



UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
95.12 - Algoritmos y Programación II

Trabajo práctico 0 Programación C++

Alumnos:

Briñoli, Ariel Eduardo 78863 abrinoli@fi.uba.ar
Faganelli, Federico 96669 fedefaganelli.coc@gmail.com
Merlo, Ignacio 97905 imerlo@fi.uba.ar

2° cuatrimestre 2020

N° Entrega	Fecha
1	12/11/2020
2	26/11/2020

1. Objetivo

El objetivo del trabajo práctico es implementar un programa que permita leer y procesar transacciones y ensamblar un bloque, perteneciente a la tecnología blockchain. El usuario también debe poder indicar el flujo de salida y de entrada de los resultados.

2. Funcionamiento del programa

Se va a crear un programa que reciba transacciones por un stream de entrada y ensamble un bloque con todas ellas una vez finalizada la lectura. El usuario debe introducir por línea de comandos el archivo de entrada y de salida. Los flags que se utilizan por línea de comandos pueden ser introducidos en cualquier orden y son los siguientes:

- `-input (-i)`: Permite introducir el archivo de entrada. De no estar presente este flag o introducirse el carácter `'-'` como parámetro, se asumirá la entrada como el flujo cin.
- `-output (-o)`: Permite introducir el archivo de salida. De no estar presente este flag o introducirse el carácter `'-'` como parámetro, se asumirá la salida como el flujo cout.
- `-difficulty (-d)`: Es de carácter obligatorio y sirve para definir cual es la dificultad del minado del bloque.

3. Implementaciones

3.1. Block.cpp

Se empleó una clase para el Blockchain el cual consiste en 2 partes:

- Body: Aquí estarán las transacciones a realizar, que como pueden variar en cantidad se las implementarán como parte de una lista.
- Header: Contiene información del bloque

3.2. Lista.h

La lista implementada consiste en una lista doble enlazada, que puede ser usada como tal, o como cola. La clase se encuentra implementada como template, por lo que puede ser creada conteniendo distintos tipos de datos. A su vez, la clase contiene a dos subclases: nodo e iterador. La primera es la estructura unitaria de la lista, sus atributos son un puntero al nodo anterior y siguiente y el dato en sí. No incluye mas métodos que constructores y destructor. La clase iterador, es utilizada para avanzar de manera sencilla a través de la lista. Su único atributo es un puntero al nodo en el que actualmente se encuentra el iterador posicionado. Entre sus métodos se incluyen constructores, métodos para avanzar y retroceder, comparadores, indicadores de fin de lista, y un método para obtener el dato del nodo actual. Los atributos de la clase lista son simplemente dos punteros al primer y último nodo de la lista y una variable `size_t` que contiene el tamaño actual de la lista.

4. Compilación

La compilación se realiza mediante un archivo Makefile, el cual se entrega junto al código fuente. El compilador utilizado es `g++` con los siguientes flags:

- `-std=c++11` para especificar que se utilice el standard 11 debido a que utilizamos funciones que lo requieren, tales como `std::stoi` y `std::to_string`.
- `-O0` para no optimizar, lo cual hace más rápida la compilación.
- `-g3` para agregar información de debugging a los archivos binarios obtenidos.
- `-Wall` para activar lo warnings.

Para compilar, se ejecuta el comando `make` en el mismo directorio de los archivos fuente.

5. Testing

5.1. Caso 1 - Archivo de entrada con salida standard

Se invoca al programa con un archivo de entrada:

```
tp0.exe -d 3 -i input.txt
```

De esta manera, se recibe el archivo de transacciones y se muestra el bloque generado en el stream de salida standard.

```

fedede@fedede-VirtualBox: ~/Algo2/TrabajoPractico0/Tp0/src
fedede@fedede-VirtualBox:~/Algo2/TrabajoPractico0/Tp0/src$ cat input.txt
1
48df0779 2 d4cc51bb
1
250.5 842f33e9
fedede@fedede-VirtualBox:~/Algo2/TrabajoPractico0/Tp0/src$ ./tp0 -d 3 -i input.txt
Header Hash:
0ce6196a94680c6ea012355e36b923d7384a532723b457312ce559299a591018

ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
9e7d5aac9d0503953f3f24b583f4bde22c80e3e6eea58551879d354a6457e5f1
3
1345567917
1
1
48df0779 2 d4cc51bb
1
250.5 842f33e9
fedede@fedede-VirtualBox:~/Algo2/TrabajoPractico0/Tp0/src$ █

```

Figura 1: Entrada con archivo .txt - Salida por el stream standard.

5.2. Caso 2 - Archivo de entrada con archivo de salida

En este otro caso, en vez de imprimir en el stream de salida standard, escribimos a un archivo .txt La invocación es la siguiente:

```
tp0.exe -d 3 -i input.txt -o outputFile.txt
```

```

fedede@fedede-VirtualBox: ~/Algo2/TrabajoPractico0/Tp0/src
fedede@fedede-VirtualBox:~/Algo2/TrabajoPractico0/Tp0/src$ ./tp0 -d 3 -i input.txt -
o outputFile.txt
Header Hash:
03556395fa0c35b0e6989201bf5a5102de87bea16cf59c14b13a8cb53edd15a5

fedede@fedede-VirtualBox:~/Algo2/TrabajoPractico0/Tp0/src$ cat outputFile.txt
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
9e7d5aac9d0503953f3f24b583f4bde22c80e3e6eea58551879d354a6457e5f1
3
1055024105
1
1
48df0779 2 d4cc51bb
1
250.5 842f33e9
fedede@fedede-VirtualBox:~/Algo2/TrabajoPractico0/Tp0/src$ █

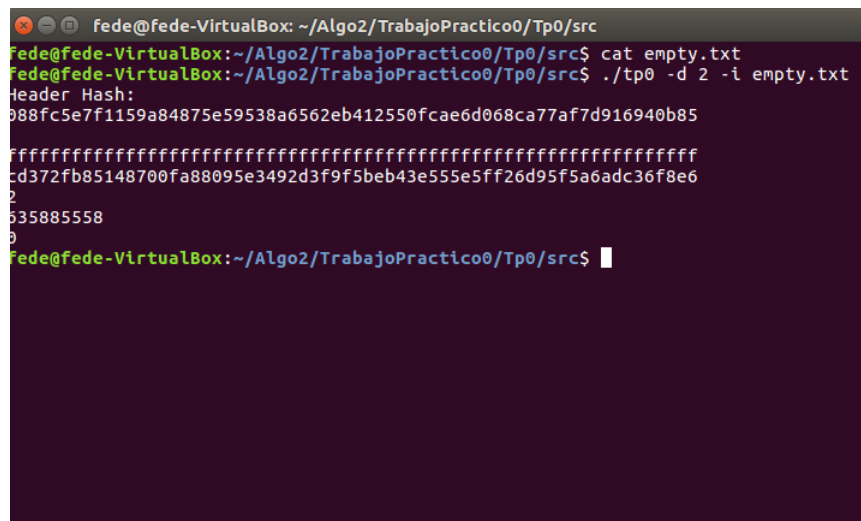
```

Figura 2: Entrada con archivo .txt - Salida por archivo .txt.

5.3. Caso 3 - Archivo de entrada vacío con salida standard

En este último caso, se introduce un archivo sin contenido con lo cual se genera el hash de las transacciones con el doble SHA256 de un string vacío.

```
tp0.exe -d 2 -i empty.txt
```



```
fedede@fedede-VirtualBox: ~/Algo2/TrabajoPractico0/Tp0/src
fedede@fedede-VirtualBox:~/Algo2/TrabajoPractico0/Tp0/src$ cat empty.txt
fedede@fedede-VirtualBox:~/Algo2/TrabajoPractico0/Tp0/src$ ./tp0 -d 2 -i empty.txt
Header Hash:
988fc5e7f1159a84875e59538a6562eb412550fcae6d068ca77af7d916940b85

ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
cd372fb85148700fa88095e3492d3f9f5beb43e555e5ff26d95f5a6adc36f8e6
2
535885558
9
fedede@fedede-VirtualBox:~/Algo2/TrabajoPractico0/Tp0/src$
```

Figura 3: Entrada con archivo .txt vacío - Salida por stream standard.