FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Algoritmos y Programación II (75.04/95.12)

TP0 - Programación C++

Grupo 3

Integrantes:

Lareo, Matias F.	97.916	mlareo@fi.uba.ar
Alvarez R., Ricardo A.	103.737	ralvarezr@fi.uba.ar
Carballeda, Ignacio L. J.	91.646	icarballeda@fi.uba.ar

Docentes:

Ing. Patricia Calvo	Titular de Cátedra	pmcalvo@fi.uba.ar
Ing. Leandro Santi	Auxiliar	lsanti@fi.uba.ar
Lic. Lucio Santi	Auxiliar	lsanti@dc.uba.ar
Ornella Pit	Auxiliar-Corrección TP	ornella.pit@hotmail.com

2do Cuatrimestre - 2020

Índice

1. Introducción

En el presente informe se expone la primera parte correspondiente al trabajo practico de Algoritmos y programación 2. Se implemento un bloque de **Algochain** (modelo basado en **Blockchain**) en **C++**. Aplicando todos los conceptos vistos en clase sobre este lenguaje y conforme al enunciado dado en la clase práctica.

2. Desarrollo

2.1. Explicación general

Para el desarrollo del presente Trabajo Práctico se modularizaron todos los tipos de datos indicados en el enunciado. En principio, se pensó un bloque en 2 grandes partes, *header* y *body*.

En cuanto al *body*, se implementaron todas las clases necesarias (Véase Figura 1) como si fueran contenedores, como lo son: *body*, *txn*, *input*, *output* y *outpoint*. Cada una de ellas, con sus métodos *get* y *set* correspondientes. En el caso de la clase *body*, se implementaron métodos de parseo, lectura y escritura de datos. También, se sobrecargó el operador << para facilitar la interacción con *streams* de salida como cout.

Para el parseo del archivo de entrada, *txns.txt*, se utilizaron *input string streams*, dada la facilidad para manipular y transformar los *strings* obtenidos del mismo archivo. También, fueron contemplados diferentes casos posibles en los que pueda venir un archivo no acorde al formato establecido por el enunciado.

El formato deberá incluir:

- 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 secuncia de *outputs* de la transacción con hash tx_id

  $addr$, la dirección de origen de los fondos (que debe coincidir con la dirección del output referenciado).

- Luego de la secuencia de *inputs*, sigue una lınea 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 numero de punto flotante que representa la cantidad de *Algocoins* a transferir en este output, y

  $addr$, la dirección de destino de tales fondos.

Con respecto al *header*, se creó la clase *header* con los atributos especificados en la Figura 1, con sus métodos *get* y *set* correspondientes, así como también la sobrecarga del operador <<.

Así mismo, se implementó la clase *block* cuyos atributos son las clases *header* y *body*, y sus métodos son los *get* y *set* correspondientes, la sobrecarga del operador <<, un método para cargar una transacción, un método para escribir en un archivo, un método de impresión y un método privado para evaluar el $proof - of - work$.

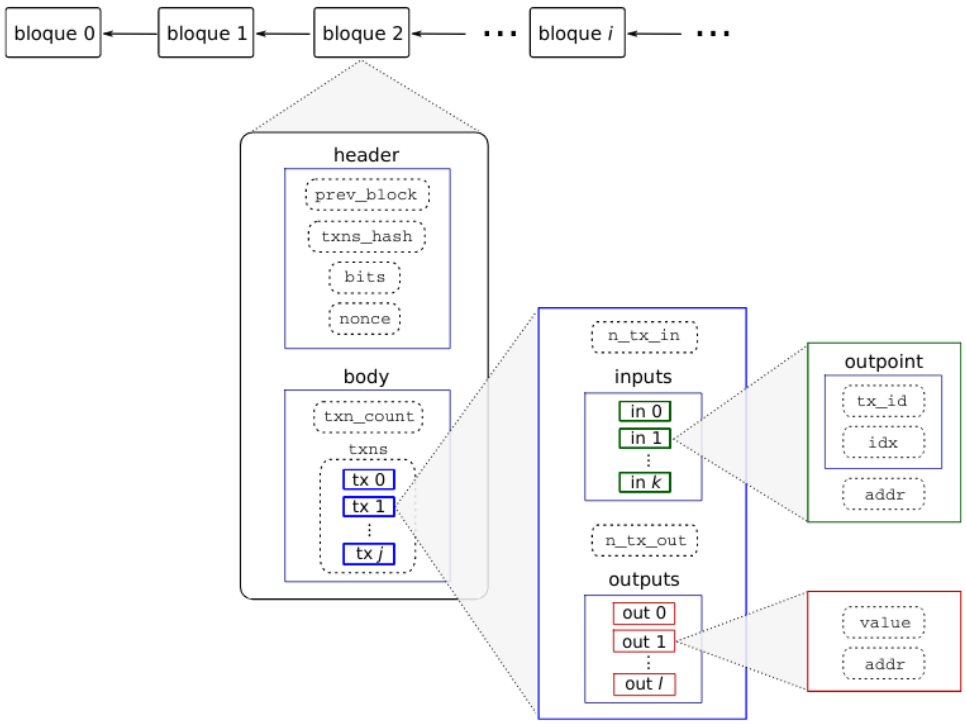

Figura 1: Esquema de Algochain

2.2. Partes relevantes del programa

2.2.1. Implementación del $proof - of - work$

El método $proof - of - work$ pertenece a la clase *block*, el mismo se encuentra en la sección 7.5. En pocas palabras la idea detrás de este método es iterar realizando la operación *sha256(sha256(header))* hasta que el resultado de la misma (operación) resulte en un *hash* con tantos ceros como el usuario haya indicado en la dificultad a la hora de comenzar con el programa.

El *header* que cumpla con lo antes mencionado será seteado como *hash* del bloque minado.

2.2.2. Sobrecargando el operador <<

Para concatenar la información del bloque, se opto por sobrecargar el operador <<. Resulta mucho mas cómodo y inteligible realizar una operación << en lugar de realizar una llamada a un método que concatene del estilo *block.cat*().

```
ostream &operator<<(ostream &os, Block b)
{
    return os << b._header
              << b._body;
}
```

Se hizo lo mismo para *body* y *header*

```
ostream &operator<<(ostream &os, Body b)
{
    return os << b.cat();
}
```

```
ostream &operator<<(ostream &os, Header h)
{
    return os << h.cat();
}
```

En donde se llama a métodos que concatenan ordenadamente y respetando el formato la información, como por ejemplo en *header*:

```
string Header::cat()
{
    string s = "";

    s += _prev_block;
    s += '\n';
    s += _txns_hash;
    s += '\n';
    s += to_string(_bits);
    s += '\n';
    s += to_string(_nonce);
    s += '\n';

    return s;
}
```

3. Compilando el programa

3.1. Compilación

Para compilar el programa, de debe utilizar el **Makefile** que se encuentra en el directorio raíz del repositorio. El compilador utilizado es **gnu++11**. Para compilar el programa simplemente se deberá ejecutar con la opción **all** como se ve en la siguiente linea.

```
make all
```

3.2. Uso del programa

3.2.1. Interfaz

- -d, o –difficulty, que indica la dificultad esperada d del minado del bloque.Esta opción es de carácter obligatorio El programa no puede continuar en su ausencia.
- -i, o –input, que permite controlar el stream de entrada de las transacciones. El programa puede recibir las transacciones a partir del archivo con el nombre pasado como argumento. Si dicho argumento es "-", el programa las leera 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 "-".

3.2.2. Ejecutando el programa

El binario generado en el paso anterior se llama **algochain.bin**. A continuación se da un ejemplo de uso.

```
./algochain.bin -i txns.txt -o block.txt -d 3
```

4. Verificando fugas de memoria

4.1. Corriendo Valgrind

Se ejecuto el programa Valgrind para detectar fugas de memoria con los parámetros indicados en la siguiente linea.

```
valgrind --leak-check=full --track-origins=yes --show-reachable=yes\
    ./algochain.bin -i txns.txt -o block.txt -d 11
```

4.2. Resultado de correr Valgrind

```
==64== Memcheck, a memory error detector
==64== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==64== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==64== Command: ./algochain.bin -i txns.txt -o block.txt -d 11
==64==
==64== error calling PR_SET_PTRACER, vgdb might block
==64==
==64== HEAP SUMMARY:
==64==     in use at exit: 0 bytes in 0 blocks
==64==   total heap usage: 5,180 allocs, 5,180 frees, 703,098 bytes allocated
==64==
==64== All heap blocks were freed -- no leaks are possible
==64==
==64== For counts of detected and suppressed errors, rerun with: -v
==64== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4.3. Conclusión

El resultado de correr el programa de análisis *Valgrind* indica que no se detectaron fugas de memoria, ya que en la salida del mismo se indica que hay tantas llamadas a pedidos de memoria como liberaciones de la misma.

5. Corridas de prueba

5.1. Pruebas realizadas

En principio, se hicieron pruebas al *body* y al *header* por separado. Luego de haber constatado el correcto funcionamiento de cada uno se procedió a probarlos nuevamente agrupados como atributos del bloque, por consiguiente, se verificó que todos los métodos siguieran funcionando de manera apropiada siendo usados ahora por los métodos propios del bloque.

Para el *body*, en primer lugar se probaron los métodos de *get* y *set*. Posteriormente, se probaron varios archivos *txns.txt* con diferentes errores para así validar el correcto funcionamiento del parseo. Estas pruebas se realizaron variando la cantidad *inputs*, *outputs*, número de transacciones, usando *hashes* inválidos y valores negativos.

En el caso del *header*, se probaron pequeños programas en los cuales se pudieran obtener y establecer los diferentes atributos del mismo, así como también la impresión de sus parámetros mediante el operador <<.

Para el bloque se probó el funcionamiento de sus métodos de *set*, *get* e *impresión*, así como también se probó que se actualizara el $txsn\_hash$ correctamente y se que pudiera obtener un *hash* válido mediante el $proof - of - work$, y a su vez, que se pudiera escribir con el formato correcto en un stream de salida.

5.2. Inputs y resultados

5.2.1. Caso 1

Comando a ejecutar.

```
./algochain.bin -i txns_test_files/txns0.txt -o block.txt -d 10

less txns_test_files/txns0.txt
```

```
3
f680e0021dcaf15d161604378236937225eeecae85cc6cda09ea85fad4cc51bb 2
 ↪  f680e0021dcaf15d161604378236937225eeecae85cc6cda09ea85fad4cc51bb
f680e0021dcaf15d161604378236937225eeecae85cc6cda09ea85fad4cc51bb 2
 ↪  f680e0021dcaf15d161604378236937225eeecae85cc6cda09ea85fad4cc51bb
f680e0021dcaf15d161604378236937225eeecae85cc6cda09ea85fad4cc51bb 2
 ↪  f680e0021dcaf15d161604378236937225eeecae85cc6cda09ea85fad4cc51bb
1
3 1a429a356b1d25b7d57c0f9a6d5fed8a290cb42374185887dcd2874548df0779
```

Resultado:

```
less block.txt
```

```
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
0a8c1c4e0e4c9e40fa1dd17e317946f7091c19068394c5bad15d5d296c29485a
10
3174
1
3
```

```
f680e0021dcaf15d16160437823693722 5eeecae85cc6cda09ea85fad4cc51bb 2
↪ f680e0021dcaf15d16160437823693722 5eeecae85cc6cda09ea85fad4cc51bb
f680e0021dcaf15d16160437823693722 5eeecae85cc6cda09ea85fad4cc51bb 2
↪ f680e0021dcaf15d16160437823693722 5eeecae85cc6cda09ea85fad4cc51bb
f680e0021dcaf15d16160437823693722 5eeecae85cc6cda09ea85fad4cc51bb 2
↪ f680e0021dcaf15d16160437823693722 5eeecae85cc6cda09ea85fad4cc51bb
1
3 1a429a356b1d25b7d57c0f9a6d5fed8a290cb42374185887dcd2874548df0779
```

Del resultado se destaca que el *header* del bloque obtenido posee una dificultad de 10, tal y como se indicó desde la línea de comandos, y que el *nonce* resultante es 3174. Con estos datos se puede calcular el *hash* del *header* para verificar así el funcionamiento del $proof - of - work$. De esta manera se obtiene que el *hash* es el siguiente:

```
0039a7eadcf6db911ce45d5343ccca9522f5ca29e97dfd622ba3a7c668303ffe
```

Del *hash* calculado (realizando el doble sha256 del *header*)se puede determinar fácilmente que los diez primeros bits son 0 ya que los tres primeros bytes son: 003, lo cual en bits es: 0000 0000 0011. Cumpliendo así efectivamente con el $proof - of - work$ requerido.

5.2.2. Caso 2: Archivo de entrada contiene solo un cero

Comando a ejecutar.

```
./algochain.bin -i txns_test_files/txns2.txt -o block.txt -d 10
```

```
less txns_test_files/txns2.txt
```

```
0
```

Resultado:

```
ERROR: Invalid number of transaction inputs
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
a0644979b83bf716e0796ce315a1d9f7883ce057f91a2ad9a330b75c6453477c
10
28
0
```

En este caso el resultado obtenido es un aviso de error ya que se espera que haya al menos una transacción válida.

5.2.3. Caso 3: Archivo de entrada está vacío

Comando a ejecutar.

```
./algochain.bin -i txns_test_files/txns1.txt -o block.txt -d 10
```

Resultado:

```
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
cd372fb85148700fa88095e3492d3f9f5beb43e555e5ff26d95f5a6adc36f8e6
10
1878
0
```

Finalmente, en este último caso el resultado obtenido es el esperado. Como se puede ver, con una dificultad de 10, al programa le tomó 1879 intentos obtener un *sha256(sha256())* de un archivo vacío.

6. Conclusiones finales

El presente trabajo nos sirvió para aplicar lo aprendido en las clases teóricas y prácticas sobre algoritmos implementados en **C++**. A su vez fue una buena introducción al mundo de la tecnología **blockchain**. La parte que nos resulto mas trabajosa fue la de entender los requerimientos del enunciado para luego traducirlos a código.

La experiencia fue positiva, esperamos que el próximo trabajo parta del actual para ir aprendiendo mas sobre **blockchain** (que nos resulto mas que interesante) y **C++**.

7. Código fuente

El código generado para resolver este trabajo práctico fue implementado en **C++11**. El mismo es de dominio publico y está disponible en el repositorio del grupo de trabajo.

7.1. main.cpp

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

#include "include/body.h"
#include "include/header.h"
#include "include/block.h"
#include "include/sha256.h"
#include "include/cmdline.h"
#include "include/errorlog.h"
#include "include/arghandler.h"

option_t options[] = {
    {1, "d", "difficulty", "1", opt_set_difficulty, OPT_MANDATORY},
    {1, "o", "output", "-", opt_set_output, OPT_DEFAULT},
    {1, "i", "input", "-", opt_set_input, OPT_DEFAULT},
    {
        0,
    }};

using namespace std;

int main(int argc, char *argv[])
{
    string _input_file;
    string _output_file;
    size_t _difficulty;
    cmdline cmdl(options);
    cmdl.parse(argc, argv);
    tie(_input_file, _output_file, _difficulty) = opt_get_values();
    Block block0(_input_file);
    block0.setDifficulty(_difficulty);
    block0.updateTxnsHash();
    block0.writeToFile(_output_file);
}
```

7.2. arghandler.h

```
#ifndef _ARGHANDLER_H_
#define _ARGHANDLER_H_

```

```
#include "cmdline.h"
#include <string>

void opt_set_difficulty(string const &arg);
void opt_set_output(string const &arg);
void opt_set_input(string const &arg);
std::tuple<string, string, size_t> opt_get_values(void);

#endif
```

7.3. arghandler.cpp

```
#include <fstream>
#include <string>
#include <sstream>
#include <tuple>

#include "../include/errorlog.h"
#include "../include/arghandler.h"

static string _input_file;
static string _output_file;
static size_t _difficulty;

std::tuple<string, string, size_t> opt_get_values(void)
{
    return std::make_tuple(_input_file, _output_file, _difficulty);
}

void opt_set_difficulty(string const &arg)
{
        char *pEnd;
        _difficulty = strtol(arg.c_str(), &pEnd, 10);

    if(_difficulty > 100)
    {
        showWarning(MSG_WARNING_DIFFICULTY_INVALID);
        _difficulty = 0;
    }
}

void opt_set_output(string const &arg)
{
        std::stringstream out(arg);
        if (out.good())
                out >> _output_file;
}

void opt_set_input(string const &arg)
{
        if (arg == "-")
```

```
        {
                cin >> _input_file;
        }
        else
        {
                std::stringstream in(arg);
                if (!in.good())
                {
                        showError(MSG_ERROR_OPENING_A_FILE, "arghandler.cpp");
                        exit(1);
                }
                in >> _input_file;
        }
}
```

7.4. block.h

```
#ifndef _BLOCK_H_
#define _BLOCK_H_
#include <fstream>
#include <string>
#include <sstream>
#include "../include/errorlog.h"
#include "../include/body.h"
#include "../include/header.h"
#include "../include/txn.h"
#include "../include/input.h"
#include "../include/output.h"

class Block
{
private:
    Header _header;
    Body _body;

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

    void setHeader(const Header &);
    void setBody(const Body &);
    void setDifficulty(const size_t &);
    Header const &getHeader() const;
    Body const &getBody() const;
    void updateTxnsHash();

    void loadTxn(const string);
    void writeToFile(const string);
```

```
35      void print();
36
37      friend std::ostream &operator<<(std::ostream &, Block);
38
39  private:
40      void proofOfWork();
41  };
42
43  #endif
```

7.5. block.cpp

```
1   #include "../include/block.h"
2   #include "../include/body.h"
3   #include "../include/sha256.h"
4   #include "../include/errorlog.h"
5   #include <iostream>
6   #include <string>
7   #include <bitset>
8
9   using namespace std;
10
11  Block::Block()
12  {
13  }
14
15  Block::Block(const Header &h, const Body &b)
16  {
17      _header = h;
18      _body = b;
19  }
20
21  Block::Block(const Block &b)
22  {
23      _header = b._header;
24      _body = b._body;
25  }
26
27  Block::~Block()
28  {
29  }
30
31  Block::Block(string filepath)
32  {
33      loadTxn(filepath);
34  }
35
36  void Block::loadTxn(const string filepath)
37  {
38      ifstream txns_file(filepath);
39      Txn newTxn;
```

```
40      if (!txns_file.good())
41      {
42          showError(MSG_ERROR_INVALID_FILEPATH);
43          return;
44      }
45
46      while (!txns_file.eof())
47      {
48          string line;
49          int n_tx_in = 0;
50          int n_tx_out = 0;
51          getline(txns_file, line);
52          istringstream inputStream(line);
53          if (line.length() == 0)
54              return;
55          inputStream >> n_tx_in;
56          if (n_tx_in < 1)
57          {
58              showError(MSG_ERROR_INVALID_N_TX_IN);
59              return;
60          }
61
62          for (int i = 0; i < n_tx_in; ++i)
63          {
64              getline(txns_file, line);
65              istringstream input_data(line);
66              string tx_id;
67              int id_x;
68              string addr;
69              input_data >> tx_id;
70              if (tx_id.length() != 64 || input_data.fail())
71              {
72                  showError(MSG_ERROR_INVALID_TX_ID, "Input number " + (i + 1));
73                  return;
74              }
75              input_data >> id_x;
76              if (id_x < 0 || input_data.fail())
77              {
78                  showError(MSG_ERROR_INVALID_IDX, "Input number " + (i + 1));
79                  return;
80              }
81              input_data >> addr;
82              if (addr.length() != 64 || input_data.fail())
83              {
84                  showError(MSG_ERROR_INVALID_ADDR, "Input number " + (i + 1));
85                  return;
86              }
87              Input newInput(addr, tx_id, id_x);
88              newTxn.addInput(newInput);
89          }
90
```

```
91          getline(txns_file, line);
92          istringstream outputStream(line);
93          outputStream >> n_tx_out;
94          if (n_tx_out > n_tx_in || outputStream.fail())
95          {
96              showError(MSG_ERROR_INVALID_N_TX_OUT);
97              return;
98          }
99          for (int i = 0; i < n_tx_out; ++i)
100         {
101
102             getline(txns_file, line);
103             istringstream output_data(line);
104             float value;
105             string addr;
106             output_data >> value;
107             if (value < 0 || output_data.fail())
108             {
109                 showError(MSG_ERROR_INVALID_OUTPUT_VALUE, "Output number " + (i + 1));
110                 return;
111             }
112             output_data >> addr;
113             if (addr.length() != 64 || output_data.fail())
114             {
115                 showError(MSG_ERROR_INVALID_OUTPUT_ADDR, "Output number " + (i + 1));
116                 return;
117             }
118             Output newOutput(addr, value);
119             newTxn.addOutput(newOutput);
120         }
121
122         _body.addTxn(newTxn);
123     }
124     txns_file.close();
125 }
126
127 void Block::print()
128 {
129     cout << _header.cat() << endl;
130     cout << _body.cat() << endl;
131 }
132
133 void Block::writeToFile(string filepath)
134 {
135     if(filepath[0]!='-')
136     {
137         ofstream block_file(filepath);
138         if (!block_file.good())
139         {
140             showError(MSG_ERROR_INVALID_FILEPATH);
141             return;
```

```
142         }
143         block_file << _header.cat()
144                 << _body.cat();
145 
146         if (!block_file.good())
147         {
148             showError(MSG_ERROR_WRITING_TO_FILE);
149             return;
150         }
151         block_file.close();
152     }
153     else
154     {
155         cout << _header.cat()
156              << _body.cat();
157     }
158 }
159 
160 void Block::setHeader(const Header &h)
161 {
162     _header = h;
163 }
164 
165 void Block::setBody(const Body &b)
166 {
167     _body = b;
168 }
169 
170 void Block::setDifficulty(const size_t &d)
171 {
172     _header.setBits(d);
173 }
174 
175 Header const &Block::getHeader() const
176 {
177     return _header;
178 }
179 
180 Body const &Block::getBody() const
181 {
182     return _body;
183 }
184 
185 void Block::updateTxnsHash()
186 {
187 
188     string s = _body.cat();
189     _header.setTxnsHash(sha256(sha256(s)));
190 
191     proofOfWork();
192 }
```

```
193
194 void Block::proofOfWork()
195 {
196
197     size_t d = _header.getBits(); //Obtengo la dificultad
198
199     if (d == 0) //Si es 0, es indistinto el hash.
200         return;
201
202     size_t count = 0;
203     bitset<4> c;
204     bool _minning = true;
205
206     while (_minning)
207     {
208
209         string s = _header.cat();
210
211         string h = sha256(sha256(s));
212
213         for (size_t j = 0; j < 32; j++)
214         { //Reviso los 32 bytes del hash.
215
216             if ((h[j] - 48) > 48)   //Si esta entre a-f
217                 c = (h[j] - 87);
218             else                    //Si esta entre 0-9
219                 c = (h[j] - 48);
220
221             size_t i = 0;
222
223             /*
224                 Pruebo que cada bit sea 0. El bit c[0] = LSb y c[3] = MSb
225             */
226             for ( ;i < 4; i++)
227             {
228                 if (c[3 - i] != 0)
229                     break;
230                 count++;
231             }
232
233             if (i == 0)
234             {
235                 _header.incrementNonce();
236                 count = 0;
237                 break;
238             }
239
240             if ((count < d) && ((count % 4) != 0))
241             {
242                 _header.incrementNonce();
243                 count = 0;
```

```
244                 break;
245             }
246 
247             if (count >= d)
248             { //Si el count es mayor o igual a la dificultad
249                 _minning = false;
250                 break; //Salgo del ciclo.
251             }
252         }
253     }
254 }
255 
256 ostream &operator<<(ostream &os, Block b)
257 {
258     return os << b._header
259               << b._body;
260 }
```

7.6. body.h

```
1  #ifndef _BODY_H_
2  #define _BODY_H_
3  
4  #include <cstdlib>
5  
6  #include "../include/txn.h"
7  #include "../include/sha256.h"
8  #include "../include/algovector.h"
9  
10 using namespace std;
11 
12 class Body
13 {
14 public:
15     Body();
16     ~Body();
17     size_t getTxnCount(void);
18     void setTxnCount(size_t txn_count);
19     void addTxn(Txn);
20     string cat();
21     friend ostream &operator<<(ostream &, Body);
22 
23 private:
24     size_t _txn_count;
25     algoVector<Txn> _txns;
26 };
27 
28 #endif /** _BODY_H_ */
```

7.7. body.cpp

```
#include "../include/body.h"
#include "../include/txn.h"
#include "../include/algovector.h"

#include <iostream>

using namespace std;

Body::~Body(void)
{
}

Body::Body(void)
{
    _txn_count = 0;
}

void Body::setTxnCount(size_t txn_count)
{
    _txn_count = txn_count;
}

size_t Body::getTxnCount(void)
{
    return _txn_count;
}

void Body::addTxn(Txn newTxn)
{
    _txns.push_back(newTxn);
    _txn_count = _txn_count + 1;
}

string Body::cat()
{
    string concatTxns = "";
    if (_txn_count == 0)
    {
        return concatTxns.append("0\n");
    }
    concatTxns.append(to_string(_txn_count));
    concatTxns.append("\n");
    for (size_t i = 0; i < _txn_count; i++)
    {
        concatTxns.append(to_string(_txns[i].getNTxIn()));
        concatTxns.append("\n");
        for (size_t j = 0; j < _txns[i].getNTxIn(); ++j)
        {
            concatTxns.append((_txns[i].getInputs())[j].getAddr());
```

```
            concatTxns.append(" ");
            concatTxns.append(to_string((_txns[i].getInputs())[j].getOutpointIdx()));
            concatTxns.append(" ");
            concatTxns.append((_txns[i].getInputs())[j].getOutpointTxId());
            concatTxns.append("\n");
        }
        concatTxns.append(to_string(_txns[i].getNTxOut()));
        concatTxns.append("\n");

        for (size_t j = 0; j < _txns[i].getNTxOut(); ++j)
        {
            concatTxns.append(to_string((_txns[i].getOutputs())[j].getValue()));
            concatTxns.append(" ");
            concatTxns.append((_txns[i].getOutputs())[j].getAddr());
            if (j != _txns[i].getNTxOut() - 1)
                concatTxns.append("\n");
        }
    }

    return concatTxns;
}

ostream &operator<<(ostream &os, Body b)
{
    return os << b.cat();
}
```

7.8. header.h

```
#ifndef _HEADER_H_
#define _HEADER_H_

#include <cstdlib>
#include <string>

using namespace std;

class Header
{
    private:
        string _prev_block = "fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
        string _txns_hash;
        size_t _bits;
        size_t _nonce = 0;
    public:
        Header();                                   //Constructor simple.
        Header(size_t &);                           //Constructor con _bits.
        Header(const Header &);                     //Constructor copia.
        Header const &operator=(Header const &);    //Operador '='.
        ~Header();                                  //Destructor.

```

```
23          string const &getPrevBlock() const;          //Devuelve _prev_block.
24          size_t const &getBits() const;               //Devuelve _bits.
25          size_t const &getNonce() const;              //Devuelve _nonce.
26          string const &getTxnsHash() const;           //Devuelve _txns_hash.
27          void setBits(size_t const &);                //Establece _bits.
28          void setTxnsHash(string const &);            //Establece _txns_hash.
29          void incrementNonce();                       //Aumenta el valor del _nonce.
30          string cat();                                //Concatena el contenido del Header
31
32          friend ostream &operator<<(ostream &, Header); //Operador '<<'.
33
34  };
35
36  #endif /** _HEADER_H_ */
```

7.9. header.cpp

```
1   #include "../include/header.h"
2
3   #include <iostream>
4
5   using namespace std;
6
7   Header::Header()
8   {
9       _bits = 0;
10      _txns_hash = "";
11  }
12
13  Header::Header(size_t &b)
14  {
15      _bits = b;
16      _txns_hash = "";
17  }
18
19  Header::Header(const Header &h)
20  {
21      _bits = h._bits;
22      _txns_hash = h._txns_hash;
23      _nonce = h._nonce;
24      _prev_block = h._prev_block;
25  }
26
27  Header const &Header::operator=(Header const &r)
28  {
29
30      _bits = r._bits;
31      _txns_hash = r._txns_hash;
32      _prev_block = r._prev_block;
33      _nonce = r._nonce;
34      return *this;
```

```
35  }
36
37  Header::~Header()
38  {
39  }
40
41  string const &Header::getPrevBlock() const
42  {
43      return _prev_block;
44  }
45
46  size_t const &Header::getBits() const
47  {
48      return _bits;
49  }
50
51  size_t const &Header::getNonce() const
52  {
53      return _nonce;
54  }
55
56  string const &Header::getTxnsHash() const
57  {
58      return _txns_hash;
59  }
60
61  void Header::setBits(size_t const &b)
62  {
63      _bits = b;
64  }
65
66  void Header::setTxnsHash(string const &s)
67  {
68      _txns_hash = s;
69  }
70
71  void Header::incrementNonce()
72  {
73      _nonce++;
74  }
75
76  string Header::cat()
77  {
78      string s = "";
79
80      s += _prev_block;
81      s += '\n';
82      s += _txns_hash;
83      s += '\n';
84      s += to_string(_bits);
85      s += '\n';
```

```
86      s += to_string(_nonce);
87      s += '\n';
88
89      return s;
90  }
91
92  ostream &operator<<(ostream &os, Header h)
93  {
94      return os << h.cat();
95  }
```

7.10. input.h

```
1   #ifndef _INPUTS_H_
2   #define _INPUTS_H_
3
4   #include <cstdlib>
5   #include <string>
6
7   #include "../include/outpoint.h"
8   #include "../include/algovector.h"
9
10  using namespace std;
11
12  class Input
13  {
14  public:
15      Input();
16      Input(string, string, size_t);
17      ~Input();
18      string getAddr();
19      string getOutpointTxId();
20      size_t getOutpointIdx();
21
22  private:
23      string addr;
24      Outpoint _outpoint;
25  };
26
27  #endif /** _INPUTS_H_ */
```

7.11. input.cpp

```
1   #include "../include/input.h"
2
3   #include <iostream>
4
5   using namespace std;
6
7   Input::Input()
8   {
```

```
}

Input::Input(string a, string tx_id, size_t idx) : _outpoint(tx_id, idx)
{
    addr = a;
}

Input::~Input()
{
}

string Input::getAddr()
{
    return addr;
}

string Input::getOutpointTxId()
{
    return _outpoint.getTxId();
}

size_t Input::getOutpointIdx()
{
    return _outpoint.getIdx();
}
```

7.12. outpoint.h

```
#ifndef _OUTPOINT_H_
#define _OUTPOINT_H_

#include <cstdlib>
#include <string>
using namespace std;

class Outpoint
{
public:
    Outpoint();
    Outpoint(string, size_t);
    ~Outpoint();
    string getTxId();
    size_t getIdx();

private:
    string tx_id;
    size_t idx;
};

#endif /** _OUTPOINT_H_ */
```

7.13. outpoint.cpp

```
#include "../include/outpoint.h"

#include <iostream>

using namespace std;

Outpoint::Outpoint()
{
}

Outpoint::Outpoint(string t, size_t id)
{
    tx_id = t;
    idx = id;
};

Outpoint::~Outpoint()
{
}

string Outpoint::getTxId()
{
    return tx_id;
};

size_t Outpoint::getIdx()
{
    return idx;
};
```

7.14. txn.h

```
#ifndef _TXN_H_
#define _TXN_H_

#include <cstdlib>

#include "../include/output.h"
#include "../include/input.h"
#include "../include/outpoint.h"
#include "../include/algovector.h"

using namespace std;

class Txn
{
public:
    Txn(); //default
    Txn(size_t, algoVector<Input>, size_t, algoVector<Output>);
```

```
18      ~Txn();
19      size_t getNTxIn();
20      algoVector<Input> getInputs();
21      size_t getNTxOut();
22      algoVector<Output> getOutputs();
23      void addInput(Input);
24      void addOutput(Output);
25
26  private:
27      size_t _n_tx_in;
28      algoVector<Input> _inputs;
29      size_t _n_tx_out;
30      algoVector<Output> _outputs;
31  };
32
33  #endif /** _TXN_H_ */
```

7.15. txn.cpp

```
1   #include "../include/txn.h"
2   #include "../include/output.h"
3   #include "../include/input.h"
4
5   #include <iostream>
6
7   using namespace std;
8
9   Txn::Txn()
10  {
11      _n_tx_in = 0;
12      _n_tx_out = 0;
13  }
14
15  Txn::Txn(size_t nti, algoVector<Input> in, size_t nto, algoVector<Output> out)
16  {
17      _n_tx_in = nti;
18      _inputs = in;
19      _n_tx_out = nto;
20      _outputs = out;
21  }
22
23  Txn::~Txn()
24  {
25  }
26
27  size_t Txn::getNTxIn()
28  {
29      return _n_tx_in;
30  }
31
32  algoVector<Input> Txn::getInputs()
```

```
{
    return _inputs;
}

size_t Txn::getNTxOut()
{
    return _n_tx_out;
}

algoVector<Output> Txn::getOutputs()
{
    return _outputs;
}

void Txn::addInput(Input newInput)
{
    _inputs.push_back(newInput);
    _n_tx_in = _n_tx_in + 1;
}

void Txn::addOutput(Output newOutput)
{
    _outputs.push_back(newOutput);
    _n_tx_out = _n_tx_out + 1;
}
```

7.16. errorlog.h

```
#ifndef _ERRORLOG_H_
#define _ERRORLOG_H_

#include <iostream>
#include <string>

using namespace std;

// Error msgs
#define MSG_ERROR_INVALID_FILEPATH "Invalid filepath"
#define MSG_ERROR_WRITING_TO_FILE "Couldn't write to file"
#define MSG_ERROR_INVALID_N_TX_IN "Invalid number of transaction inputs"
#define MSG_ERROR_INVALID_TX_ID "Invalid transaction ID"
#define MSG_ERROR_INVALID_IDX "Invalid IDx"
#define MSG_ERROR_INVALID_ADDR "Invalid address"
#define MSG_ERROR_INVALID_N_TX_OUT "Invalid number of transaction outputs"
#define MSG_ERROR_INVALID_OUTPUT_VALUE "Invalid output value"
#define MSG_ERROR_INVALID_OUTPUT_ADDR "Invalid output address"
#define MSG_ERROR_OPENING_A_FILE "Couldn't open the file"

// Warning msgs
#define MSG_WARNING_TXN_FILE_IS_EMPTY "Transaction file is empty"
#define MSG_WARNING_DIFFICULTY_INVALID "Difficulty is invalid, using default: d=0"
```

```
24
25 void showError(string, string = "");
26 void showWarning(string);
27
28 #endif
```

7.17. errorlog.cpp

```
1  #include "../include/errorlog.h"
2
3  using namespace std;
4
5  void showError(string err, string det)
6  {
7      if (det != "")
8          cerr << "ERROR: " << err << " (" << det << ")" << endl;
9      else
10         cerr << "ERROR: " << err << endl;
11 }
12
13 void showWarning(string err)
14 {
15     cerr << "WARNING: " << err << endl;
16 }
```

7.18. algovector.h

```
1  // Self implementation of
2  // the algoVector Class in C++
3
4  #ifndef _ALGOVECTOR_H_
5  #define _ALGOVECTOR_H_
6
7  template <class T>
8  class  algoVector
9  {
10 public:
11
12     typedef T * iterator;
13
14     algoVector();
15     algoVector(unsigned int size);
16     algoVector(unsigned int size, const T & initial);
17     algoVector(const algoVector<T> & v);
18     ~algoVector();
19
20     unsigned int capacity() const;
21     unsigned int size() const;
22     bool empty() const;
23     iterator begin();
24     iterator end();
```

```
25      T & front();
26      T & back();
27      void push_back(const T & value);
28      void pop_back();
29
30      void reserve(unsigned int capacity);
31      void resize(unsigned int size);
32
33      T & operator[](unsigned int index);
34      algoVector<T> & operator=(const algoVector<T> &);
35      void clear();
36  private:
37      unsigned int my_size;
38      unsigned int my_capacity;
39      T * buffer;
40  };
41
42  template<class T>
43  algoVector<T>::algoVector()
44  {
45      my_capacity = 0;
46      my_size = 0;
47      buffer = 0;
48  }
49
50  template<class T>
51  algoVector<T>::algoVector(const algoVector<T> & v)
52  {
53      my_size = v.my_size;
54      my_capacity = v.my_capacity;
55      buffer = new T[my_size];
56      for (unsigned int i = 0; i < my_size; i++)
57          buffer[i] = v.buffer[i];
58  }
59
60  template<class T>
61  algoVector<T>::algoVector(unsigned int size)
62  {
63      my_capacity = size;
64      my_size = size;
65      buffer = new T[size];
66  }
67
68  template<class T>
69  algoVector<T>::algoVector(unsigned int size, const T & initial)
70  {
71      my_size = size;
72      my_capacity = size;
73      buffer = new T [size];
74      for (unsigned int i = 0; i < size; i++)
75          buffer[i] = initial;
```

```
76      //T();
77  }
78
79  template<class T>
80  algoVector<T> & algoVector<T>::operator = (const algoVector<T> & v)
81  {
82      delete[ ] buffer;
83      my_size = v.my_size;
84      my_capacity = v.my_capacity;
85      buffer = new T [my_size];
86      for (unsigned int i = 0; i < my_size; i++)
87          buffer[i] = v.buffer[i];
88      return *this;
89  }
90
91  template<class T>
92  typename algoVector<T>::iterator algoVector<T>::begin()
93  {
94      return buffer;
95  }
96
97  template<class T>
98  typename algoVector<T>::iterator algoVector<T>::end()
99  {
100     return buffer + size();
101 }
102
103 template<class T>
104 T& algoVector<T>::front()
105 {
106     return buffer[0];
107 }
108
109 template<class T>
110 T& algoVector<T>::back()
111 {
112     return buffer[my_size - 1];
113 }
114
115 template<class T>
116 void algoVector<T>::push_back(const T & v)
117 {
118     if (my_size >= my_capacity)
119         reserve(my_capacity +5);
120     buffer [my_size++] = v;
121 }
122
123 template<class T>
124 void algoVector<T>::pop_back()
125 {
126     my_size--;
```

```
127 }
128
129 template<class T>
130 void algoVector<T>::reserve(unsigned int capacity)
131 {
132     if(buffer == 0)
133     {
134         my_size = 0;
135         my_capacity = 0;
136     }
137     T * Newbuffer = new T [capacity];
138     //assert(Newbuffer);
139     unsigned int l_Size = capacity < my_size ? capacity : my_size;
140     //copy (buffer, buffer + l_Size, Newbuffer);
141
142     for (unsigned int i = 0; i < l_Size; i++)
143         Newbuffer[i] = buffer[i];
144
145     my_capacity = capacity;
146     delete[] buffer;
147     buffer = Newbuffer;
148 }
149
150 template<class T>
151 unsigned int algoVector<T>::size()const//
152 {
153     return my_size;
154 }
155
156 template<class T>
157 void algoVector<T>::resize(unsigned int size)
158 {
159     reserve(size);
160     my_size = size;
161 }
162
163 template<class T>
164 T& algoVector<T>::operator[](unsigned int index)
165 {
166     return buffer[index];
167 }
168
169 template<class T>
170 unsigned int algoVector<T>::capacity()const
171 {
172     return my_capacity;
173 }
174
175 template<class T>
176 algoVector<T>::~algoVector()
177 {
```

```
178     delete[ ] buffer;
179 }
180 template <class T>
181 void algoVector<T>::clear()
182 {
183     my_capacity = 0;
184     my_size = 0;
185     buffer = 0;
186 }
187
188 #endif //_ALGOVECTOR_H_
```

7.19. Makefile

```
1  IDIR = ../include
2  CCFLAGS = -Wall -I$(IDIR) -std=gnu++11
3  CC = g++
4
5  all: algochain
6
7  algochain: main.o sha256.o cmdline.o block.o body.o header.o txn.o \
8          input.o output.o outpoint.o errorlog.o arghandler.o
9
10         $(CC) $(CCFLAGS) -o algochain.bin build/main.o build/sha256.o \
11         build/cmdline.o build/txn.o build/block.o build/body.o build/header.o \
12         build/input.o build/output.o build/outpoint.o build/errorlog.o \
13         build/arghandler.o
14
15 main.o: main.cpp include/sha256.h include/cmdline.h
16         $(CC) $(CCFLAGS) -c main.cpp -o build/main.o
17
18 sha256.o: src/sha256.cpp include/sha256.h
19         $(CC) $(CCFLAGS) -c src/sha256.cpp -o build/sha256.o
20
21 cmdline.o: src/cmdline.cpp include/cmdline.h
22         $(CC) $(CCFLAGS) -c src/cmdline.cpp -o build/cmdline.o
23
24 block.o:  src/block.cpp include/block.h
25         $(CC) $(CCFLAGS) -c src/block.cpp -o build/block.o
26
27 header.o:  src/header.cpp include/header.h
28         $(CC) $(CCFLAGS) -c src/header.cpp -o build/header.o
29
30 body.o:  src/body.cpp include/body.h
31         $(CC) $(CCFLAGS) -c src/body.cpp -o build/body.o
32
33 txn.o:  src/txn.cpp include/txn.h
34         $(CC) $(CCFLAGS) -c src/txn.cpp -o build/txn.o
35
36 input.o:  src/input.cpp include/input.h
37         $(CC) $(CCFLAGS) -c src/input.cpp -o build/input.o
```

```
38
39 output.o:  src/output.cpp include/output.h
40         $(CC) $(CCFLAGS) -c src/output.cpp -o build/output.o
41
42 outpoint.o:  src/outpoint.cpp include/outpoint.h
43         $(CC) $(CCFLAGS) -c src/outpoint.cpp -o build/outpoint.o
44
45 errorlog.o:  src/errorlog.cpp include/errorlog.h
46         $(CC) $(CCFLAGS) -c src/errorlog.cpp -o build/errorlog.o
47
48 arghandler.o:  src/arghandler.cpp include/arghandler.h
49         $(CC) $(CCFLAGS) -c src/arghandler.cpp -o build/arghandler.o
50
51 clean:
52         $(RM) build/*.o algochain
```