



ALGORITMOS Y PROGRAMACIÓN II (75.04/95.12)

---

## TRABAJO PRÁCTICO N° 0

---

### 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 12/11/2020

---

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Diseño (decisiones fundacionales)</b>	<b>2</b>
<b>3. Implementación</b>	<b>2</b>
<b>4. Clases</b>	<b>3</b>
4.1. Clase BlockchainManager . . . . .	3
4.2. Clase BlockchainFileManager . . . . .	6
4.3. Clase BlockchainBuilder . . . . .	10
4.4. Clase Block . . . . .	13
4.5. Clase Transaction . . . . .	17
4.6. Clase TransactionInput . . . . .	20
4.7. Clase TransactionOutput . . . . .	22
4.8. Clases sha256 cmdlime . . . . .	23
<b>5. Compilación</b>	<b>23</b>
<b>6. Pruebas sobre el programa</b>	<b>25</b>
6.1. Sin parámetros . . . . .	25
6.2. Archivo de entrada en otro formato (no .txt) . . . . .	25
6.3. Parámetro No-Válido . . . . .	25
6.4. Input inexistente . . . . .	25
6.5. input.txt vacío . . . . .	26
6.6. No existe archivo en input con ese nombre . . . . .	26
6.7. Imágenes de Prueba y Transformaciones . . . . .	26
<b>7. Conclusión</b>	<b>27</b>
<b>8. Anexo I</b>	<b>27</b>
8.1. Enunciado . . . . .	27

## 1. Introducción

El siguiente trabajo práctico 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 debe recibir un archivo en formato texto, con extensión **.txt** (*input.txt*). En el mismo se encuentra la información necesaria para construir un *bloque* (unidad básica de la Algochain). A partir de dicha información el proceso subsiguiente consiste en parsear adecuadamente dicha información, y efectuar los procesos correspondientes (principalmente el *hashing* característico de ésta tecnología) para obtener el bloque buscado, como *output* del programa. A continuación se detallan las especificidades de la implementación correspondiente a lo expuesto arriba.

## 2. Diseño (decisiones fundacionales)

Dentro del paradigma OOP, se ha optado por un acercamiento que se propone una modularización tanto en la columna vertebral de la funcionalidad del programa como de los procesos adyacentes o de soporte. Particularmente se observará (más en detalle en la sección correspondiente) que la lógica central del programa se encuentra en la clase llamada **BlockchainBuilder**, mientras que las demás clases realizan tareas de manejo de archivos, validación y parseo, entre otras. Cabe destacar que las acciones correspondientes al manejo de errores en caso de que la información contenida en el archivo de entrada no sea válida también son llevadas a cabo por una clase particular.

## 3. Implementación

La implementación es prácticamente secuencial. El flujo del programa sigue la siguiente lógica:

1. Validación de argumentos
2. Apertura de *input.txt* y selección de dificultad del hash
3. Validación del formato de la información en *input.txt*
4. Parseo de la información
5. Cálculo iterativo de la función de hash
6. Ensamblaje del bloque
7. Generación del archivo de salida

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

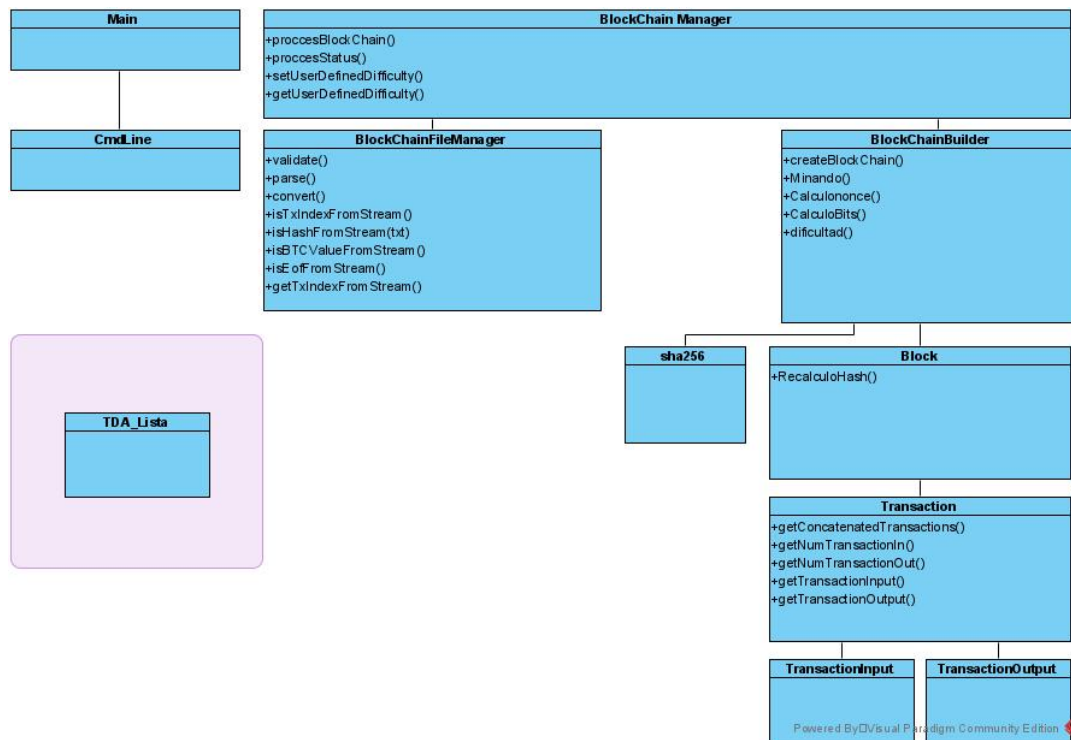


Figura 3.1: Diagrama de Clases. Obs: Se obviaron metodos constructores, destructores, *getters* y *setters*

## 4. Clases

### 4.1. Clase BlockchainManager

La clase **BlockchainManager** parte de la idea de gestionar el proyecto. Si se realiza una analogía con una oficina esta clase seria el jefe de la empresa. La idea fue tratar de conceptualizar y generar un TDA Manager. Podrían inferirse algunas funciones elementales :

1. Gestionar el flujo del programa a través de estados : Cada proceso del 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, 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 implementar esta clase de manera estática. De esta manera no existe un objeto *Manager* sino mas bien opera de entidad. Esto facilitó integrar la clase *cmdline* a través del método **setUserDefinedDifficulty()**, pues logra cargar el atributo estático *UserDefinedDifficulty* sin necesidad de instanciar la clase. En cuanto a el manejo de errores, se decidió utilizar la función *std::abort()* puesto que esta no invoca destructores antes de cortar el flujo, lo que nos obligaba a hacer un manejo de memoria dinámica correcto antes de cortar el flujo.

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

```

1 //----- BlockchainManager.cpp -----//
2
3 #include "BlockchainManager.h"
4
5
6 void BlockchainManager::proccesBlockChain(std::istream *iss, std::ostream *oss) {
7     BlockchainBuilder builder(BlockChainManager::getUserDefinedDifficulty());
8     BlockchainFileManager fileManager;
9
10    std::cout<< "Begin Validate ...";
11    BlockchainManager::proccesStatus( fileManager.validate(iss) );
12
13    std::cout<< "Begin Parsing ..." ;
14    BlockchainManager::proccesStatus( fileManager.parse(iss, builder.getRawPointer()) );
15
16    std::cout<< "Begin Creating Block ..." ;
17    BlockchainManager::proccesStatus( builder.createBlockChain() );
18
19
20    std::cout<< "Begin Converting Block to File ..." << std::endl;
21    BlockchainManager::proccesStatus( fileManager.convert(oss, builder.getBlockChainPointer())
22    );
23
24    std::cout<< std::endl;
25    std::cout<< "Finish mining with hash :" << builder.getObtainedHash() << std::endl;
26 }
27
28 void BlockchainManager::proccesStatus(status_t status) {
29     switch(status) {
30
31     case STATUS_OK:
32         std::cout << "Done" << std::endl;
33         break;
34     case STATUS_FINISH_CONVERT_SUCCESSFULLY:
35         break;
36     case STATUS_CORRUPT_FORMAT:
37         std::cout << "Error de Formato: Formato Incorrecto" << std::endl;
38         std::cerr << "Error de Formato: Formato Incorrecto" << std::endl;
39         std::abort();
40         break;
41     case STATUS_CORRUPT_FORMAT_BAD_HASH:
42         std::cout << "Error de Formato: Hash incorrecto" << std::endl;
43         std::cerr << "Error de Formato: Hash incorrecto" << std::endl;
44         std::abort();
45         break;
46     case STATUS_CORRUPT_FORMAT_BAD_TXINDEX:
47         std::cout << "Error de Formato: Indice de Tx incorrecto" << std::endl;
48         std::cerr << "Error de Formato: Indice de Tx incorrecto" << std::endl;
49         std::abort();
50         break;
51     case STATUS_CORRUPT_FORMAT_BAD_TXIN:
52         std::cout << "Error de Formato: Indice Tx In Incorrecto" << std::endl;
53         std::cerr << "Error de Formato: Indice Tx In Incorrecto" << std::endl;
54         std::abort();
55         break;
56     case STATUS_CORRUPT_FORMAT_BAD_TXOUT:
57         std::cout << "Error de Formato: Indice Tx Out Incorrecto" << std::endl;
58         std::cerr << "Error de Formato: Indice Tx Out Incorrecto" << std::endl;
59         std::abort();
60         break;
61     case STATUS_CORRUPT_FORMAT_BAD_BTCVALUE:
62         std::cout << "Error de Formato: Valor de Bitcoin Incorrecto" << std::endl;

```

```

62     std::cerr << "Error de Formato: Valor de Bitcoin Incorrecto" << std::endl;
63     std::abort();
64     break;
65 case STATUS_BAD_ALLOC:
66     std::cout << "Error de sistema: Memoria insuficiente" << std::endl;
67     std::cerr << "Error de sistema: Memoria insuficiente" << std::endl;
68     std::abort();
69     break;
70 case STATUS_BAD_READ_INPUT_FILE:
71     std::cout << "Error de Lectura: Archivo de entrada danado" << std::endl;
72     std::cerr << "Error de Lectura: Archivo de entrada danado" << std::endl;
73     std::abort();
74     break;
75 case STATUS_BAD_READ_OUTPUT_FILE:
76     std::cout << "Error de Lectura: Archivo de salida danado" << std::endl;
77     std::cerr << "Error de Lectura: Archivo de salida danado" << std::endl;
78     std::abort();
79     break;
80 case STATUS_NO_BLOCKS_TO_CONVERT:
81     std::cout << "Error de Conversion: No hay nada que convertir" << std::endl;
82     std::cerr << "Error de Conversion: No hay nada que convertir" << std::endl;
83     std::abort();
84     break;
85 default:
86     std::cout << std::endl;
87     break;
88 }
89 }
90
91
92 #define DIFFICULTY_DEFAULT_VALUE 3
93 size_t BlockChainManager::userDefinedDifficulty = DIFFICULTY_DEFAULT_VALUE;
94
95 void BlockChainManager::setUserDefinedDifficulty(int d){
96     if( d < 0 ){
97         std::cout << "Error de Formato: Dificultad debe ser mayor a cero " << std::endl;
98         std::cerr << "Error de Formato: Dificultad debe ser mayor a cero" << std::endl;
99         std::abort();
100     }
101     userDefinedDifficulty = (size_t) d;
102 }
103
104
105 size_t BlockChainManager::getUserDefinedDifficulty( void ){
106     return userDefinedDifficulty;
107 }

```

```

1 //----- BlockChainManager.h -----//
2
3
4 #ifndef BLOCKCHAINMANAGER_H_
5 #define BLOCKCHAINMANAGER_H_
6
7 #include<string>
8 #include <iostream>
9 #include "BlockChainStatus.h"
10
11
12 class BlockChainManager {
13     status_t state;
14     static size_t userDefinedDifficulty;
15 public:
16
17     static void procesBlockChain( std::istream *iss, std::ostream *oss );
18     static void procesStatus( status_t status );
19
20     static void setUserDefinedDifficulty( int d );
21     static size_t getUserDefinedDifficulty( void );

```

```

22 };
23 };
24
25 #endif /* BLOCKCHAINMANAGER_H_ */

```

## 4.2. 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. El motivo de existencia de una clase interfaz archivo-programa es la flexibilidad que brinda si se decidiera utilizar otro formato de archivo (i.e. *.csv* o *.inf*) en el futuro, es decir, toda su lógica es reutilizable. La forma de trabajo de *BlockchainFileManager* es la siguiente.

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
2. *validate()*
3. 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 sera 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()*
4. Conversión : Una vez terminada la lógica principal la clase recibirá una lista de bloques (en este caso, una lista de 1 bloque) y comenzara 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
2
3 #include "BlockchainFileManager.h"
4
5
6 BlockchainFileManager::BlockchainFileManager() {
7     pRawData = NULL;
8 }
9
10 BlockchainFileManager::~BlockchainFileManager() {
11     if(this->pRawData != NULL){
12         pRawData->inTx = 0;
13         delete [] pRawData->IN_tableOfTxId;           pRawData->IN_tableOfTxId    = NULL;
14         delete [] pRawData->IN_tableOfIndex;           pRawData->IN_tableOfIndex  = NULL;
15         delete [] pRawData->IN_tableOfAddr;            pRawData->IN_tableOfAddr   = NULL;
16         pRawData->outTx = 0;
17         delete [] pRawData->OUT_tableOfValues;         pRawData->OUT_tableOfValues = NULL;
18         delete [] pRawData->OUT_tableOfAddr;           pRawData->OUT_tableOfAddr  = NULL;
19         delete pRawData;
20         pRawData = NULL;
21     }
22 }

```

```

23 | status_t BlockchainFileManager::validate(std::istream * iss){
24 |     int inTxTotal,outTxTotal;
25 |
26 |     if( this->isTxIndexFromStream(iss,'\n',&inTxTotal) == false )    return
STATUS_CORRUPT_FORMAT_BAD_TXIN;
27 |     for(int inTx = 0 ; inTx < inTxTotal ; inTx++){
28 |         if( this->isHashFromStream(iss,' ') == false )                return
STATUS_CORRUPT_FORMAT_BAD_HASH;
29 |         if( this->isTxIndexFromStream(iss,' ') == false )            return
STATUS_CORRUPT_FORMAT_BAD_TXINDEX;
30 |         if( this->isHashFromStream(iss) == false )                    return
STATUS_CORRUPT_FORMAT_BAD_HASH;
31 |     }
32 |     if( this->isTxIndexFromStream(iss,'\n',&outTxTotal) == false )    return
STATUS_CORRUPT_FORMAT_BAD_TXOUT;
33 |     for(int outTx = 0 ; outTx < outTxTotal ; outTx++){
34 |         if( this->isBTCValueFromStream(iss,' ') == false )            return
STATUS_CORRUPT_FORMAT_BAD_BTCVALUE;
35 |         if( this->isHashFromStream(iss) == false )                    return
STATUS_CORRUPT_FORMAT_BAD_HASH;
36 |     }
37 |     if( this->isEofFromStream(iss) == false )                          return
STATUS_CORRUPT_FORMAT;
38 |
39 |     return STATUS_OK;
40 | }
41 |
42 | bool BlockchainFileManager::isTxIndexFromStream(std::istream *iss,char delim , int * pValue)
43 | {
44 |     int IndexValue;
45 |     std::string line;
46 |     std::stringstream ss;
47 |
48 |     std::getline(*iss, line,delim);
49 |     ss.str(line);
50 |     if ((ss >> IndexValue).fail())    return false;
51 |     if (IndexValue < 0)                return false;
52 |     //Debug
53 |     //std::cout << line << std::endl;
54 |     if(pValue != NULL) *pValue = IndexValue;
55 |     return true;
56 | }
57 |
58 | bool BlockchainFileManager::isHashFromStream(std::istream *iss,char delim, std::string *
pString)
59 | {
60 |     std::string line;
61 |     std::stringstream ss;
62 |     std::getline(*iss, line,delim);
63 |     if( line.back() != '\r'){
64 |         if ( line.size() != 64 )    return false;}
65 |     else{
66 |         if ( line.size() != 64 + 1 ) return false;}
67 |     //Debug
68 |     //std::cout << line << std::endl;
69 |     if(pString != NULL) *pString = line;
70 |     return true;
71 | }
72 |
73 |
74 | bool BlockchainFileManager::isBTCValueFromStream(std::istream *iss,char delim,float * pFloat)
75 | {
76 |     float floatValue;
77 |     std::string line;
78 |     std::stringstream ss;
79 |
80 |     std::getline(*iss, line,delim);
81 |     ss.str(line);
82 |     if ((ss >> floatValue).fail())    return false;
83 |     if (floatValue < 0)                return false;

```



```

84     //Debug
85     //std::cout << line << std::endl;
86     if(pFloat != NULL) *pFloat = floatValue;
87     return true;
88 }
89
90 bool BlockchainFileManager::isEofFromStream(std::istream *iss){
91     std::string line;
92     return (std::getline(*iss, line))? false : true;
93 }
94
95
96 status_t BlockchainFileManager::parse(std::istream * iss, raw_t * &pBuilderRawData){
97     //Vuelvo al principio del File para hacer la carga
98     iss->clear();
99     iss->seekg(0, iss->beg);
100
101     //Creo el archivo raw_t en el entorno del filemanager
102     this->pRawData = new raw_t{0};
103     if(pRawData == NULL) return STATUS_BAD_ALLOC;
104     pRawData->inTx = this->getTxIndexFromStream(iss, '\n');
105
106     pRawData->IN_tableOfTxId = new std::string[pRawData->inTx];
107     pRawData->IN_tableOfIndex = new int[pRawData->inTx];
108     pRawData->IN_tableOfAddr = new std::string[pRawData->inTx];
109     if(      pRawData->IN_tableOfTxId == NULL ||
110          pRawData->IN_tableOfIndex == NULL ||
111          pRawData->IN_tableOfAddr == NULL ) return STATUS_BAD_ALLOC;
112
113     for(int i = 0; i < pRawData->inTx; i++)
114     {
115         pRawData->IN_tableOfTxId[i] = this->getHashFromStream(iss, ' ');
116         pRawData->IN_tableOfIndex[i] = this->getTxIndexFromStream(iss, ' ');
117         pRawData->IN_tableOfAddr[i] = this->getHashFromStream(iss);
118     }
119
120     pRawData->outTx = this->getTxIndexFromStream(iss, '\n');
121     pRawData->OUT_tableOfValues = new float[pRawData->outTx];
122     pRawData->OUT_tableOfAddr = new std::string[pRawData->outTx];
123     if(      pRawData->OUT_tableOfValues == NULL ||
124          pRawData->OUT_tableOfAddr == NULL ) return STATUS_BAD_ALLOC;
125
126     for(int i = 0; i < pRawData->outTx; i++)
127     {
128         pRawData->OUT_tableOfValues[i] = this->getBTCValueFromStream(iss, ' ');
129         pRawData->OUT_tableOfAddr[i] = this->getHashFromStream(iss);
130     }
131
132     pBuilderRawData = this->pRawData;
133
134     return STATUS_OK;
135 }
136
137
138 int BlockchainFileManager::getTxIndexFromStream(std::istream *iss, char delim)
139 {
140     int IndexValue;
141     std::string line;
142     std::stringstream ss;
143
144     std::getline(*iss, line, delim);
145     ss.str(line);
146     ss >> IndexValue;
147     return IndexValue;
148 }
149
150 std::string BlockchainFileManager::getHashFromStream(std::istream *iss, char delim)
151 {
152     std::string line;
153     std::stringstream ss;

```

```

154     std::getline(*iss, line, delim);
155     return line;
156 }
157
158 float BlockChainFileManager::getBTCValueFromStream(std::istream *iss, char delim)
159 {
160     float floatValue;
161     std::string line;
162     std::stringstream ss;
163
164     std::getline(*iss, line, delim);
165     ss.str(line);
166     ss >> floatValue;
167     return floatValue;
168 }
169
170
171 status_t BlockChainFileManager::convert(std::ostream * iss, const lista <Block *> & BlockChain
172 ) {
173     lista <Block *> ::iterador it(BlockChain);
174     std::string obtainedHash;
175
176     if(!iss->good())                return STATUS_BAD_READ_OUTPUT_FILE;
177     if( BlockChain.vacia() )        return STATUS_NO_BLOCKS_TO_CONVERT;
178     while(!it.extremo()){
179         *iss << it.dato()->getpre_block() << '\n';
180         *iss << it.dato()->gettxns_hash() << '\n';
181         *iss << it.dato()->getbits( )      << '\n';
182         *iss << it.dato()->getnonce()      << '\n';
183         *iss << it.dato()->RecalculoHash();
184         it.avanzar();
185     }
186     return STATUS_FINISH_CONVERT_SUCCESSFULLY;
187 }

```

```

1
2 #ifndef BLOCKCHAINFILEMANAGER_H_
3 #define BLOCKCHAINFILEMANAGER_H_
4
5 #include <iostream>
6 #include <sstream>
7 #include <ostream>
8 #include "BlockChainStatus.h"
9 #include "BlockChainBuilder.h"
10 #include "BlockChainDataTypes.h"
11
12 class BlockChainFileManager {
13 private:
14     raw_t * pRawData;
15
16     bool isTxIndexFromStream(std::istream *iss, char delim = '\n', int * pValue = NULL);
17     bool isHashFromStream(std::istream *iss, char delim = '\n', std::string * pString = NULL);
18     bool isBTCValueFromStream(std::istream *iss, char delim = '\n', float * pFloat = NULL);
19     bool isEofFromStream(std::istream *iss);
20     int  getTxIndexFromStream(std::istream *iss, char delim = '\n');
21     std::string getHashFromStream(std::istream *iss, char delim = '\n');
22     float getBTCValueFromStream(std::istream *iss, char delim = '\n');
23 public:
24     BlockChainFileManager();
25     ~BlockChainFileManager();
26     status_t validate(std::istream * iss);
27     status_t parse(std::istream * iss, raw_t * &pRawData);
28     status_t convert(std::ostream * oss, const lista <Block *> & BlockChain);
29 };
30
31 #endif /* BLOCKCHAINFILEMANAGER_H_ */

```

### 4.3. Clase BlockChainBuilder

La clase **BlockChainBuilder** es el corazón del proyecto, y como su nombre lo indica es el constructor o armador de la *BlockChain*. En esta clase se encuentra la verdadera lógica del código: la creación de la lista de bloques. Opera al mismo nivel lógico que **BlockChainFileManager**, siendo ambos "empleados" de **BlockChainManager**. 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 calculo del *nonce* y la utilización de la clase SHA256.

En un inicio, se propuso una clase Block capaz de instanciarse y realizar todas las tareas, pero a medida que se avanzaba sobre el código se hizo evidente la necesidad de una lógica superior capaz de trabajar con la lista de bloques. *BlockChainBuilder* concentra entonces la lógica principal y *Block* pasa a ser una clase contenedora reutilizable.

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 metodo *minando()*.

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

```
1
2
3
4 #include "BlockChainBuilder.h"
5
6 BlockChainBuilder::BlockChainBuilder() : BlocklActual(), ListaBlocks(), hash_resultado( "" ),
7     bits( 3 /* El valor por default establecido en el TP0 */, pRawData(NULL){}
8
9 BlockChainBuilder::BlockChainBuilder(size_t d) : BlocklActual(), ListaBlocks(), hash_resultado
10    ( "" ), bits( d ), pRawData(NULL){}
11
12 BlockChainBuilder::~BlockChainBuilder() {
13     if ( ! this->ListaBlocks.vacia() ){
14         // lista <Transaction>::iterador it();
15         lista <Block *>::iterador it(ListaBlocks);
16         /* Itero la lista para recuperar todos los strings de la coleccion Transaction
17            que necesito para calcular el Hash.
18         */
19         it = this->ListaBlocks.primer();
20         while ( ! it.extremo() ) {
21             delete it.dato();
22             it.avanzar();
23         }
24     }
25 }
26
27 int BlockChainBuilder::CheckHexa( string value ) {
28     unsigned int i;
29     for (i = 0; i != value.length(); ++i) {
30         if ( ! isxdigit ( value[i] ) ) break;
31     }
32     if ( i < value.length() ) return i;
33     return 0;
34 }
35
36 bool BlockChainBuilder::CheckHash( std::string valor, TiposHash Tipo ) {
37     if ( valor.empty() ) {
38         return false;
39     }
40     else if ( Tipo == TiposHash::clavehash256 && valor.length() != LargoHashEstandar ) {
41         return false;
42     }
43     else if ( Tipo == TiposHash::clavefirma && valor.length() != LargoHashFirma ) {
44         return false;
```

```

45     }
46     else {
47         int i = CheckHexa( valor );
48         if ( i > 0 ) {
49
50             return false;
51         }
52         else return true;
53     }
54 }
55
56
57 std::string BlockChainBuilder::Calculononce() {
58     static int contador = 0;
59     contador++;
60     return std::to_string( contador );
61 }
62
63
64 bool BlockChainBuilder::CalculoBits( std::string hash, size_t bits ) {
65
66     int test = BlockChainBuilder::CheckDificultadOk( hash, bits);
67     if ( test == 1 ) {
68         //std::cout << "Dificultad Ok < " << test << std::endl;
69         return true;
70     }
71     else if ( test < 0 ) {
72         //std::cout << "Error: " << test << std::endl;
73         return false;
74     }
75     else {
76         //std::cout << "Dificultad < " << test << std::endl;
77         return false;
78     };
79 }
80
81
82 bool BlockChainBuilder::Minando() {
83     std::string resultado = "",nonce;
84
85     if ( ! this->ListaBlocks.vacia() ) {
86         lista <Block *>::iterador it;
87         /* Itero la lista para recuperar todos los strings de la coleccion Transaction
88            que necesito para calcular el Hash.
89         */
90         it = this->ListaBlocks.primer();
91         do {
92             this->BlocklActual = it.dato();
93             do{
94                 this->BlocklActual->setnonce( BlockChainBuilder::Calculononce());
95                 resultado += this->BlocklActual->getpre_block();
96                 resultado += this->BlocklActual->gettxns_hash();    // <- falta definir el
mÃtodo que extrae el string en la Clase Transaction.
97                 resultado += this->BlocklActual->getnonce();
98                 resultado += this->BlocklActual->RecalculoHash();
99                 //if ( resultado.length() > 0 ) {
100                     this->hash_resultado = sha256( sha256( resultado ) );
101                     //}
102                 }while(! CalculoBits( this->hash_resultado, this->bits ) );
103                 it.avanzar();
104             } while ( ! it.extremo() );
105             return true;
106         }
107         return false;
108     }
109
110 const char* BlockChainBuilder::hex_char_to_bin( char c )
111 {
112     // TODO handle default / error
113     // https://stackoverflow.com/questions/18310952/convert-strings-between-hex-format-and-

```

```

binary-format
114 switch( toupper(c) )
115 {
116     case '0': return "0000";
117     case '1': return "0001";
118     case '2': return "0010";
119     case '3': return "0011";
120     case '4': return "0100";
121     case '5': return "0101";
122     case '6': return "0110";
123     case '7': return "0111";
124     case '8': return "1000";
125     case '9': return "1001";
126     case 'A': return "1010";
127     case 'B': return "1011";
128     case 'C': return "1100";
129     case 'D': return "1101";
130     case 'E': return "1110";
131     case 'F': return "1111";
132     default: return "";
133 }
134 }
135
136 std::string BlockchainBuilder::hex_str_to_bin_str( const std::string & hex )
137 {
138     // TODO use a loop from <algorithm> or smth
139     std::string bin;
140     std::string hexbin;
141     for( size_t i = 0; i != hex.length(); ++i ) {
142         hexbin = hex_char_to_bin( hex[i] );
143         if ( hexbin.empty() ) return "";
144         bin += hexbin;
145     }
146     return bin;
147 }
148
149 int BlockchainBuilder::dificultad( const std::string value, const size_t dif ) {
150     // Se corta el recorrido de la cadena una vez alcanzado el valor dif
151     size_t j = 0;
152
153     if ( value.empty() ) return -1;
154     else if ( dif == 0 ) return -1;
155
156     for ( size_t i = 0; value[ i ]; i++ ) {
157         if ( value[ i ] == '0' ) j++;
158         else if ( value[ i ] == '1' ) break;
159         else return -1;
160         if ( j++ >= dif ) break;
161     }
162     return j;
163 }
164
165 int BlockchainBuilder::CheckDificultadOk( std::string cadenaHexa, const size_t dif ) {
166     int d;
167     if ( cadenaHexa.empty() ) return -3;
168     if ( dif == 0 ) return -2;
169     d = dificultad( cadenaHexa, dif );
170     if ( d < 0 ) return -1;
171     return (size_t) d >= dif ? 1 : 0;
172 }
173
174
175 status_t BlockchainBuilder::createBlockChain( void ){
176     Block * newBlock = new Block(*pRawData);
177     newBlock->setpre_block( "ffffffffffffffffffffffffffffffffffffffffffffffffffffffff"
178 );
179     newBlock->settxns_hash(sha256(sha256(newBlock->RecalculoHash())));
180     newBlock->setbits(this->bits);
181     this->ListaBlocks.insertar(newBlock);
182     this->Minando();

```

```

182     return STATUS_OK;
183 }

```

```

1
2
3 #ifndef BLOCKCHAINBUILDER_H_
4 #define BLOCKCHAINBUILDER_H_
5
6 #include "TiposHash.h"
7 #include "BlockChainDataTypes.h"
8 #include "BlockChainStatus.h"
9 #include "Block.h"
10 #include "lista.h"
11 #include "sha256.h"
12
13
14 class BlockChainBuilder {
15 private:    // Redundante pero m  s legible
16     /* Anterior */
17     static int CheckHexa( std::string value ); // <- esta le ser  a m  s util a
18     BlockChainFileManager
19     /* Datos privados */
20     Block * BlocklActual;
21     lista <Block *> ListaBlocks;
22     std::string hash_resultado;
23     size_t bits;    /* La dificultad de bits */
24     /* Nuevo */
25     raw_t * pRawData; // raw_t es el dato raw que devuelve filemanager. De aca builder saca
26     los datos
27     bool CalculoBits( std::string hash, size_t bits );
28     bool Minando();
29     static std::string hex_str_to_bin_str( const std::string & hex );
30     static const char* hex_char_to_bin( char c );
31     static int dificultad( const std::string value, const size_t dif ); //
32     -1 -> Error
33
34 public:
35     BlockChainBuilder();
36     BlockChainBuilder(size_t d);
37     virtual ~BlockChainBuilder();
38     /* Getters */
39     unsigned int getbits();
40     std::string getObtainedHash(){return hash_resultado;};
41     raw_t *& getRawPointer(){return pRawData;}
42     lista <Block *> getBlockChainPointer(){return ListaBlocks;};
43     /* Setters */
44     bool setbits( unsigned int valor );
45     /* M  todos */
46     unsigned int cantidadBlocks();
47     static int CheckDificultadOk( const std::string cadenaHexa, const size_t dif ); // Error
48     -> < 0, No -> 0, 0k -> 1
49     static bool CheckHash( std::string valor, TiposHash Tipo = TiposHash::clavehash256 );
50     static std::string Calculononce();
51     status_t createBlockChain(void);
52
53 };
54
55 #endif /* BLOCKCHAINBUILDER_H_ */

```

## 4.4. Clase Block

```

1
2
3 #include "Block.h"

```

```

4
5
6 // Constructores
7 Block::Block()
8 : pre_block(""), txns_hash(""), bits(3 /* El valor por default establecido en el TP0 */),
9   nonce(0), eBlock(StatusBlock::BlockSinDatos), txn_count(0), CurTran(NULL)
10 // ver el #define DIFFICULTY_DEFAULT_VALUE 3
11 {
12     //this->ListaTran = NULL;
13     // this->CurTran = NULL;
14     // this->txn_count = 0;
15     // this->eBlock = StatusBlock::BlockSinDatos;
16 }
17
18 Block::Block( const raw_t & raw )
19 : pre_block(""), txns_hash(""), bits( 3 /* El valor por default establecido en el TP0 */),
20   nonce(0), eBlock(StatusBlock::BlockSinDatos)
21 {
22     /* Basicamente:
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 deberia 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 dinamico vector.h raw_t?
33         En este caso se recibe solo un raw_t, igualmente lo cargo en una lista, para
34         hacerlo mas generico.
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 habra que iterar pasando el mismo tipo de parametros 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
54 Block::~Block() {
55     // ListaTran se autodestruye, antes debo liberar la memoria asignada en cada elemento *
56     // ListaTran de la lista
57     if ( ! this->ListaTran.vacia() ) {
58
59         lista <Transaction *>::iterador it(ListaTran);
60         /* Itero la lista para recuperar todos los strings de la coleccion Transaction
61            que necesito para calcular el Hash.
62         */
63         it = this->ListaTran.primer();
64         while ( ! it.extremo() ) {
65             delete it.dato();
66             it.avanzar();
67         }
68     }
69 }
70
71 // Getters
72 unsigned int Block::gettxn_count() {
73     return this->txn_count;
74 }

```

```

65 }
66
67 std::string Block::getpre_block() {
68     return this->pre_block;
69 }
70
71 std::string Block::gettxns_hash() {
72     return this->txns_hash;
73 }
74
75 unsigned int Block::getbits() {
76     return this->bits;
77 }
78
79 unsigned int Block::getnonce() {
80     return this->nonce;
81 }
82
83 std::string Block::getcadenaprehash() {
84     return this->cadena_prehash;
85 }
86
87 // Setters
88 bool Block::setpre_block( std::string valor ) {
89     if ( valor.empty() ) {
90         this->pre_block = "";
91         // Hay que anotar, en un status ?, el error o disparar un throw
92     }
93     else {
94         /* 1) Debo validar que sea una cadena de 32 bytes o 64 digitos Hexa
95            2) Chequear que cada byte sea un caracter hexa valido.
96            2) Chequear que cada byte sea un caracter hexa valido. Se elimina se supone que
97               vien externamente validado.
98            if ( BlockChainBuilder::CheckHash( valor, TiposHash::clavehash256 ) ) {
99                this->pre_block = valor;
100            */
101            this->pre_block = valor;
102        }
103        return true;
104    }
105
106 bool Block::settxns_hash( std::string valor ) {
107     if ( valor.empty() ) {
108         this->txns_hash = "";
109         // Hay que anotar, en un status ?, el error o disparar un throw
110     }
111     else {
112         /* 1) Debo validar que sea una cadena de 32 bytes o 64 digitos Hexa
113            2) Chequear que cada byte sea un caracter hexa valido. Se elimina se supone que
114               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 }

```



```

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

```

La clase *Block* representa el nodo de la lista donde se guarda la información. Eventualmente sera utilizado para la creación de cadenas. Como se menciona, consta de métodos para su acceso destacando el metodo *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*.

```

1
2 #ifndef BLOCK_H_
3 #define BLOCK_H_
4
5 #include <cstdlib>
6 #include <string>
7 #include "lista.h"
8 #include "TiposHash.h"
9 #include "Transaction.h"

```

```

10
11 #include "BlockchainDataTypes.h"
12
13 // const size_t LargoHashEstandar = 64;
14 // const size_t LargoHashFirma = 40; // Hash Publica de la Cuenta
15 // https://stackoverflow.com/questions/2268749/defining-global-constant-in-c
16 // Analisis de Pro vs Contras contra #define y otras formas
17
18 using namespace std;
19
20 class Block {
21     private:
22         // Atributos Seccion Header
23         std::string pre_block;
24         std::string txns_hash; // <- retiene el hash256(hash256(cadena_prehash))
25         unsigned int bits; /* La dificultad de bits */
26         unsigned int nonce;
27         StatusBlock eBlock;
28         // Atributos Seccion Body;
29         unsigned int txn_count;
30         lista <Transaction *> ListaTran;
31         Transaction * CurTran;
32         std::string cadena_prehash;
33         // Metodos privados
34         std::string RecalculoHash( void );
35
36     public:
37         // Metodos
38         // Constructores
39         Block();
40         Block( const raw_t & raw );
41         //Block( const & std::string previo_block, size_t bits, const & raw_t );
42         // size_t bits sale de BlockchainManager::getUserDefinedDifficulty(void), pero
43         // referenciar a esta clase implica un encastramiento indeseado.
44         // Destructor
45         ~Block();
46         // Getters
47         unsigned int gettxn_count();
48         std::string getpre_block();
49         std::string gettxns_hash();
50         unsigned int getbits();
51         unsigned int getnonce();
52         std::string getcadenaprehash();
53         // Setters
54         bool setpre_block( std::string valor );
55         bool settxns_hash( std::string valor ); // Debo dejar el metodo de asignacion. El
56         // calculo Hash es externo al objeto block, no esta encapsulado.
57         bool setbits( unsigned int valor );
58         bool setnonce( int valor ); // Debo dejar el metodo de asignacion. El calculo
59         // del Nonce es externo al objeto block, no esta encapsulado.
60         bool settransaction( const raw_t & raw ); // TODO
61         StatusBlock EstatusBlock();
62 };
63
64 #endif /* BLOCK_H_ */

```

## 4.5. Clase Transaction

La clase *Transaction* al igual que la clase *Block* son clases contendoras. Particularmente Builder invoca a

```

1 #include "Transaction.h"
2
3
4 /*---Constructores---*/
5
6 /*Descripcion: Instancia el objeto Transaction vacio
7 //Precondicion: -

```

```

8 //Postcondicion: La lista de transacciones de entrada y salida apuntan a NULL*/
9
10 Transaction::Transaction() {
11     this->n_tx_in = 0;
12     this->n_tx_out = 0;
13 }
14
15 /* Descripcion: Instancia el objeto Transaction a partir de un archivo raw_t
16
17 // Precondicion:
18
19 // Postcond dos punteros a memoria de tamano definido
20 // precargados con los datos de raw_t*/
21
22 Transaction::Transaction( const raw_t & Raw ){
23     this->n_tx_in = Raw.inTx;
24     for(int i = 0; i < this->n_tx_in ;i++ )
25     {
26         try {
27             TransactionInput * pTxInput = new TransactionInput;
28             pTxInput->setTxId(Raw.IN_tableOfTxId[i]);
29             pTxInput->setIdx(Raw.IN_tableOfIndex[i]);
30             pTxInput->setAddr(Raw.IN_tableOfAddr[i]);
31             this->ListaTranIn.insertar(pTxInput);
32         }
33         catch (std::bad_alloc& ba)
34         {
35             std::cerr << "bad_alloc caught: " << ba.what() << '\n';
36         }
37     }
38     this->n_tx_out = Raw.outTx;
39     for(int i = 0; i < this->n_tx_out ;i++ )
40     {
41         try {
42             TransactionOutput * pTxOutput = new TransactionOutput;
43             pTxOutput->setValue(Raw.OUT_tableOfValues[i]);
44             pTxOutput->setAddr(Raw.OUT_tableOfAddr[i]);
45             this->ListaTranOut.insertar(pTxOutput);
46         }
47         catch (std::bad_alloc& ba)
48         {
49             std::cerr << "bad_alloc caught: " << ba.what() << '\n';
50         }
51     }
52 }
53
54
55 /*Descripcion: Destruye elemento de Transaction
56 //Precondicion: Si se envia una transaccion nula no es necesario que se realice accion
57 //Postcondicion: Objeto destruido, memoria liberada, punteros a null y parametros a cero.*/
58
59 Transaction::~Transaction(){
60     if ( ! this->ListaTranIn.vacia() ) {
61         lista <TransactionInput *>::iterador it(ListaTranIn);
62         it = this->ListaTranIn.primer();
63         do{
64             delete it.dato();
65             it.avanzar();
66         }while ( ! it.extremo() );
67     }
68     if ( ! this->ListaTranOut.vacia() ) {
69         lista <TransactionOutput *>::iterador it(ListaTranOut);
70         it = this->ListaTranOut.primer();
71         do {
72             delete it.dato();
73             it.avanzar();
74         }while ( ! it.extremo() );
75     }
76 }
77

```

```

78  /*---Getters---//
79
80  //Descripcion: Devuelve cantidad de transacciones de input
81  //Precondicion:
82  //Postcondicion: */
83
84  int Transaction::getNumTransactionIn(){
85      return this->n_tx_in;
86  }
87
88  /*Descripcion: Devuelve cantidad de transacciones de output
89  //Precondicion:
90  //Postcondicion: */
91
92  int Transaction::getNumTransactionOut(){
93      return this->n_tx_out;
94  }
95
96  /*Descripcion: Obtiene la transaccion de la lista de entradas
97  //Precondicion: Si el indice esta fuera de rango debe devolver null
98  //Postcondicion: */
99
100 TransactionInput * Transaction::getTransactionInput(int index){
101     size_t index_ = (size_t)index;
102     if( index < 0 || index_ > this->ListaTranIn.tamano())
103         return NULL;
104     else{
105         lista <TransactionInput *>::iterador it(this->ListaTranIn);
106         int counter = 0;
107         while(counter != index){
108             it.avanzar();
109             counter++;
110         }
111         return it.dato();
112     }
113 }
114
115 /*Descripcion: Obtiene la transaccion de la lista de salidas
116 //Precondicion: Si el indice esta fuera de rango debe devolver null
117 //Postcondicion: */
118
119 TransactionOutput * Transaction::getTransactionOutput(int index){
120     size_t index_ = (size_t)index;
121     if( index < 0 || index_ > this->ListaTranOut.tamano())
122         return NULL;
123     else{
124         lista <TransactionOutput *>::iterador it(this->ListaTranOut);
125         int counter = 0;
126         while(counter != index){
127             it.avanzar();
128             counter++;
129         }
130         return it.dato();
131     }
132 }
133
134 /*Descripcion: Devuelve un string de los valores concatenados de la listas
135 //para ser aplicado el hash correspondiente por fuera
136 //Precondicion: Se considera todo precargado antes
137 //Postcondicion: */
138
139
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     concatenation << this->n_tx_in << '\n';
145     for(itIn = ListaTranIn.primer(); !itIn.extremo() ; itIn.avanzar()){
146         concatenation<< itIn.dato()->getTxId() <<' ';
147         concatenation<< itIn.dato()->getIdx() <<' ';

```

```

148         concatenation<< itIn.dato()->getAddr() <<'\\n';
149     }
150     concatenation << this->n_tx_out << '\\n';
151     for(itOut = ListaTranOut.primerO(); !itOut.extremo() ; itOut.avanzar()){
152         concatenation<< itOut.dato()->getValue() <<' ';
153         concatenation<< itOut.dato()->getAddr() <<'\\n';
154     }
155     /* std::cout <<concatenation.str()<<std::endl; //DEBUG
156     return concatenation.str();
157 }

```

```

1
2
3 #ifndef TRANSACTION_H_
4 #define TRANSACTION_H_
5
6 #include "TransactionInput.h"
7 #include "TransactionOutput.h"
8 #include "BlockchainDataTypes.h"
9 #include "lista.h"
10 #include <iostream>
11 #include <sstream>
12 #include <cstdint> // Para NULL
13
14 class Transaction {
15 private:
16     int n_tx_in; // Indica cantidad total de inputs
17     lista <TransactionInput *> ListaTranIn; // Lista de inputs
18     int n_tx_out; // Indica cantidad total de outputs
19     lista <TransactionOutput *> ListaTranOut; // Lista de outputs
20 public:
21     ///---Constructores---//
22     Transaction();
23     Transaction(int n_tx_in,int n_tx_out);
24     Transaction( const raw_t & raw);
25     ~Transaction();
26     ///---Getters---//
27     int getNumTransactionIn();
28     int getNumTransactionOut();
29     TransactionInput * getTransactionInput(int index);
30     TransactionOutput * getTransactionOutput(int index);
31     ///---Setters---//
32     ///---Otros---//
33     std::string getConcatenatedTransactions();
34 };
35
36 #endif /* TRANSACTION_H_ */

```

## 4.6. Clase TransactionInput

```

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

```

```

15 //Descripcion: Destruye el objeto TransactionInput
16 //Precondicion:
17 //Postcondicion: Todos los parametros iniciados en 0 o vacio
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 //Descripcion: Carga el atributo idx
58 //Precondicion: Se asume validado previamente
59 //Postcondicion:
60 void TransactionInput::setIdx(int idx){
61     this->outpoint.idx = idx;
62 }
63
64 //Descripcion: Carga el atributo addr
65 //Precondicion: Se asume validado previamente
66 //Postcondicion:
67 void TransactionInput::setAddr(std::string addr){
68     this->addr = addr;
69 }
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_ */

```

## 4.7. Clase TransactionOutput

```

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_ */

```

## 4.8. Clases sha256 cmdline

Las clases **sha256** y **cmdline** fueron provistas por la cátedra. La clase **sha256** es utilizada por la clase **BlockchainBuilder** para calcular 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. 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 manteniendo al proyecto ordenado y agilizando su desarrollo. Se ejecuta desde la terminal de LINUX o WINDOWS simplemente insertando el comando **make all**.

```

ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/TP1/Master/algoritmos-ii-tp1/TP1_DEF/src$ make clean
rm -f core main.exe Complejo.o main.o cmdline.o ComplexPlane.o ComplexTransform.o Images.o Token.o
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/TP1/Master/algoritmos-ii-tp1/TP1_DEF/src$ make all
g++ -std=c++11 -g -Wall -c Complejo.cpp
g++ -std=c++11 -g -Wall -c main.cpp
g++ -std=c++11 -g -Wall -c cmdline.cpp
g++ -std=c++11 -g -Wall -c ComplexPlane.cpp
g++ -std=c++11 -g -Wall -c ComplexTransform.cpp
g++ -std=c++11 -g -Wall -c Images.cpp
g++ -std=c++11 -g -Wall -c Token.cpp
g++ -std=c++11 -g -Wall -o main.exe Complejo.o main.o cmdline.o ComplexPlane.o ComplexTransform.o Images.o Token.o
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/TP1/Master/algoritmos-ii-tp1/TP1_DEF/src$

```

Figura 5.1: Compilación usando make clean y make all

makefile.



```

2 | OBJECTS = main.o sha256.o cmdline.o TransactionInput.o TransactionOutput.o Transaction.o Block
   | .o BlockchainBuilder.o BlockchainFileManager.o BlockchainManager.o # Los archivos
   | compilados individuales
3 | PROGR = miner.exe # Nombre del archivo ejecutable
4 | CPPFLAGS = -g -Wall -pedantic -Werror# -g opcion de g++ para debugear
5 |
6 | # Compiladores #
7 | CC = g++ -std=c++17 # Para linux
8 | CCW = i686-w64-mingw32-g++ --static # Para windows (requiere mingw32)
9 |
10 | $(PROGR) : $(OBJECTS)
11 |     $(CC) $(CPPFLAGS) -o $(PROGR) $(OBJECTS)
12 | main.o : main.cpp cmdline.h BlockchainManager.h
13 |     $(CC) $(CPPFLAGS) -c main.cpp
14 | cmdline.o : cmdline.cc cmdline.h
15 |     $(CC) $(CPPFLAGS) -c cmdline.cc
16 | sha246.o : sha246.cpp sha246.h
17 |     $(CC) $(CPPFLAGS) -c sha246.cpp
18 | BlockchainManager.o : BlockchainManager.cpp BlockchainManager.h BlockchainFileManager.h
   | BlockchainBuilder.h BlockchainStatus.h
19 |     $(CC) $(CPPFLAGS) -c BlockchainManager.cpp
20 | BlockchainFileManager.o : BlockchainFileManager.cpp BlockchainFileManager.h BlockchainBuilder.
   | h BlockchainDataTypes.h BlockchainStatus.h
21 |     $(CC) $(CPPFLAGS) -c BlockchainFileManager.cpp
22 | BlockchainBuilder.o : BlockchainBuilder.cpp BlockchainBuilder.h lista.h sha256.h Block.h
   | TiposHash.h BlockchainDataTypes.h BlockchainStatus.h
23 |     $(CC) $(CPPFLAGS) -c BlockchainBuilder.cpp
24 | Block.o : Block.cpp Block.h Transaction.h lista.h TiposHash.h BlockchainDataTypes.h
25 |     $(CC) $(CPPFLAGS) -c Block.cpp
26 | Transaction.o : Transaction.cpp Transaction.h lista.h TransactionOutput.h TransactionInput.h
   | BlockchainDataTypes.h
27 |     $(CC) $(CPPFLAGS) -c Transaction.cpp
28 | TransactionOutput.o : TransactionOutput.cpp TransactionOutput.h
29 |     $(CC) $(CPPFLAGS) -c TransactionOutput.cpp
30 | TransactionInput.o : TransactionInput.cpp TransactionInput.h
31 |     $(CC) $(CPPFLAGS) -c TransactionInput.cpp
32 |
33 |
34 |
35 |
36 | clean:
37 |     rm -f core $(PROGR) $(OBJECTS)
38 | all: $(PROGR)
39 |     $(CC) $(CPPFLAGS) -o $(PROGR) $(OBJECTS)
40 | run : $(PROGR)
41 |     valgrind --leak-check=yes ./$(PROGR) -i *.txt -o - -d 3

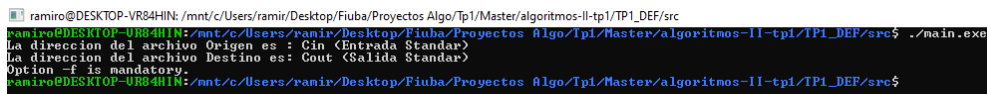
```

## 6. Pruebas sobre el programa

A continuación se presentan las distintas formas en que se comporta el programa para distintos parámetros. Se incluyen pruebas en Linux.

### 6.1. Sin parámetros

```
1 ./main.exe
2 La direccion del archivo Origen es : Cin (Entrada Standar)
3 La direccion del archivo Destino es: Cout (Salida Standar)
4 Option -f is mandatory.
```

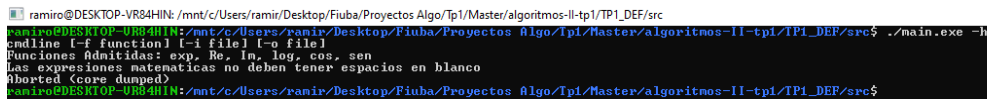


```
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/TP1/Master/algoritmos-II-tp1/TP1_DEF/src$ ./main.exe
La direccion del archivo Origen es : Cin (Entrada Standar)
La direccion del archivo Destino es: Cout (Salida Standar)
Option -f is mandatory.
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/TP1/Master/algoritmos-II-tp1/TP1_DEF/src$
```

Figura 6.1: Prueba sin parámetros en Linux

### 6.2. Archivo de entrada en otro formato (no .txt)

```
1 ./main.exe -h
2 cmdline [-f function] [-i file] [-o file]
3 Funciones Admitidas: exp, Re, Im, log, cos, sen
4 Las expresiones matematicas no deben tener espacios en blanco
5 Aborted (core dumped)
```

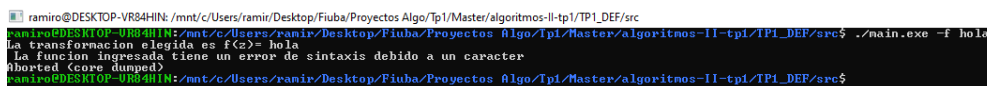


```
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/TP1/Master/algoritmos-II-tp1/TP1_DEF/src$ ./main.exe -h
cmdline [-f function] [-i file] [-o file]
Funciones Admitidas: exp, Re, Im, log, cos, sen
Las expresiones matematicas no deben tener espacios en blanco
Aborted (core dumped)
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/TP1/Master/algoritmos-II-tp1/TP1_DEF/src$
```

Figura 6.2: Prueba opción Help en Linux

### 6.3. Parámetro No-Válido

```
1 ./main.exe -f hola
2 La transformacion elegida es f(z)= hola
3 La funcion ingresada tiene un error de sintaxis debido a un caracter
4 Aborted (core dumped)
```



```
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/TP1/Master/algoritmos-II-tp1/TP1_DEF/src$ ./main.exe -f hola
La transformacion elegida es f(z)= hola
La funcion ingresada tiene un error de sintaxis debido a un caracter
Aborted (core dumped)
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/TP1/Master/algoritmos-II-tp1/TP1_DEF/src$
```

Figura 6.3: Prueba parámetro Invalido en Linux

### 6.4. Input inexistente

```
1 ./main.exe -i inexistente.pgm -f z
2 La direccion del archivo Origen es :inexistente.pmg
3 cannot open inexistente.pmg.
4 Aborted (core dumped)
```

```

ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$ ./main.exe -i inexistente.png -f z
La direccion del archivo Origen es :inexistente.png
cannot open inexistente.png.
Aborted (core dumped)
ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$

```

Figura 6.4: Prueba Entrada inexistente en Linux

## 6.5. input.txt vacío

```

1 ./main.exe -i dragon.ascii.png -f z
2 La direccion del archivo Origen es :dragon.ascii.png
3 La transformacion elegida es f(z)= z
4 La funcion se ingreso correctamente
5 La transformacion elegida es f(z)= z
6 La direccion del archivo Destino es: Cout (Salida Standar)
7 Procesando imagen...
8 Formato no .pgm

```

```

ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$ ./main.exe -i dragon.ascii.png -f z
La direccion del archivo Origen es :dragon.ascii.png
La transformacion elegida es f(z)= z
La funcion se ingreso correctamente
La transformacion elegida es f(z)= z
La direccion del archivo Destino es: Cout (Salida Standar)
Procesando imagen...
Formato no .pgm
ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$

```

Figura 6.5: Prueba Entrada no PGM en Linux

## 6.6. No existe archivo en input con ese nombre

A continuación se presentan algunos errores sintácticos al escribir las expresiones matemáticas, dada que las combinaciones de errores son infinitas se muestran solo algunas.

```

ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$ ./main.exe -i dragon.ascii.pgm -f ".z"
La direccion del archivo Origen es :dragon.ascii.pgm
La transformacion elegida es f(z)= .z
La funcion Ingresada tiene un error de sintaxis debido a un caracter
Aborted (core dumped)
ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$ ./main.exe -i dragon.ascii.pgm -f "1.z"
La direccion del archivo Origen es :dragon.ascii.pgm
La transformacion elegida es f(z)= 1.z
La funcion Ingresada tiene un error de sintaxis debido a un caracter
Aborted (core dumped)
ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$ ./main.exe -i dragon.ascii.pgm -f "1+z"
La direccion del archivo Origen es :dragon.ascii.pgm
La transformacion elegida es f(z)= 1+z
La funcion Ingresada tiene un error de sintaxis debido a un caracter
Aborted (core dumped)
ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$ ./main.exe -i dragon.ascii.pgm -f "log(z"
La direccion del archivo Origen es :dragon.ascii.pgm
La transformacion elegida es f(z)= log(z
Se detecto la funcion log
La funcion Ingresada no esta balanceada
Aborted (core dumped)

```

Figura 6.6: Prueba Errores Sintacticos en Linux -1

```

ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$ ./main.exe -i dragon.ascii.pgm -f "Hola(z)"
La direccion del archivo Origen es :dragon.ascii.pgm
La transformacion elegida es f(z)=Hola(z)
Se detecto la funcion Hola
Hola No es una funcion valida : Escriba "h" para ayuda y ver las funciones validas
La funcion Ingresada tiene un error de sintaxis debido a un caracter
Aborted (core dumped)
ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$ ./main.exe -i dragon.ascii.pgm -f "z**"
La direccion del archivo Origen es :dragon.ascii.pgm
La transformacion elegida es f(z)=z**
La funcion Ingresada tiene un error de sintaxis debido a un caracter
Aborted (core dumped)
ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$ ./main.exe -i dragon.ascii.pgm -f "z^"
La direccion del archivo Origen es :dragon.ascii.pgm
La transformacion elegida es f(z)=z^
La funcion Ingresada tiene un error de sintaxis debido a un caracter
Aborted (core dumped)
ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$ ./main.exe -i dragon.ascii.pgm -f "z/"
La direccion del archivo Origen es :dragon.ascii.pgm
La transformacion elegida es f(z)=z/
La funcion Ingresada tiene un error de sintaxis debido a un caracter
Aborted (core dumped)
ramiro@DESKTOP-UB44H1N:~/mnt/c/Users/ramir/Desktop/Fiuba/Proyectos_Algo/Tpl/Master/algoritmos-II-tpl/TPI_DEF/src$ ./main.exe -i dragon.ascii.pgm -f "z1"
La direccion del archivo Origen es :dragon.ascii.pgm
La transformacion elegida es f(z)=z1
La funcion Ingresada tiene un error de sintaxis debido a un caracter
Aborted (core dumped)

```

Figura 6.7: Prueba Errores Sintacticos en Linux -2

## 6.7. Imágenes de Prueba y Transformaciones

A continuación se muestran dos secciones:

- La primera mostrara las distintas transformaciones para la misma imagen
- La segunda mostrara una transformación en particular para muchas imágenes

Con estas dos secciones se desea mostrar la variedad de transformaciones e imágenes soportadas por el programa.

## 7. Conclusión

En esta instancia del trabajo, se creó una base funcional que expresa en su ejecución el concepto de la tecnología Blockchain al lograr simular el ensamblaje de una unidad funcional de la misma; utilizando para su implementación el paradigma de objetos. Dentro de los acontecimientos destacables se puede mencionar que, durante la fase de diseño surgió una diferencia de criterio en virtud de los límites entre las clases a la hora de procesar y convertir información. Particularmente, ésto se dio entre **BlockChainFileManager** y **BlockChainBuidier** en lo que respecta al parseo de la información de entrada. Cómo se expuso en la sección correspondiente, se optó por que dicha tarea sea llevada a cabo por **BlockChainFileManager**, siguiendo un criterio conceptual. 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.

## 8. Anexo I

### 8.1. Enunciado

# 75.04/95.12 Algoritmos y Programación II

## Trabajo práctico 0: programación C++

Universidad de Buenos Aires - FIUBA  
Segundo cuatrimestre de 2020

### 1. Objetivos

Ejercitar conceptos básicos de programación C++, implementando un programa y su correspondiente documentación que resuelva el problema descripto 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 impreso 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

**Bitcoin** es, posiblemente, la criptomoneda más importante de la actualidad. Los trabajos prácticos de este cuatrimestre están destinados a comprender los detalles técnicos más relevantes detrás de Bitcoin –en particular, la tecnología de **blockchain**. Para ello, trabajaremos con **ALGOCHAIN**, una simplificación de la blockchain orientada a capturar los conceptos esenciales de la tecnología.

En este primer acercamiento al problema, nos abocaremos a leer y procesar **transacciones** y ensamblar un **bloque** a partir de estas. Estos conceptos serán debidamente introducidos en la Sección 4.1, donde daremos una breve introducción a Bitcoin y blockchain. Una vez hecho esto, presentaremos la **ALGOCHAIN** en la Sección 4.2, destacando al mismo tiempo las similitudes y diferencias más importantes con la blockchain propiamente dicha. Las tareas a realizar en el presente trabajo se detallan en la Sección 4.3.

## 4.1. Introducción a Bitcoin y blockchain

Una *criptomoneda* es un activo digital que actúa como medio de intercambio utilizando tecnología criptográfica para asegurar la autenticidad de las transacciones. Bitcoin es, tal vez, la criptomoneda más importante en la actualidad. Propuesta en 2009 por una persona (o grupo de personas) bajo el seudónimo *Satoshi Nakamoto* [2], se caracterizó por ser la primera criptomoneda descentralizada que propuso una solución al problema de *double-spending* sin involucrar una tercera parte de confianza<sup>1</sup>. La idea esencial (y revolucionaria) que introdujo Bitcoin se basa en un registro descentralizado de todas las transacciones procesadas en el que cualquiera puede asentar operaciones. Este registro, replicado y distribuido en cada nodo de la red, se conoce como **blockchain**.

La blockchain no es otra cosa que una lista enlazada de *bloques*<sup>2</sup>. Los bloques agrupan *transacciones* y son la unidad básica de información de la blockchain (i.e., son los nodos de la lista). Cuando un usuario introduce una nueva transacción  $t$  en la red, las propiedades de la blockchain garantizan una detección eficiente de cualquier otra transacción que extraiga los fondos de la misma operación referenciada por  $t$ . En caso de que esto sucediera, se considera que el usuario está intentando hacer *double-spending* y la transacción es consecuentemente invalidada por los nodos de la red.

En lo que sigue describiremos los conceptos más importantes detrás de la blockchain. La Figura 1 provee un resumen visual de todo estos conceptos en el marco de la ALGOCHAIN.

### 4.1.1. Funciones de hash criptográficas

Una *función de hash criptográfica* es un algoritmo matemático que toma una cantidad arbitraria de bytes y computa una tira de bytes de una longitud fija (en adelante, un *hash*). Es importante que dichas funciones sean *one-way*, en el sentido de que sea computacionalmente inviable hacer una “ingeniería reversa” sobre la salida para reconstruir una posible entrada. Otra propiedad que suelen tener dichas funciones es un *efecto avalancha* en el que cambios incluso en bits aislados de la entrada derivan en hashes significativamente diferentes.

Bitcoin emplea la función de hash SHA256, ampliamente utilizada en una gran variedad de protocolos de autenticación y encrición [3]. Esta genera una salida de 32 bytes que usualmente se representa mediante 64 dígitos hexadecimales. A modo de ejemplo, el valor de  $\text{SHA256}(\text{'Sarasa.'})$  es

9c231858fa5fef160c1e7ecfa333df51e72ec04e9c550a57c59f22fe8bb10df2

### 4.1.2. Direcciones y firmas digitales

Una *dirección* de Bitcoin es básicamente un hash de 160 bits de la clave pública de un usuario. Mediante algoritmos criptográficos asimétricos, los usuarios pueden generar pares de claves mutuamente asociadas (pública y privada). La *clave privada* se emplea para *firmar*

<sup>1</sup>El doble gasto (o *double-spending*) ocurre cuando el emisor del dinero crea más de una transacción a partir de una misma operación previa. Naturalmente, sólo una de las nuevas transacciones debería ser válida puesto que, de lo contrario, el emisor estaría multiplicando dinero.

<sup>2</sup>Técnicamente, la blockchain es más bien un árbol de bloques, pero esto será abordado en el contexto del siguiente trabajo práctico.

los mensajes que desean transmitirse. Cualquier receptor puede luego verificar que la firma es válida utilizando la *clave pública* asociada. Esta clase de métodos criptográficos ofrecen garantías de que es computacionalmente difícil reconstruir la clave privada a partir de la información públicamente disponible.

#### 4.1.3. Transacciones

Una *transacción* en Bitcoin está definida por una lista de entradas (*inputs*) y otra de salidas (*outputs*). Un *output* se representa a través de un par (*value*, *addr*), donde *value* indica la cantidad de bitcoins que recibirá el destinatario y *addr* es el hash criptográfico de la clave pública del destinatario. Por otro lado, un *input* puede entenderse como una tupla (*tx\_id*, *idx*, *key*, *sig*) tal que:

- *tx\_id* indica el hash de una transacción previa de la que esta nueva transacción toma fondos,
- *idx* es un índice dentro de la secuencia de *outputs* de dicha transacción (los fondos de este *input*, luego, provienen de dicho *output*),
- *key* es la clave pública asociada a tal *output*, y
- *sig* es la firma digital del hash de la transacción usando la clave privada asociada a la clave pública del *output*.

En consecuencia, cada *input* hace referencia a un *output* anterior en la blockchain (los campos *tx\_id* y *idx* suelen agruparse en una estructura común bajo el nombre de *outpoint*). Para verificar que el uso de dicho *output* es legítimo, se calcula el hash de la clave pública y se verifica que sea igual a la que figura en el *output* utilizado. Luego, basta con verificar la firma digital con esa clave pública para asegurar la autenticidad de la operación.

Para garantizar la validez de una transacción, es importante verificar no sólo que cada *input* es válido sino también que la suma de los *outputs* referenciados sea mayor o igual que la suma de los *outputs* de la transacción. La diferencia entre ambas sumas, en caso de existir, es lo que se conoce como *transaction fee*. Este valor puede ser reclamado por quien agrega la transacción a la blockchain (como retribución por suministrar poder de cómputo para realizar el minado de un nuevo bloque).

Naturalmente, una vez que un *output* de una transacción haya sido utilizado, este no podrá volver a utilizarse en el futuro. En otras palabras, cada nueva transacción sólo puede referenciar *outputs* que no fueron utilizados previamente. Estos últimos se conocen como *unspent transaction outputs* (UTXOs).

#### 4.1.4. Bloques

Toda transacción de Bitcoin pertenece necesariamente a un *bloque*. Cada bloque está integrado por un encabezado (*header*) y un cuerpo (*body*). En el header se destaca la siguiente información:

- El hash del bloque antecesor en la blockchain (*prev\_block*),

- El hash de todas las transacciones incluidas en el bloque (`txns_hash`),
- La *dificultad* con la cual este bloque fue ensamblado (`bits`), y
- Un campo en el que se puede poner datos arbitrarios, permitiendo así alterar el hash resultante (`nonce`).

El cuerpo de un bloque, por otro lado, incluye la cantidad total de transacciones (`txn_count`) seguido de la secuencia conformada por dichas transacciones (`txns`).

#### 4.1.5. Minado de bloques

Para que un bloque sea válido y pueda en consecuencia ser aceptado por la red de Bitcoin, debe contar con una prueba de trabajo (*proof-of-work*) que debe ser difícil de calcular y, en simultáneo, fácil de verificar. El mecanismo detrás de este proceso se conoce como *minado*. Las entidades encargadas de agrupar transacciones y ensamblar bloques válidos son los *mineros*.

Los mineros obtienen una recompensa en bitcoins cuando agregan un bloque a la blockchain. Esto último se logra calculando la *proof-of-work* del nuevo bloque, lo cual a su vez se realiza con poder de cómputo. La *proof-of-work* de un bloque consiste en un hash  $h = \text{SHA256}(\text{SHA256}(\text{header}))$  tal que su cantidad de ceros en los bits más significativos es mayor o igual que un valor derivado del campo `bits` del header del bloque. A los efectos prácticos, consideraremos que, si el campo `bits` indica un valor  $d$ , la cantidad de ceros en los bits más significativos de  $h$  debe ser  $\geq d$ .

El esfuerzo necesario para ensamblar un bloque que cumpla con la dificultad de la red crece exponencialmente con la cantidad de ceros requerida. Esto se debe a que agregar un cero extra a la dificultad disminuye a la mitad la cantidad de hashes que cumplan con dicha restricción. No obstante esto, para verificar que un bloque cumple con esta propiedad, basta con computar dos veces la función de hash SHA256. De esta forma, se puede comprobar fácilmente que un minero realizó una cierta cantidad de trabajo para hallar un bloque válido.

## 4.2. Algochain: la blockchain de Algoritmos II

La blockchain simplificada con la que estaremos trabajando a lo largo del cuatrimestre es la ALGOCHAIN. Al igual que la blockchain, la ALGOCHAIN se compone de bloques que agrupan transacciones. A su vez, las transacciones constan de una secuencia de *inputs* y otra de *outputs* que siguen los mismos lineamientos esbozados más arriba. La Figura 1 muestra un esquema de alto nivel de la ALGOCHAIN.

### 4.2.1. Direcciones

Una de las diferencias más importantes con la blockchain radica en una simplificación intencional del proceso de verificación y validación de direcciones al momento de procesar las transacciones: la ALGOCHAIN no utiliza firmas digitales ni claves públicas. En su lugar, tanto los *inputs* como los *outputs* de las transacciones referencian directamente direcciones de origen y destino de los fondos, respectivamente. Esta *dirección* la interpretaremos como un hash SHA256 de una cadena de caracteres arbitraria que simbolice la dirección propiamente dicha



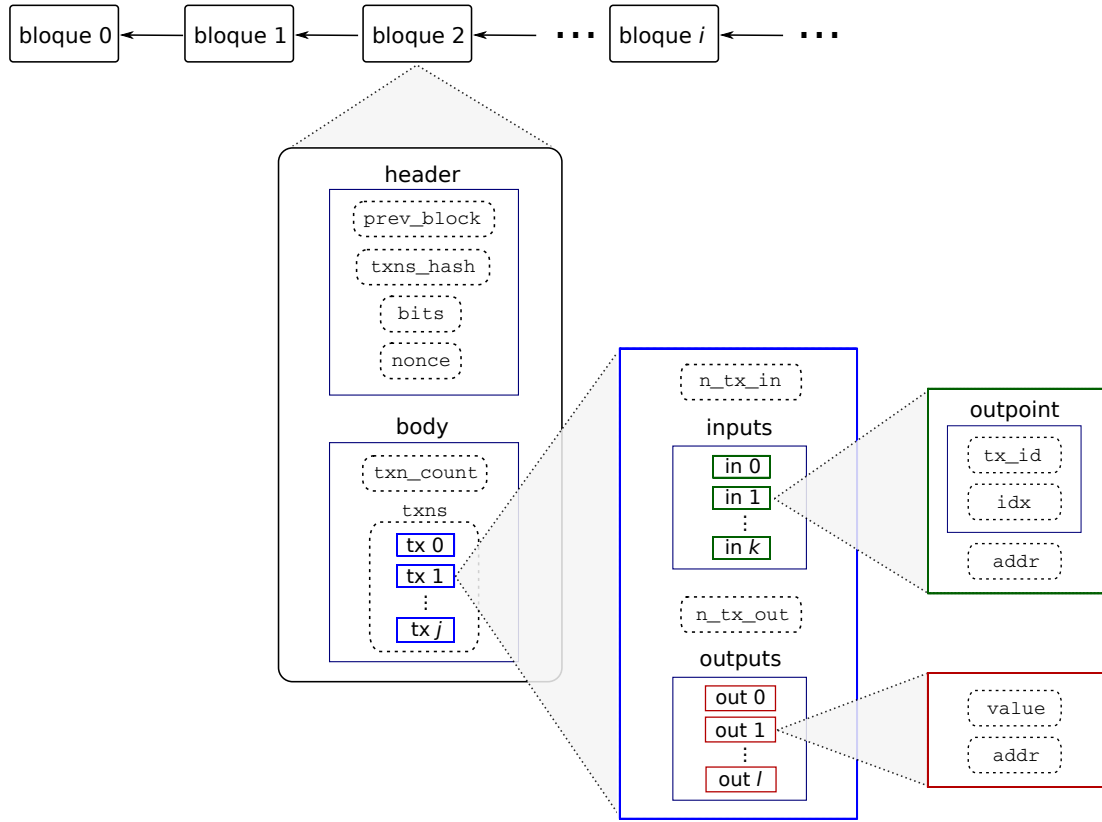


Figura 1: Esquema de alto nivel de la ALGOCHAIN

del usuario. A la hora de procesar una nueva transacción  $t$ , simplemente deberá validarse que la dirección `addr` de cada *input* de  $t$  coincida exactamente con el valor del `addr` especificado en el *output* referenciado en dicho *input*.

A modo de ejemplo, si la dirección real de un usuario de nuestra ALGOCHAIN fuese Segurola y Habana, los campos `addr` de las transacciones que involucren a dicho usuario deberían contener el valor `addr = SHA256('Segurola y Habana')`, que equivale a

485c8c85be20ebb6a9f6dd586b0f9eb6163aa0db1c6e29185b3c6cd1f7b15e9e

#### 4.2.2. Hashes de bloques y transacciones

Tal como ocurre en blockchain, y como explicamos en la Sección 4.1, en ALGOCHAIN identificamos unívocamente bloques y transacciones mediante hashes SHA256 dobles.

El campo `prev_block` del header de un bloque  $b$  indica el hash del bloque antecesor  $b'$  en la ALGOCHAIN. De este modo, `prev_block = SHA256(SHA256( $b'$ ))`. Dicho hash lo calcularemos sobre una concatenación secuencial de todos los campos de  $b'$  respetando exactamente el formato de bloque que describiremos en la Sección 4.4.

De forma análoga, el campo `tx_id` de los *inputs* de las transacciones lo calcularemos con un doble hash SHA256 sobre una concatenación de todos los campos de la transacción correspondiente.

Finalmente, el campo `txns_hash` del header de un bloque  $b$  contendrá también un doble hash SHA256 de todas las transacciones incluidas en  $b$ . En el contexto de este trabajo práctico, dicho hash lo calcularemos sobre una concatenación de todas las transacciones respetando exactamente el formato que describiremos en la Sección 4.4. En otras palabras, dadas las transacciones  $t_0, t_1 \dots, t_j$  del bloque  $b$ ,

$$\text{txns\_hash} = \text{SHA256}(\text{SHA256}(t_0 t_1 \dots t_j))$$

### 4.3. Tareas a realizar

Para apuntalar los objetivos esenciales de este trabajo práctico, esbozados en la Sección 1, deberemos escribir un programa que reciba transacciones por un *stream* de entrada y ensamble un bloque con todas ellas una vez finalizada la lectura. Dicho bloque deberá escribirse en un *stream* de salida. De este modo, al estar trabajando con único bloque, por convención dejaremos fijo el valor del campo `prev_block` en su header. Dicho campo debe instanciarse en

```
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

Naturalmente, nuestros bloques deben satisfacer los requisitos de validez delineados en la Sección 4.1.5. En particular, nos interesa exhibir la correspondiente *proof-of-work* para poder reclamar las eventuales recompensas derivadas del minado. Para ello, nuestros programas recibirán como parámetro el valor de la *dificultad*  $d$  esperada. En otras palabras, debemos garantizar que la cantidad de ceros en los bits más significativos de nuestro hash  $h$  es  $\geq d$ , siendo  $h = \text{SHA256}(\text{SHA256}(\text{header}))$ . Recordar que, en caso de no encontrar un hash  $h$  válido, es posible intentar sucesivas veces modificando el campo `nonce` del header del bloque (la Sección 4.4 describe en detalle el formato de dicho header). Este campo puede instanciarse con valores numéricos arbitrarios tantas veces como sea necesario hasta dar con un hash válido.

Para simplificar el proceso de desarrollo, la cátedra suministrará código C++ para calcular hashes SHA256.

### 4.4. Formatos de la Algochain

En esta Sección detallaremos el formato de las transacciones y bloques de la ALGOCHAIN. Tener en cuenta que es sumamente importante **respetar de manera estricta** este formato. Mostraremos algunos ejemplos concretos en la Sección 4.6.

#### 4.4.1. Transacciones

Toda transacción de la ALGOCHAIN debe satisfacer el siguiente formato:

- Empieza con una línea que contiene el campo entero `n_tx_in`, que indica la cantidad total de *inputs*.

- Luego siguen los *inputs*, uno por línea. Cada *input* consta de tres campos separados entre sí por un único espacio:
  - *tx\_id*, el hash de la transacción de donde este *input* toma fondos,
  - *idx*, un valor entero no negativo que sirve de índice sobre la secuencia de *outputs* de la transacción con hash *tx\_id*, y
  - *addr*, la dirección de origen de los fondos (que debe coincidir con la dirección del *output* referenciado).
- Luego de la secuencia de *inputs*, sigue una línea con el campo entero *n\_tx\_out*, que indica la cantidad total de *outputs* en la transacción.
- Las *n\_tx\_out* líneas siguientes contienen la secuencia de *outputs*, uno por línea. Cada *output* consta de los siguientes campos, separados por un único espacio:
  - *value*, un número de punto flotante que representa la cantidad de Algocoins a transferir en este *output*, y
  - *addr*, la dirección de destino de tales fondos.

#### 4.4.2. Bloques

Como indicamos en la Sección 4.1.4, todo bloque arranca con un header. El formato de nuestros headers es el siguiente:

- El primer campo es *prev\_block*, que contiene el hash del bloque completo que antecede al bloque actual en la ALGOCHAIN.
- Luego sigue el campo *txns\_hash*, que contiene el hash de todas las transacciones incluidas en el bloque. El cálculo de este hash debe realizarse de acuerdo a las instrucciones de la Sección 4.2.2.
- A continuación sigue el campo *bits*, un valor entero positivo que indica la dificultad con la que fue minada este bloque.
- El último campo del header es el *nonce*, un valor entero no negativo que puede contener valores arbitrarios. El objetivo de este campo es tener un espacio de prueba modificable para poder generar hashes sucesivos hasta satisfacer la dificultad del minado.

Todos estos campos deben aparecer en líneas independientes. En la línea inmediatamente posterior al *nonce* comienza la información del body del bloque:

- La primera línea contiene el campo *txn\_count*, un valor entero positivo que indica la cantidad total de transacciones incluidas en el bloque.
- A continuación siguen una por una las *txn\_count* transacciones. Todas ellas deben respetar el formato de transacción de la Sección anterior.

## 4.5. Interfaz

La interacción con nuestros programas se dará a través de la línea de comando. Las opciones a implementar en este trabajo práctico son las siguientes:

- `-d`, o `--difficulty`, que indica la dificultad esperada  $d$  del minado del bloque. En otras palabras, el hash  $h = \text{SHA256}(\text{SHA256}(\text{header}))$  debe ser tal que la cantidad de ceros en sus bits más significativos sea  $\geq d$ . Esta opción es de carácter obligatorio (i.e., el programa no puede continuar en su ausencia).
- `-i`, o `--input`, que permite controlar el stream de entrada de las transacciones. El programa deberá recibir las transacciones a partir del archivo con el nombre pasado como argumento. Si dicho argumento es `"-"`, el programa las leerá de la entrada standard, `std::cin`.
- `-o`, o `--output`, que permite direccionar la salida al archivo pasado como argumento o, de manera similar a la anterior, a la salida standard `-std::cout` si el argumento es `"-"`.

## 4.6. Ejemplos

Consideremos la siguiente transacción  $t$ :

```
1
48df0779 2 d4cc51bb
1
250.5 842f33e9
```

Por una cuestión de espacio, los hashes involucrados en estos ejemplos aparecen representados por sus últimos 8 bytes. De este modo, el hash `tx_id` del *input* de  $t$  y las direcciones *addr* referenciadas en el *input* y en el *output* son, respectivamente,

```
26429a356b1d25b7d57c0f9a6d5fed8a290cb42374185887dcd2874548df0779
f680e0021dcaf15d161604378236937225eeecae85cc6cda09ea85fad4cc51bb
0618013fa64ac6807bdea212bbdd08ffc628dd440fa725b92a8b534a842f33e9
```

Esta transacción consta de un único *input* y un único *output*. El *input* toma fondos de alguna supuesta transacción  $t'$  cuyo hash es `48df0779`. En particular, los fondos provienen del tercer *output* de  $t'$  (observar que *idx* es 2). La dirección de origen de los fondos es `d4cc51bb`. Por otra parte, el *output* de  $t$  deposita 250,5 Algos en la dirección `842f33e9`.

Supongamos ahora que contamos con un archivo de transacciones que contiene la información de  $t$ :

```
$ cat txns.txt
1
48df0779 2 d4cc51bb
1
250.5 842f33e9
```

La siguiente invocación solicita ensamblar un nuevo bloque con esta transacción:

```
$ ./tp0 -i txns.txt -o block.txt -d 3
```

Notar que el bloque debe escribirse al archivo `block.txt`. Además, la dificultad de minado sugerida es 3: esto nos dice que el hash del header de nuestro bloque debe comenzar con 3 o más bits nulos. Una posible salida podría ser la siguiente:

```
$ cat block.txt
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
155dc94b29dce95bb2f940cdd2d7e0bce66dca9370c3ed96d50a30b3d84f8c4c
3
12232
1
48df0779 2 d4cc51bb
1
250.5 842f33e9
```

Observar que el valor del *nonce* es 12232. Si bien dicho *nonce* permite encontrar un hash del header satisfactorio, esta elección por supuesto no es única. El hash del header, bajo esta elección, resulta

```
045b22553f86219b1ecb68bc34a623ecff7fe1807be806a3ccfa9f1b3df5cfc0
```

Como puede verse, el hash comienza con cuatro bits nulos.

Un ejemplo aún más simple (y curioso) consiste en invocar nuestros programas con una entrada vacía:

```
$ touch empty.txt
$ ./tp0 -i empty.txt -d 3
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
cd372fb85148700fa88095e3492d3f9f5beb43e555e5ff26d95f5a6adc36f8e6
3
59329
0
```

En primer lugar, notemos que, al no especificar un *stream* de salida, el programa dirige la escritura del bloque a la salida estándar. El bloque construido, si bien no incluye ninguna transacción, contiene información válida en su header. Notar que el campo `txns.hash` se calcula en este caso a partir del doble hash SHA256 de una cadena de caracteres vacía.

## 4.7. 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 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 12 de noviembre 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] Wikipedia, "SHA-2." <https://en.wikipedia.org/wiki/SHA-2>.