlockBadAlloc;
162         std::cerr << "bad_alloc caught: " << ba.what() << '\n';
163         return false;
164     }
165 }
166
167 std::string Block::RecalculoHash( void ) {
168     std::string cadena = "";
169     if ( ! this->ListaTran.vacia() ) {
170         lista <Transaction *>::iterador it(ListaTran);
171         /* Itero la lista para recuperar todos los strings de la coleccion Transaction
172         que necesito para calcular el Hash.
173         */
174         it = this->ListaTran.primer();
175         while ( ! it.extremo() ) {
176             cadena += it.dato()->getConcatenatedTransactions();
177             it.avanzar();
178         }
179     }
180     if ( ! cadena.empty() ) {
181         this->cadena_prehash = cadena;
182         this->eBlock = StatusBlock::BlockCalculadoCadena_prehash;
183     }
184     else this->eBlock = StatusBlock::BlockPendienteCadena_prehash;

```

```

182     return cadena;
183 }

```

```

1
2 //Archivo fuente header clase Block / AlgoBlock del tp0 para la materia 9512 Algoritmos y
   Programacion 2.
3
4 #ifndef BLOCK_H_
5 #define BLOCK_H_
6
7 #include <cstdlib>
8 #include <string>
9 #include "lista.h"
10 #include "TiposHash.h"
11 #include "Transaction.h"
12
13 #include "BlockchainDataTypes.h"
14
15 // const size_t LargoHashEstandar = 64;
16 // const size_t LargoHashFirma = 40; // Hash Publica de la Cuenta
17 // https://stackoverflow.com/questions/2268749/defining-global-constant-in-c
18 // Análisis de Pro vs Contras contra #define y otras formas
19
20 using namespace std;
21
22 class Block {
23     private:
24         // Atributos Seccion Header
25         std::string pre_block;
26         std::string txns_hash; // <- retiene el hash256(hash256(cadena_prehash))
27         unsigned int bits; /* La dificultad de bits */
28         unsigned int nonce;
29         StatusBlock eBlock;
30         // Atributos Seccion Body;
31         unsigned int txn_count;
32         lista <Transaction *> ListaTran;
33         Transaction * CurTran;
34         std::string cadena_prehash;
35         // Métodos privados
36         std::string RecalculoHash( void );
37
38     public:
39         // Métodos
40         // Constructores
41         Block();
42         Block( const raw_t & raw );
43         //Block( const & std::string previo_block, size_t bits, const & raw_t );
44         // size_t bits sale de BlockchainManager::getUserDefinedDifficulty(void), pero
         referenciar a esta clase implica un encastramiento indeseado.
45         // Destructor
46         ~Block();
47         // Getters
48         unsigned int gettxn_count();
49         std::string getpre_block();
50         std::string gettxns_hash();
51         unsigned int getbits();
52         unsigned int getnonce();
53         std::string getcadenaprehash();
54         // Setters
55         bool setpre_block( std::string valor );
56         bool settxns_hash( std::string valor ); // Debo dejar el metodo de asignacion. El
         calculo Hash es externo al objeto block, no está encapsulado.
57         bool setbits( unsigned int valor );
58         bool setnonce( int valor ); // Debo dejar el método de asignacion. El calculo
         del Nonce es externo al objeto block, no esta encapsulado.
59         bool settransaction( const raw_t & raw ); // TODO
60         StatusBlock EstatusBlock();
61 };

```

```
62  
63 #endif /* BLOCK_H_ */
```

## 4.10. Clase Transaction

La clase *Transaction* al igual que la clase *Block* son clases contenedoras. La lógica mas importante de esta clase se encuentra en el constructor que recibe el *raw\_t* y en el método *getConcatenatedTransaction()*. El constructor al recibir el *raw* instancia nodos de memoria dinámica, para luego precargarlos con la información de *raw* e insertarlos en la lista correspondiente de entrada o salida. Como en esta zona también es necesario gran cantidad de memoria dinámica, se utilizaron bloques Try-Catch en caso de problemas de memoria, promoviendo la robustez del código. Cabe destacar que la implementación de las lista de inputs y Outputs fue creada en eligiendo como nodos a las clases *TransactionInput* y *TransactionOutput* respectivamente.

```
1  
2 /*  
3  * Transaction.cpp  
4  *  
5  * Created on: 25 oct. 2020  
6  * Author: Ramiro  
7  */  
8  
9  
10  
11 #include "Transaction.h"  
12  
13  
14 //---Constructores---//  
15  
16 //Descripcion: Instancia el objeto Transaction vacio  
17 //Precondicion: -  
18 //Postcondicion: La lista de transacciones de entrada y salida apuntan a NULL  
19 Transaction::Transaction() {  
20     this->n_tx_in = 0;  
21     this->n_tx_out = 0;  
22 }  
23  
24 //Descripcion: Instancia el objeto Transaction a partir de un archivo raw_t  
25 //Precondicion:  
26 //Postcondicion: Dos punteros a memoria de tamaño definido creados y  
27 // precargados con los datos de raw_t  
28 Transaction::Transaction( const raw_t & Raw ) {  
29     //TODO preparar Transaction para una cadena de Raw  
30     this->n_tx_in = Raw.inTx;  
31     for(int i = 0; i < this->n_tx_in ;i++)  
32     {  
33         try {  
34             TransactionInput * pTxInput = new TransactionInput;  
35             pTxInput->setTxId(Raw.IN_tableOfTxId[i]);  
36             pTxInput->setIdx(Raw.IN_tableOfIndex[i]);  
37             pTxInput->setAddr(Raw.IN_tableOfAddr[i]);  
38             this->ListaTranIn.insertar(pTxInput);  
39         }  
40         catch (std::bad_alloc& ba)  
41         {  
42             std::cerr << "bad_alloc caught: " << ba.what() << '\n';  
43         }  
44     }  
45     this->n_tx_out = Raw.outTx;  
46     for(int i = 0; i < this->n_tx_out ;i++ )  
47     {  
48         try {  
49             TransactionOutput * pTxOutput = new TransactionOutput;  
50             pTxOutput->setValue(Raw.OUT_tableOfValues[i]);  
51             pTxOutput->setAddr(Raw.OUT_tableOfAddr[i]);  
52             this->ListaTranOut.insertar(pTxOutput);  
53         }
```

```

54         catch (std::bad_alloc& ba)
55         {
56             std::cerr << "bad_alloc caught: " << ba.what() << '\n';
57         }
58     }
59 }
60
61
62 //Descripcion: Destruye elemento de Transaction
63 //Precondicion: Si se envia una transaccion nula no es necesario que se realice accion
64 //Postcondicion: Objeto destruido, memoria liberada, punteros a null y parametros a cero.
65 Transaction::~Transaction(){
66     if ( ! this->ListaTranIn.vacia() ) {
67         lista <TransactionInput *>::iterador it(ListaTranIn);
68         it = this->ListaTranIn.primer();
69         do{
70             delete it.dato();
71             it.avanzar();
72         }while ( ! it.extremo() );
73     }
74     if ( ! this->ListaTranOut.vacia() ) {
75         lista <TransactionOutput *>::iterador it(ListaTranOut);
76         it = this->ListaTranOut.primer();
77         do {
78             delete it.dato();
79             it.avanzar();
80         }while ( ! it.extremo() );
81     }
82 }
83
84 //---Getters---//
85
86 //Descripcion: Devuelve cantidad de transacciones de input
87 //Precondicion:
88 //Postcondicion:
89 int Transaction::getNumTransactionIn(){
90     return this->n_tx_in;
91 }
92
93 //Descripcion: Devuelve cantidad de transacciones de output
94 //Precondicion:
95 //Postcondicion:
96 int Transaction::getNumTransactionOut(){
97     return this->n_tx_out;
98 }
99
100 //Descripcion: Obtiene la transaccion de la lista de entradas
101 //Precondicion: Si el indice esta fuera de rango debe devolver null
102 //Postcondicion:
103 TransactionInput * Transaction::getTransactionInput(int index){
104     size_t index_ = (size_t)index;
105     if( index < 0 || index_ > this->ListaTranIn.tamano())
106         return NULL;
107     else{
108         lista <TransactionInput *>::iterador it(this->ListaTranIn);
109         int counter = 0;
110         while(counter != index){
111             it.avanzar();
112             counter++;
113         }
114         return it.dato();
115     }
116 }
117
118 //Descripcion: Obtiene la transaccion de la lista de salidas
119 //Precondicion: Si el indice esta fuera de rango debe devolver null
120 //Postcondicion:
121 TransactionOutput * Transaction::getTransactionOutput(int index){
122     size_t index_ = (size_t)index;
123     if( index < 0 || index_ > this->ListaTranOut.tamano())

```

```

124         return NULL;
125     else{
126         lista <TransactionOutput *>::iterador it(this->ListaTranOut);
127         int counter = 0;
128         while(counter != index){
129             it.avanzar();
130             counter++;
131         }
132         return it.dato();
133     }
134 }
135
136 //Descripcion: Devuelve un string de los valores concatenados de la listas
137 //para ser aplicado el hash correspondiente por fuera
138 //Precondicion: Se considera todo precargado antes
139 //Postcondicion:
140 std::string Transaction::getConcatenatedTransactions( void ){
141     lista <TransactionInput *>::iterador itIn(this->ListaTranIn);
142     lista <TransactionOutput *>::iterador itOut(this->ListaTranOut);
143     //std::ostringstream concatenation;
144     std::string concatenation;
145     // concatenation << this->n_tx_in << '\n';
146     concatenation += std::to_string( this->n_tx_in );
147     concatenation += '\n';
148     for(itIn = ListaTranIn.primer(); !itIn.extremo() ; itIn.avanzar()){
149         // concatenation<< itIn.dato()->getTxId() <<' ';
150         // concatenation<< itIn.dato()->getIdx() <<' ';
151         // concatenation<< itIn.dato()->getAddr() <<'\n';
152         concatenation += itIn.dato()->getTxId();
153         concatenation += ' ';
154         concatenation += std::to_string( itIn.dato()->getIdx() );
155         concatenation += ' ';
156         concatenation += itIn.dato()->getAddr();
157         concatenation += '\n';
158     }
159     concatenation += std::to_string( this->n_tx_out );
160     concatenation += '\n';
161     // concatenation << this->n_tx_out << '\n';
162     for(itOut = ListaTranOut.primer(); !itOut.extremo() ; itOut.avanzar()){
163         // concatenation<< itOut.dato()->getValue() <<' ';
164         // concatenation<< itOut.dato()->getAddr() <<'\n';
165         concatenation += float_to_string_w_precision( itOut.dato()->getValue() , 1 );
166         concatenation += ' ';
167         concatenation += itOut.dato()->getAddr();
168         concatenation += '\n';
169     }
170     //std::cout <<concatenation.str()<<std::endl; //DEBUG
171     //return concatenation.str();
172     return concatenation;
173 }
174
175
176
177 std::string Transaction::float_to_string_w_precision(float val, int p)
178 {
179     if( p < 0 ) return "";
180     p = (unsigned int) p;
181     std::stringstream stream;
182     stream << std::fixed << std::setprecision(p) << val;
183     return stream.str();
184 }

```

```

1
2
3 /*
4  * Transaction.h
5  *
6  * Created on: 25 oct. 2020

```



```

7  *      Author: Ramiro
8  */
9
10 #ifndef TRANSACTION_H_
11 #define TRANSACTION_H_
12
13 #include "TransactionInput.h"
14 #include "TransactionOutput.h"
15 #include "BlockchainDataTypes.h"
16 #include "lista.h"
17 #include <iostream>
18 #include <sstream>
19 #include <iomanip>
20 #include <cstdint> // Para NULL
21
22 class Transaction {
23 private:
24     int n_tx_in; // Indica cantidad total de inputs
25     lista <TransactionInput *> ListaTranIn; // Lista de inputs
26     int n_tx_out; // Indica cantidad total de outputs
27     lista <TransactionOutput *> ListaTranOut; // Lista de outputs
28
29     std::string float_to_string_w_precision(float value, int p);
30 public:
31     ///---Constructores---//
32     Transaction();
33     Transaction(int n_tx_in, int n_tx_out);
34     Transaction(const raw_t & raw);
35     ~Transaction();
36     ///---Getters---//
37     int getNumTransactionIn();
38     int getNumTransactionOut();
39     TransactionInput * getTransactionInput(int index);
40     TransactionOutput * getTransactionOutput(int index);
41     ///---Setters---//
42     ///---Otros---//
43     std::string getConcatenatedTransactions();
44 };
45
46 #endif /* TRANSACTION_H_ */

```

## 4.11. Clase TransactionInput

La clase *TransactionInput* opera como clase contenedora y soporte de *Transaction*. En si, es una clase que tiene la única función de ser instanciada por *Transaction* y luego ser nodo de ella. Se puede destacar la definición una estructura privada por dentro de la clase llamada *outpoint*. Esto realmente no cambia la lógica del programa pero permite tener una mayor comprensión puesto que jerarquiza y diferencia los atributos de la clase. Además, por encontrarse en la parte privada no puede definirse un outpoint por fuera sin que *TransactionInput* fuera instanciada.

```

1  #include "TransactionInput.h"
2
3  ///---Constructores---//
4
5  //Descripcion: Construye el objeto TransactionInput vacio
6  //Precondicion:
7  //Postcondicion: Todos los parametros iniciados en 0 o vacio
8  TransactionInput::TransactionInput() {
9      this->outpoint.idx = 0;
10     this->outpoint.tx_id = "";
11     this->addr = "";
12 }
13
14 //Descripcion: Destruye el objeto TransactionInput
15 //Precondicion:
16 //Postcondicion: Todos los parametros iniciados en 0 o vacio
17

```

```

18 //Los hashes no deben quedar en ninguna zona
19 TransactionInput::TransactionInput(){
20     this->outpoint.idx = 0;
21     this->outpoint.tx_id = "";
22     this->addr = "";
23 }
24
25     //---Getters---//
26
27 //Descripcion: Devuelve el parametro tx_id del outpoint
28 //Precondicion:
29 //Postcondicion:
30 const std::string TransactionInput::getTxId(void) const{
31     return this->outpoint.tx_id;
32 }
33
34 //Descripcion: Devuelve el parametro idx del outpoint
35 //Precondicion:
36 //Postcondicion:
37 int TransactionInput::getIdx(void) const{
38     return this->outpoint.idx;
39 }
40
41 //Descripcion: Devuelve el parametro addr
42 //Precondicion:
43 //Postcondicion:
44 const std::string TransactionInput::getAddr(void) const{
45     return this->addr;
46 }
47
48     //---Setters---//
49
50 //Descripcion: Carga el atributo tx_id
51 //Precondicion: Se asume validado previamente
52 //Postcondicion:
53 void TransactionInput::setTxId(std::string tx_id){
54     this->outpoint.tx_id = tx_id;
55 }
56
57
58 //Descripcion: Carga el atributo idx
59 //Precondicion: Se asume validado previamente
60 //Postcondicion:
61 void TransactionInput::setIdx(int idx){
62     this->outpoint.idx = idx;
63 }
64
65 //Descripcion: Carga el atributo addr
66 //Precondicion: Se asume validado previamente
67 //Postcondicion:
68 void TransactionInput::setAddr(std::string addr){
69     this->addr = addr;
70 }

```

```

1
2 #ifndef TRANSACTIONINPUT_H_
3 #define TRANSACTIONINPUT_H_
4
5 #include <string>
6
7 class TransactionInput {
8 private:
9     struct outpoint{
10         std::string tx_id;
11         int idx;
12     }outpoint;
13     std::string addr;
14 public:

```

```

15 //---Constructores---//
16 TransactionInput();
17 ~TransactionInput();
18 //---Getters---//
19 const std::string getTxId(void) const;
20 int getIdx(void) const;
21 const std::string getAddr(void) const;
22 //---Setters---//
23 void setTxId(std::string tx_id);
24 void setIdx(int idx);
25 void setAddr(std::string addr);
26 //---Otros---//
27 };
28
29 #endif /* TRANSACTIONINPUT_H_ */

```

## 4.12. Clase TransactionOutput

Al igual que TransactionInput, TransactionOutput opera como clase contenedora y a su mismo nivel. Siendo su única función operar como nodo para la clase Transaction y poseer los getter y setters adecuados para acceder a sus atributos. Con respecto a la implementacion

```

1
2
3 #include "TransactionOutput.h"
4
5 //---Constructores---//
6
7 //Descripcion: Construye el objeto TransactionOutput vacio
8 //Precondicion:
9 //Postcondicion: Atributos inicializados en cero o vacio
10 TransactionOutput::TransactionOutput() {
11     this->value = 0;
12     this->addr = "";
13 }
14
15 //Descripcion: Destruye el objeto TransactionOutput
16 //Precondicion:
17 //Postcondicion: Atributos en cero y strings vacios
18 TransactionOutput::~TransactionOutput() {
19     this->value = 0;
20     this->addr = "";
21 }
22 //---Getters---//
23
24 //Descripcion: Devuelve el valor de Value
25 //Precondicion:
26 //Postcondicion:
27 float TransactionOutput::getValue(void) const {
28     return this->value;
29 }
30
31 //Descripcion: Devuelve el arreglo de char del parametro addr
32 //Precondicion:
33 //Postcondicion: Debe ser un rvalue lo que devuelve
34 const std::string TransactionOutput::getAddr(void) const {
35     return this->addr;
36 }
37 //---Setters---//
38
39 //Descripcion: Carga el atributo value
40 //Precondicion: Se asume validado previamente
41 //Postcondicion:
42 void TransactionOutput::setValue(float value) {
43     this->value = value;
44 }
45 }

```

```

46
47 //Descripcion: Carga el atributo addr
48 //Precondicion: Se asume validado previamente
49 //Postcondicion:
50 void TransactionOutput::setAddr(std::string addr) {
51     this->addr = addr;
52 }

```

```

1
2 #ifndef TRANSACTIONOUTPUT_H_
3 #define TRANSACTIONOUTPUT_H_
4
5 #include <string>
6
7 class TransactionOutput {
8 private:
9     float value;
10    std::string addr;
11 public:
12    //---Constructores---//
13    TransactionOutput();
14    ~TransactionOutput();
15    //---Getters---//
16    float getValue(void) const;
17    const std::string getAddr(void) const;
18    //---Setters---//
19    void setValue(float value);
20    void setAddr(std::string addr);
21    //---Otros---//
22 };
23
24 #endif /* TRANSACTIONOUTPUT_H_ */

```

### 4.13. Clases sha256 cmdline

Las clases **sha256** y **cmdline** fueron provistas por la cátedra. La clase **sha256** es utilizada por la clase BlockchainBuilder para calcular y encontrar el hash correcto necesario para finalizar el ensamblaje de un bloque y unirlo a la Blockchain. La clase **cmdline** es utilizada por **main** para trabajar con los argumentos pasados por línea de comandos al programa.

## 5. Main

En si el main no es una clase, pero si en este entorno representa el ámbito del usuario. Como se aprecia en el código, no existe registro alguno de la lógica del proceso de Blockchain. Esto parte de la idea del encapsulamiento de la información. Como se observa, sólo el BlockchainManager está presente en este entorno. La analogía directa es: *¿Quién habla con el cliente?* el que mina el bloque, el que sabe de archivos, no lo habla el jefe. En este caso el BlockchainManager.

```

1
2 //=====
3 // Name      : main.cpp
4 // Author    :
5 // Version   : 2.1.1
6 // Description : Trabajo Practico Nro. 1
7 //=====
8
9
10 #include <iostream>
11 #include <fstream>
12 #include <sstream>
13 #include <string>

```

```

14 #include "cmdline.h"
15 #include "BlockchainManager.h"
16
17
18 using namespace std;
19
20
21 /*=====*/
22 //                                     PROTOTIPOS
23 /*=====*/
24
25
26 static void opt_input(string const &);
27 static void opt_output(string const &);
28
29
30
31 /*=====*/
32 //                                     ELEMENTOS GLOBALES
33 /*=====*/
34
35 static option_t options[] = {
36     {1, "i", "input", "-", opt_input, OPT_DEFAULT},
37     {1, "o", "output", "-", opt_output, OPT_DEFAULT},
38     {0, },
39 };
40
41
42 /*=====*/
43 //                                     MAIN
44 /*=====*/
45
46
47 int main(int argc, char * const argv[]){
48
49     std::cout<<"AlgoChain v2.1.1 - Grupo 5 : Galvan - Vera - Dreszman"<<std::endl;
50
51     //-----Valido Argumentos -----//
52     cmdline cmdl(options);
53     cmdl.parse(argc, argv);
54
55     //-----Le paso los archivos al Manager -----//
56
57     BlockchainManager::proccesBlockChain();
58
59     return 0;
60 }
61
62
63
64 /*=====*/
65 //                                     FUNCIONES INVOCADAS EN EL MAIN
66 /*=====*/
67
68 //----- Callbacks de CMDLINE -----//
69 static void opt_input(string const &arg)
70 {
71     // Si el nombre del archivos es "-", usaremos la entrada
72     // est?dar. De lo contrario, abrimos un archivo en modo
73     // de lectura.
74     //
75     if (arg == "-") BlockchainManager::setUserFilename(ios::in);
76     else BlockchainManager::setUserFilename(ios::in, arg.c_str(), false);
77 }
78
79 static void opt_output(string const &arg)
80 {
81
82     // Si el nombre del archivos es "-", usaremos la salida
83     // est?dar. De lo contrario, abrimos un archivo en modo

```

```

84 // de escritura.
85 //
86 //outputFileName = arg.c_str();
87 if (arg == "-") BlockChainManager::setUserFilename(ios::out);
88 else BlockChainManager::setUserFilename(ios::out, arg.c_str(), false);
89 }

```

## 6. Compilación

Se optó por utilizar la herramienta **make** en lugar de la compilación manual. Esta herramienta utiliza el archivo Makefile, automatizando así el proceso de compilación. De esta forma el proyecto se mantiene ordenado y se agiliza su desarrollo. Se ejecuta desde la terminal de LINUX simplemente insertando el comando **make**.

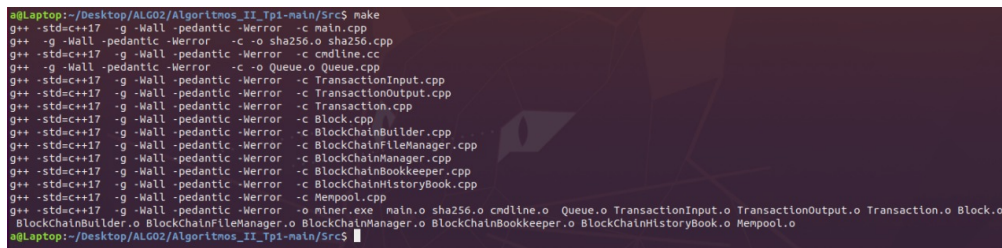


Figura 6.1: Compilación usando make

También el makefile viene integrado con la función *valgrind*. El sistema de *valgrind* realiza un chequeo de fugas de memoria por consiguiente es cómodo una vez realizado los cambios del programa ejecutar este test.

makefile.

```

1
2 OBJECTS = main.o sha256.o cmdline.o Queue.o TransactionInput.o TransactionOutput.o
3   Transaction.o Block.o BlockChainBuilder.o BlockChainFileManager.o BlockChainManager.o
4   BlockChainBookkeeper.o BlockChainHistoryBook.o Mempool.o # Los archivos compilados
5   individuales
6
7 PROGR = miner.exe # Nombre del archivo ejecutable
8 CPPFLAGS = -g -Wall -pedantic -Werror # -g opcion de g++ para debugear
9
10 # Compiladores #
11 CC = g++ -std=c++17 # Para linux
12 CCW = i686-w64-mingw32-g++ --static # Para windows (requiere mingw32)
13
14 $(PROGR) : $(OBJECTS)
15   $(CC) $(CPPFLAGS) -o $(PROGR) $(OBJECTS)
16 main.o : main.cpp cmdline.h BlockChainManager.h
17   $(CC) $(CPPFLAGS) -c main.cpp
18 cmdline.o : cmdline.cc cmdline.h
19   $(CC) $(CPPFLAGS) -c cmdline.cc
20 sha246.o : sha246.cpp sha246.h
21   $(CC) $(CPPFLAGS) -c sha246.cpp
22 queue.o : Queue.cpp Queue.h
23   $(CC) $(CPPFLAGS) -c Queue.cpp
24 Mempool.o : Mempool.cpp Mempool.h
25   $(CC) $(CPPFLAGS) -c Mempool.cpp
26 BlockChainManager.o : BlockChainManager.cpp BlockChainManager.h BlockChainFileManager.h
27   BlockChainBuilder.h BlockChainBookkeeper.h BlockChainStatus.h
28   $(CC) $(CPPFLAGS) -c BlockChainManager.cpp
29 BlockChainFileManager.o : BlockChainFileManager.cpp BlockChainFileManager.h BlockChainBuilder.
30   h BlockChainDataTypes.h BlockChainStatus.h Queue.h
31   $(CC) $(CPPFLAGS) -c BlockChainFileManager.cpp
32 BlockChainBuilder.o : BlockChainBuilder.cpp BlockChainBuilder.h lista.h sha256.h Block.h
33   TiposHash.h BlockChainDataTypes.h BlockChainStatus.h
34   $(CC) $(CPPFLAGS) -c BlockChainBuilder.cpp
35 BlockChainBookkeeper.o : BlockChainBookkeeper.cpp BlockChainBookkeeper.h BlockChainDataTypes.h
36   BlockChainStatus.h
37   $(CC) $(CPPFLAGS) -c BlockChainBookkeeper.cpp

```

```

30|BlockChainHistoryBook.o : BlockChainHistoryBook.cpp BlockChainHistoryBook.h Block.h lista.h
31|    $(CC) $(CPPFLAGS) -c BlockChainHistoryBook.cpp
32|Block.o : Block.cpp Block.h Transaction.h lista.h TiposHash.h BlockChainDataTypes.h
33|    $(CC) $(CPPFLAGS) -c Block.cpp
34|Transaction.o : Transaction.cpp Transaction.h lista.h TransactionOutput.h TransactionInput.h
35|    BlockChainDataTypes.h
36|    $(CC) $(CPPFLAGS) -c Transaction.cpp
37|TransactionOutput.o : TransactionOutput.cpp TransactionOutput.h
38|    $(CC) $(CPPFLAGS) -c TransactionOutput.cpp
39|TransactionInput.o : TransactionInput.cpp TransactionInput.h
40|    $(CC) $(CPPFLAGS) -c TransactionInput.cpp
41|
42|
43|
44|clean:
45|    rm -f core $(PROGR) $(OBJECTS)
46|all: $(PROGR)
47|    $(CC) $(CPPFLAGS) -o $(PROGR) $(OBJECTS)
48|memcheck : $(PROGR)
49|    valgrind --leak-check=yes ./$(PROGR) -i input.txt -o output.txt
50|memcheck_s : $(PROGR)
51|    valgrind -s --leak-check=yes ./$(PROGR) -i input.txt -o output.txt
52|std_out : $(PROGR)
53|    ./$(PROGR) -i input.txt -o -
54|std_in : $(PROGR)
55|    ./$(PROGR) -i - -o output.txt
56|std : $(PROGR)
57|    ./$(PROGR) -i - -o -

```







## 7.2. Pruebas de errores forzados

```
ramiro@ramiro-VirtualBox:~/Escritorio/Compartida/Tp1/Union/Algoritmos_II_Tp1/Src$ make std
./miner.exe -i - -o -
AlgoChain v2.1.1 - Grupo 5 : Galvan - Vera - Dreszman
La direccion del archivo Origen es : Cin (Entrada Standard)
La direccion del archivo Destino es: Cout (Salida Standard)
init ramiro 1000 5
Begin Parse Command ...Done
0157a02e55de6500c6d9079f0cd35558618a69413ededd434a5d96c6e3bddc4c
transfer alan 500 sergio 400
FAIL
Error Comando con argumentos invalidos
Begin Parse Command ...Fail: Error Not Defined
Finishing Execution
ramiro@ramiro-VirtualBox:~/Escritorio/Compartida/Tp1/Union/Algoritmos_II_Tp1/Src$ make std
./miner.exe -i - -o -
AlgoChain v2.1.1 - Grupo 5 : Galvan - Vera - Dreszman
La direccion del archivo Origen es : Cin (Entrada Standard)
La direccion del archivo Destino es: Cout (Salida Standard)
init ramiro 1000 5
Begin Parse Command ...Done
0157a02e55de6500c6d9079f0cd35558618a69413ededd434a5d96c6e3bddc4c
transfer ramiro alan 500 sergio 400
Begin Parse Command ...Done
d342a46ff48eb61790f14c54cf22c5d802ffc803a6d81891ce6e6f00b2dd5330
mine 10
Begin Parse Command ...Done
003a3af2602931fd0f16ef561a44a602cde8a945b2a70b3163256868f9bfff595
save prueba.txt
Begin Parse Command ...Done
Begin Converting Block to File ...
OK
```

**Figura 7.6:** Error forzado

[illegible]

**Figura 7.7:** Salida generada

```
ramiro@ramiro-VirtualBox:~/Escritorio/Compartida/Tp1/Union/Algoritmos_II_Tp1/Src$ make std
./miner.exe -i - -o -
AlgoChain v2.1.1 - Grupo 5 : Galvan - Vera - Dreszman
La direccion del archivo Origen es : Cin (Entrada Standard)
La direccion del archivo Destino es: Cout (Salida Standard)
init alan 200 5
Begin Parse Command ...Done
03a81fd22317bae2f191f6e9f776696ecb5638b6aceb2efe14303c852f9f393b
transfer alan sergio 50
Begin Parse Command ...Done
75ed4d5b51f75dd61f7fb515e215a5060221037baadb5c6281cc62af14916082
transfer sergio ramiro 25
Begin Parse Command ...Done
cb817ce851cb60c4aebcc63da302388151b620d214dff8b47becd6f0f0eb32d4
mine 20
Begin Parse Command ...Done
00000bc298b9f21f7ac517eb6f59a7794b27296a6889d6fbe6cf7aebc26b7c27
save prueba2.txt
Begin Parse Command ...Done
Begin Converting Block to File ...
OK
```

**Figura 7.8:** Prueba con doble transferencia en la Mempool y minado con 20 de dificultad



## **8. Conclusión**

Dada la complejidad requerida para la implementación y el tiempo estipulado para su realización resultó un desafío acordar un diseño y llevarlo a cabo a tiempo. Principalmente, la diferencia de criterios en cuanto a cómo estructurar el programa y las correcciones que se hicieron sobre el avance resultó lo mas desafiante.

En esta instancia del trabajo, se creó una base funcional que expresa en su ejecución el concepto de la tecnología Blockchain al lograr simular de forma mas extensiva que en el TP0 la experiencia de minado; utilizando para su implementación el paradigma de objetos.

La implementación de la clase BlockchainBookkeeper como clase "pulpo" resulta por un lado una decisión de diseño que agiliza la transferencia de información entre Mempool, BlockchainHistoryBook y los flujos de entrada y salida. En algunas situaciones, de esto mismo resultó cierta confusión sobre los alcances de cada clase. Más allá de esto último la conceptualización de esta clase en el diseño resultó positiva en el desarrollo del mismo.

En líneas mas generales, nos pareció propicio mencionar que con el correr del desarrollo surgió la percepción de que para un proyecto de estas características hay una gran variabilidad en lo que respecta a formas de modularización, según el criterio que se adopte.

Finalmente, y volviendo a lo particular, la forma elegida representa para nosotros la forma mas eficiente (en todo el sentido de la palabra) de realizarlo, logrando diferenciar las tareas y los roles de cada función propio de lo que paradigma orientado a objetos propone

## **9. Anexo I**

### **9.1. Enunciado**

# 75.04/95.12 Algoritmos y Programación II

## Trabajo práctico 1: algoritmos y estructuras de datos

Universidad de Buenos Aires - FIUBA  
Segundo cuatrimestre de 2020

### 1. Objetivos

Ejercitar conceptos relacionados con estructuras de datos, diseño y análisis de algoritmos. Escribir un programa en C++ (y su correspondiente documentación) que resuelva el problema que presentaremos más abajo.

### 2. Alcance

Este Trabajo Práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

### 3. Requisitos

El trabajo deberá ser entregado a través del campus virtual, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe de acuerdo con lo que mencionaremos en la Sección 5, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

### 4. Descripción

El propósito de este trabajo es continuar explorando los detalles técnicos de Bitcoin y blockchain, tomando como objeto de estudio nuestra versión simplificada de la blockchain introducida en el primer trabajo práctico: la ALGOCHAIN.

En esta oportunidad, extenderemos el alcance de nuestros desarrollos y operaremos con cadenas de bloques completas. Para ello, nos apoyaremos en un protocolo sencillo que permite abstraer los aspectos técnicos de la ALGOCHAIN. Al implementar este protocolo, nuestros programas podrán actuar como clientes transaccionales de la ALGOCHAIN, simplificando la operativa de cara al usuario final.

## 4.1. Tareas a realizar

A continuación enumeramos las tareas que deberemos llevar a cabo. Cada una de estas será debidamente detallada más adelante:

1. Implementación de una interfaz operativa basada en un protocolo artificial para interactuar con la ALGOCHAIN.
2. Lectura, interpretación y pre-procesamiento de una ALGOCHAIN completa.
3. Nuevo algoritmo de cómputo del campo `txns_hash` basado en árboles de Merkle.

### 4.1.1. Protocolo operacional

El protocolo con el que trabajaremos consiste en una serie de **comandos** que permiten representar distintas operaciones sobre la ALGOCHAIN. Cada comando recibe una cantidad específica de parámetros, realiza cierta acción y devuelve un resultado al usuario final.

**Conceptos preliminares** Antes de detallar los comandos, es importante definir algunos conceptos preliminares:

- **Bloque génesis:** al igual que en blockchain, al primer bloque de toda ALGOCHAIN se lo conoce como *bloque génesis*. Esencialmente, este bloque introduce un saldo inicial para un usuario dado. Puesto que no existen bloques anteriores, el campo `prev_block` de un bloque génesis debe indicar un hash nulo (i.e., con todos los bytes en 0). Este bloque debe también contar con un único *input* y un único *output*. De igual modo, el *input* debe referenciar un *outpoint* nulo, mientras que el *output* hace la asignación del saldo inicial respetando el formato usual.
- **Mempool:** el protocolo de este trabajo permitirá que nuestros programas operen como mineros de la ALGOCHAIN. Emulando el comportamiento de los mineros de Bitcoin, nuestros programas contarán con un espacio en memoria donde se alojarán las transacciones de los usuarios que aún no fueron confirmadas (i.e., que no se agregaron a la ALGOCHAIN). Este espacio se conoce como *mempool*.

### Descripción de los comandos

- `init <user> <value> <bits>`

**Descripción.** 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.

**Valor de retorno.** El hash del bloque génesis. Observar que es posible realizar múltiples invocaciones a `init` (en tales casos, el programa debe descartar la información de la ALGOCHAIN anterior).

- `transfer <src> <dst1> <value1> ... <dstN> <valueN>`



**Descripción.** Genera una nueva transacción en la que el usuario *src* transferirá fondos a una cantidad *N* de usuarios (al *i*-ésimo usuario, *dst<sub>i</sub>*, se le transferirá un monto de *value<sub>i</sub>*). Si el usuario origen no cuenta con la cantidad de fondos disponibles solicitada, la transacción debe considerarse inválida y no llevarse a cabo.

**Consideraciones adicionales.** Recordar que cada *input* de una transacción toma y utiliza la cantidad completa de fondos del *outpoint* correspondiente. En caso de que una de nuestras transacciones no utilice en sus *outputs* el saldo completo recibido en los *inputs*, la implementación de este comando debe generar un *output* adicional con el *vuelto* de la operación. Este vuelto debe asignarse a la dirección del usuario origen.

**Valor de retorno.** Hash de la transacción en caso de éxito; FAIL en caso de falla por invalidez.

■ `mine <bits>`

**Descripción.** Ensambla y agrega a la ALGOCHAIN un nuevo bloque a partir de todas las transacciones en la *mempool*. La dificultad del minado viene dada por el parámetro *bits*.

**Valor de retorno.** Hash del bloque en caso de éxito; FAIL en caso de falla por invalidez.

■ `balance <user>`

**Descripción.** Consulta el saldo disponible en la dirección del usuario *user*. Las transacciones en la *mempool* deben contemplarse para responder esta consulta.

**Valor de retorno.** Saldo disponible del usuario.

■ `block <id>`

**Descripción.** Consulta la información del bloque representado por el hash *id*.

**Valor de retorno.** Los campos del bloque siguiendo el formato usual. Debe devolver FAIL en caso de recibir un hash inválido.

■ `txn <id>`

**Descripción.** Consulta la información de la transacción representada por el hash *id*.

**Valor de retorno.** Los campos de la transacción siguiendo el formato usual. Debe devolver FAIL en caso de recibir un hash inválido.

■ `load <filename>`

**Descripción.** Lee la ALGOCHAIN serializada en el archivo pasado por parámetro.

**Valor de retorno.** Hash del último bloque de la cadena en caso de éxito; FAIL en caso de falla por invalidez de algún bloque y/o transacción. Observar que es posible realizar múltiples invocaciones a *load* (en tales casos, el programa debe descartar la información de la ALGOCHAIN anterior).

■ `save <filename>`

**Descripción.** Guarda una copia de la ALGOCHAIN en su estado actual al archivo indicado por el parámetro `filename`. Cada bloque debe serializarse siguiendo el formato usual. Los bloques deben aparecer en orden en el archivo, comenzando desde el génesis.

**Valor de retorno.** OK en caso de éxito; FAIL en caso de falla.

#### 4.1.2. Lectura de la Algochain

Tal como se infiere del comando `load`, nuestros programas deberán tener la capacidad de leer e interpretar versiones completas de la ALGOCHAIN. Esto permitirá extender e interactuar con cadenas de bloques arbitrarias, permitiendo entre otras cosas el cruce de información entre grupos distintos.

En resumen, los programas deberán poder recibir una ALGOCHAIN serializada en un archivo de entrada y leer la información bloque a bloque, posiblemente organizando los datos en estructuras convenientes para facilitar las consultas y operaciones posteriores. El formato de entrada sigue los lineamientos detallados en el enunciado del trabajo práctico anterior: una ALGOCHAIN no es otra cosa que una concatenación ordenada de bloques.

#### 4.1.3. Árboles de Merkle y hash de transacciones

Un *árbol de Merkle* [3] es un árbol binario completo en el que los nodos almacenan hashes criptográficos. Dada una secuencia de datos  $L_1, \dots, L_n$  sobre la que se desea obtener un hash, el árbol de Merkle se define computando primero los hashes  $h(L_1), \dots, h(L_n)$  y generando hojas a partir de estos valores. Cada par de hojas consecutivas es a su vez hashado concatenando los respectivos hashes, lo cual origina un nuevo nodo interno del árbol. Este proceso se repite sucesivamente nivel tras nivel, llegando eventualmente a un único hash que corresponde a la raíz del árbol. Esto se ilustra en la Figura 1.

Una particularidad interesante de un hash basado en árboles de Merkle es que resulta muy eficiente comprobar que un dato dado forma parte del conjunto de datos representado por la raíz del árbol. Esta comprobación requiere computar un número de hashes proporcional al logaritmo del número de datos iniciales (cf. el costo lineal en esquemas secuenciales como el adoptado en el primer trabajo práctico).

Siguiendo los lineamientos del protocolo de Bitcoin, en este trabajo práctico computaremos los hashes de las transacciones de un bloque a partir de un árbol de Merkle. En otras palabras, el campo `txns_hash` del header de un bloque  $b$  arbitrario deberá contener el hash SHA256 correspondiente a la raíz del árbol del Merkle construido a partir de la secuencia de transacciones de  $b$ .

En caso de que la cantidad de transacciones no pueda agruparse de a pares, la última transacción debe agruparse consigo misma para generar los hashes del nivel superior del árbol. Esta estrategia debe repetirse en cada nivel sucesivo.

**Ejemplo de cómputo** Supongamos que queremos calcular el árbol de Merkle para una secuencia de tres cadenas de caracteres:  $s_1 = \text{árbol}$ ,  $s_2 = \text{de}$  y  $s_3 = \text{Merkle}$ . El cómputo debería seguir los siguientes pasos:



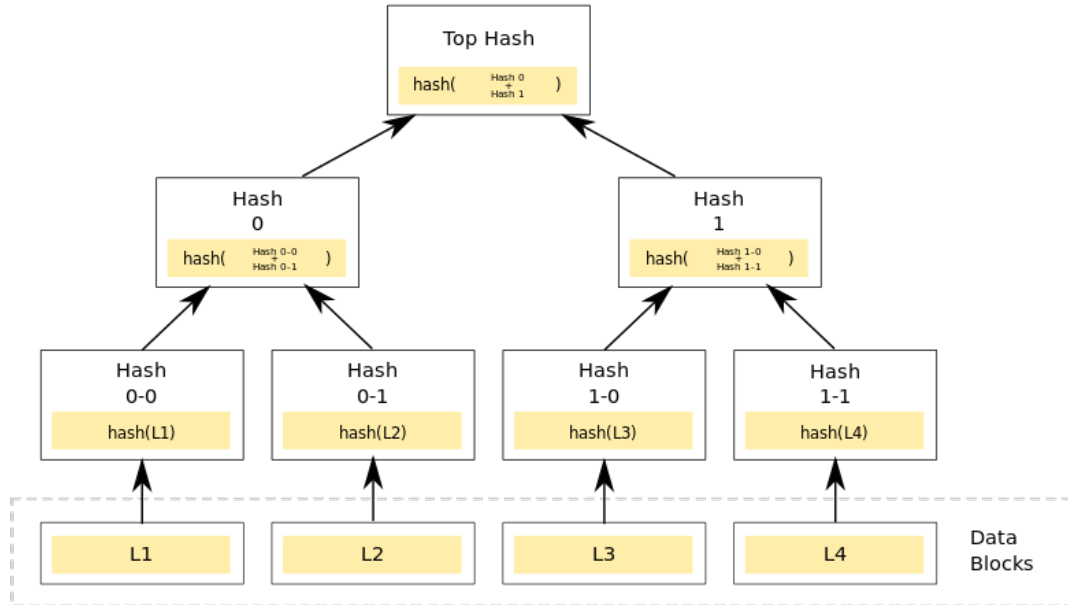


Figura 1: Esquema de un árbol de Merkle (cortesía Wikipedia). El operador +, en este contexto, indica concatenación de hashes y no suma numérica.

1. Para cada  $s_i$ , se calcula  $h_i$ , un doble hash SHA256 de dicha cadena.
2. Se agrupa  $h_1$  con  $h_2$  y  $h_3$  consigo mismo. Esto da lugar a un nuevo nivel en el árbol formado por dos nuevas cadenas  $s_{1,2} = h_1 + h_2$  y  $s_{3,3} = h_3 + h_3$  con las respectivas concatenaciones de los hashes del nivel inferior.
3. Se vuelve a repetir el proceso anterior, en esta oportunidad a partir de los hashes  $h_{1,2}$  y  $h_{3,3}$  de  $s_{1,2}$  y  $s_{3,3}$ , respectivamente.
4. De lo anterior surge un nuevo nivel del árbol con un único nodo,  $H$ . Este nodo es la raíz del árbol de Merkle para  $s_1, s_2$  y  $s_3$ .

Los hashes anteriores son los siguientes:

- $h_1 = \text{a225a1d1a31ea0d7eca83bcfe582f915539f926526634a4a8e234a072b2cec23}$
- $h_2 = \text{b2d04d58d202b5a4a7b74bc06dc86d663127518cfe9888ca0bb0e1a5d51e6f19}$
- $h_3 = \text{b96c4732b691beb72b3a8f28c59897bd58f618dbac1c3b0119bcea85ada0212f}$
- $h_{1,2} = \text{798f857ba2cdd63f03e22aa5aa52340f10da8fc8b5183dfe989ad366327d36fc}$
- $h_{3,3} = \text{af2b866e8ef21130a6ca55776f256a002215e72e99a711978534772af767fbf8}$
- $H = \text{abe24c1aeaf6f7358e1702009026c8ad146aa5321e91d36e1928bfc8e6e48896}$

#### 4.1.4. Consideraciones adicionales

- Los detalles técnicos de la blockchain y el formato de transacciones y bloques de la ALGOCHAIN fueron deliberadamente omitidos en este enunciado. Sugerimos remitirse al enunciado del primer trabajo práctico para revisar preventivamente todos estos conceptos.
- El cálculo de hashes SHA256 puede realizarse mediante la misma librería provista por la cátedra en la instancia anterior.
- Es importante remarcar que toda estructura de datos (e.g., listas, arreglos dinámicos, pilas o árboles) **debe ser implementada**. La única excepción permitida son las tablas de hash. En caso de necesitar utilizarlas, sugerimos revisar la clase `std::unordered_map` de la STL de C++ [4].

#### 4.2. Interfaz de línea de comandos

Al igual que en el primer trabajo práctico, la interacción con nuestros programas se dará a través de la línea de comandos. Las opciones a implementar en este caso son las siguientes:

- `-i`, o `--input`, que permite controlar el stream de entrada de los comandos del protocolo detallado en la Sección 4.1.1. Si este argumento es `"-"`, el programa deberá recibir los comandos por la entrada standard, `std::cin`. En otro caso, el argumento indicará el archivo de entrada conteniendo dichos comandos. Puede asumirse que cada comando aparece en una única línea dedicada.
- `-o`, o `--output`, que permite direccionar las respuestas del procesamiento de los comandos a un stream de salida. Si este argumento es `"-"`, el programa deberá mostrar las respuestas de los comandos por la salida standard, `std::cout`. En otro caso, el argumento indicará el archivo de salida donde deberán guardarse estas respuestas.

#### 4.3. Ejemplos

En lo que sigue mostraremos algunos ejemplos que ilustran el comportamiento básico del programa ante algunas entradas simples. Tener en cuenta las siguientes consideraciones:

- Los hashes mostrados podrían no coincidir con los computados por otras implementaciones, puesto que dependen entre otras cosas de la elección de los nonces al momento de minar los bloques.
- Al igual que en los ejemplos del trabajo práctico anterior, por conveniencia resumiremos algunos hashes con sus últimos 8 bytes. Las entradas y salidas de nuestros programas deben, naturalmente, trabajar con los hashes completos.

##### 4.3.1. Ejemplo trivial: entrada vacía

Si no hay comandos para procesar (i.e., el stream de entrada es vacío), el programa no debe realizar ninguna acción:

```
$ ./tp1 -i /dev/null -o output.txt
$ cat output.txt
$
```

#### 4.3.2. Múltiples inits

En los ejemplos subsiguientes, por claridad resaltaremos los comandos de entrada en color azul y con un símbolo > al comienzo.

En la invocación que se muestra más abajo, inicializamos primero una nueva cadena en la que el usuario satoshi dispone de 100 unidades de dinero. El correspondiente bloque génesis es minado con una dificultad de 10 bits. Luego de esto, reseteamos la cadena asignándole al usuario lucas una unidad de dinero:

```
$ ./tp1
> init satoshi 100 10
b983fdeb9cbe9426cc1df0ef057b44e583e5ea7531c90376eba518ecfb08a246
> balance satoshi
100.0
> balance lucas
0
> init lucas 1 10
b40495bf172be3c172a41a85f72d13e8b2e8e7e582fc7e14b05e614408b52667
> balance satoshi
0
> balance lucas
1.0
> block fb08a246
FAIL
> block 08b52667
0000000000000000000000000000000000000000000000000000000000000000
647bbe505403dca7a11d08269d02017c72eb0fc2e4398befe41cea620570e639
10
2535
1
1
00000000 0 00000000
1
1.0 f82e82dac113d37a21e2b3e0c37eab9e6fbc3657a38b0a8397d913abedab7605
$
```

Se observa lo siguiente:

- Luego del segundo init, los balances de los usuarios cambian. Puesto que este comando genera nuevas instancias de la ALGOCHAIN, es razonable que esto suceda.
- Al pedir el bloque con hash fb08a246, vemos que el programa informa una falla. Esta falla proviene de un hash de bloque inválido en la cadena actual: notar que dicho hash corresponde al bloque génesis de la primera cadena.

- El último comando solicita la información del bloque cuyo hash es 08b52667. En este caso, dicho hash coincide con el del nuevo bloque génesis, por lo que la operación es ahora exitosa.

Podemos también realizar una operatoria en modo *batch* copiando todos estos comandos en un archivo e invocando luego al programa con este archivo como entrada:

```
$ cat commands.txt
init satoshi 100 10
balance satoshi
balance lucas
init lucas 1 10
balance satoshi
balance lucas
block fb08a246
block 08b52667
$ ./tp1 -i commands.txt -o output.txt
$ cat output.txt
b983fdeb9cbe9426cc1df0ef057b44e583e5ea7531c90376eba518ecfb08a246
100.0
0
b40495bf172be3c172a41a85f72d13e8b2e8e7e582fc7e14b05e614408b52667
0
1.0
FAIL
0000000000000000000000000000000000000000000000000000000000000000
647bbe505403dca7a11d08269d02017c72eb0fc2e4398befe41cea620570e639
10
2535
1
1
00000000 0 00000000
1
1.0 f82e82dac113d37a21e2b3e0c37eab9e6fbc3657a38b0a8397d913abedab7605
$
```

Prestar especial atención a la última línea de la salida: como todo *output*, debe finalizar con un caracter de salto de línea (sugerimos remitirse al formato de transacciones y bloques detallado en el enunciado del primer trabajo práctico en caso de dudas).

#### 4.3.3. Transferencias

El próximo ejemplo utiliza el comando `transfer` para generar transacciones y mover dinero entre distintos usuarios:

```
$ ./tp1
> init satoshi 100 10
```

```

b983fdeb9cbe9426cc1df0ef057b44e583e5ea7531c90376eba518ecfb08a246
> transfer satoshi lucio 90
4ab0d8a4fdab846e9f28c1850fe06a73b446341ba7eab2cab8eae9948597e1e1
> transfer satoshi lucas 1
0a7e61b9b17c7e7e21aef8d5e65e3b036e949c7f398bd0692b5b704cf04e9b84
> balance lucio
90.0
> mine 10
5d4075e53f5cb51da5fffb3e68eef18046fc8c1327c4c4f787550b2e94e013806
> balance satoshi
9.0
> balance lucio
90.0
> balance lucas
1.0
> transaction f04e9b84
1
8597e1e1 1 ea55eb5c
2
1.0 f82e82dac113d37a21e2b3e0c37eab9e6fbc3657a38b0a8397d913abedab7605
9.0 5fe3f3a6faaef93165aff8d88e701f965b8b956ea77e3116c8c8b2cfea55eb5c

$

```

Es importante destacar lo siguiente:

- La primera invocación de transfer consume el UTXO del usuario satoshi en el bloque génesis. La transacción generada deposita 90 unidades de dinero en la dirección del usuario lucio y un vuelto de 10 unidades de dinero en la dirección de satoshi. El hash de esta transacción es 8597e1e1.
- La segunda invocación de transfer debe, necesariamente, consumir el UTXO de satoshi correspondiente a la transacción anterior (observar que el primer *output* en el bloque génesis ya fue consumido y no puede volver a utilizarse). Esta vez, se generará una nueva transacción que deposita una unidad de dinero en la dirección de lucas y un vuelto de 9 unidades de dinero en la dirección de satoshi.
- La primera invocación de balance nos dice que lucio tiene 90 unidades de dinero disponibles. Este dinero está sujeto a ser confirmado puesto que la transacción todavía se encuentra en la *mempool*.
- Luego de minar el nuevo bloque a partir de las transacciones anteriores, el saldo de lucio aparece confirmado con la misma cantidad de dinero. Por otro lado, satoshi tiene un saldo de 9 unidades de dinero, mientras que lucas sólo dispone de una unidad de dinero.
- Por último, el comando transaction solicita información sobre la transacción con hash f04e9b84. Vemos que este hash corresponde a la transacción derivada del segundo uso de transfer. Allí puede verse el vuelto de 9 unidades de dinero a la dirección de satoshi (el segundo *output* de dicha transacción).

#### 4.3.4. Lectura y escritura de cadenas

Finalmente, veamos cómo leer y escribir cadenas completas con nuestros programas. El siguiente ejemplo guarda una cadena de dos bloques al archivo `algochain.txt`:

```
$ ./tp1
> init satoshi 100 10
b983fdeb9cbe9426cc1df0ef057b44e583e5ea7531c90376eba518ecfb08a246
> transfer satoshi lucas 1 lucio 90
8b58f15e5c4408b30322daca6d14edd44ff3d067d8b1ea967dff89d5705f5ff3
> mine 10
3b44b8c5182097fa63c2e84aa27735f8cad40971c84266fb874d0bd993c15315
> save algochain.txt
OK
$
```

Notar que, esta vez, el comando `transfer` incluye múltiples destinatarios: la transacción depositará una unidad de dinero en la dirección de `lucas` y 90 unidades de dinero en la de `lucio` (esto se lleva a cabo definiendo dos *outputs* diferentes). Puesto que el saldo de `satoshi` consumido por la transacción es de 100 unidades de dinero, el vuelto que le corresponde es de 9 unidades.

En esta invocación posterior, cargamos la cadena anterior a partir del archivo generado. Observar que la información de la cadena inicial (la generada vía `init`) se descarta:

```
$ ./tp1
> init satoshi 100 10
b983fdeb9cbe9426cc1df0ef057b44e583e5ea7531c90376eba518ecfb08a246
> load algochain.txt
3b44b8c5182097fa63c2e84aa27735f8cad40971c84266fb874d0bd993c15315
> balance satoshi
9.0
> balance lucio
90.0
> balance lucas
1.0
$
```

#### 4.4. Portabilidad

Es deseable que la implementación desarrollada provea un grado mínimo de portabilidad. Sugerimos verificar nuestros programas en alguna versión reciente de UNIX: BSD o Linux.

### 5. Informe

El informe deberá incluir, como mínimo:

- Una carátula que incluya los nombres de los integrantes y el listado de todas las entregas realizadas hasta ese momento, con sus respectivas fechas.
- Documentación relevante al diseño e implementación del programa.
- Documentación relevante a los algoritmos y estructuras de datos involucrados en la solución del trabajo.
- El análisis de las complejidades solicitado en la sección 4.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C++.
- Este enunciado.

## 6. Fechas

La última fecha de entrega es el **jueves 3 de diciembre de 2020**.

## Referencias

- [1] Wikipedia, "Bitcoin Wiki." [https://en.bitcoin.it/wiki/Main\\_Page](https://en.bitcoin.it/wiki/Main_Page).
- [2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009.
- [3] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the theory and application of cryptographic techniques*, pp. 369–378, Springer, 1987.
- [4] cplusplus.com, "Unordered Map." [https://www.cplusplus.com/reference/unordered\\_map/unordered\\_map/](https://www.cplusplus.com/reference/unordered_map/unordered_map/).