Facultad de Ingenieria,
Universidad de Buenos Aires

Carlos Germán Carreño Romano

12 de junio de 2014

Índice

1. Introducción **3**

2. Diseño e Implementación **3**
2.1. NetworkElement Class 3
2.2. Validación de flujos de entrada y salida 4
2.3. Proceso del archivo de entrada 4
2.3.1. Nombre de Red 5
2.3.2. Vector de Elementos de Red 5
2.3.3. Conexionado 5
2.4. Detección de loops 6
2.5. Archivo de salida 6

3. Ejecuciones **7**
3.1. Compilación 7
3.2. Comprobación de uso de memoria con Valgrind 9

4. Códigos **10**

1. Introducción

En este trabajo práctico se pretende incorporar conceptos e implementar algunas soluciones relacionadas con la temática de las Estructuras de Datos.

2. Diseño e Implementación

Para resolver el trabajo práctico se consideró implementar un programa que responda a la siguiente secuencia de tareas:

1. Diseñar una clase NetworkElement con los atributos necesarios para la topología;
2. Validar archivos de entrada y salida;
3. Leer las líneas del archivo de entrada;
4. Identificar el Nombre de Red (key1=NetworkName);
5. Generar un vector de Elementos de Red con los datos leídos (key1=NetworkElement);
6. Conectar los elementos del vector según los datos leídos (key1=Connection);
7. Validar que no haya loops;
8. Imprimir en el archivo de salida la estructura de la topología generada;

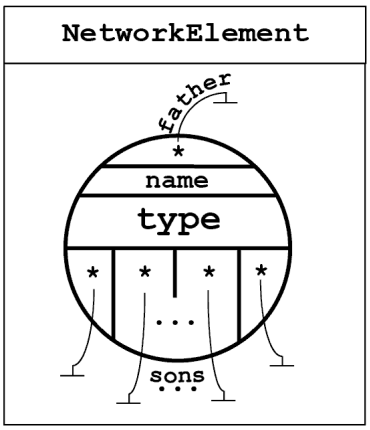
A continuación se explica brevemente las ideas desarrolladas en cada una de las tareas.

2.1. NetworkElement Class

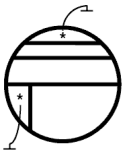
La clase NetworkElement está pensada como una estructura de datos que modela elementos para topologías de redes HFC. Básicamente instancia objetos que disponen de los siguientes atributos:

- Nombre y Tipo;
- Puntero único a objeto padre;
- Punteros múltiples a objetos hijos;
- Cantidad de objetos hijos;

En la figura siguiente se muestra un esquema del tipo de objeto que puede ser instanciado por la clase.



El diseño de los constructores con y sin argumentos, permite que se pueda instanciar un objeto NetworkElement ya sea asignándole todos los datos necesarios, o bien ninguno. En este último caso, se instancia un Elemento de Red vacío, cuyo esquema es el siguiente:



Luego, los métodos incluídos permiten la asignación de los parámetros del elemento de red, así como la lectura. El código autodocumentado del encabezado de la clase es el detallado en la sección CODE::NetworkElement Class.

2.2. Validación de flujos de entrada y salida

Para resolver el ingreso y egreso de datos, se utilizó la clase cmdline provista por los docentes. Esta clase, permite recorrer e identificar las opciones y argumentos con los que se invoca al programa mediante línea de comandos. Luego se definen una serie de métodos que permiten asignar funciones determinadas según las opciones leídas.
En particular, se definieron las funciones que deben corresponderse con los argumentos de entrada leídos y validados por cmdline() en archivos denominados `options.hpp, options.cpp`. Los encabezados de la clase cmdline y de options se aduntan en la sección CODE::cmdline y CODE::options.

El tipo de opciones admitidas para este programa están detalladas en los diccionarios globales. El detalle de los diccionarios está en la sección CODE::Dictionary.

2.3. Proceso del archivo de entrada

Luego de la validación del archivo de entrada, se procede a leer línea po línea el archivo de entrada, ya sea que éste sea un archivo en disco, o bien el flujo de

entrada estándar (cin).

Se implementó un método de la clase *sstream* que permite operar con un string como si fuera un stream[1], y con esto se logró la facilidad de poder identificar las palabras de cada línea mediante el operador sobrecargado <<.

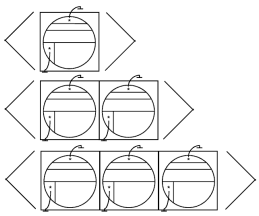
2.3.1. Nombre de Red

Asumiendo que los archivos de entrada tienen como primera línea el nombre de red, se implementó una función que recibe la cadena de caracteres mediante *getline()* correspondiente a la primer línea del archivo. Esta función desarrollada es *getNetName()* y se encarga de validar que el nombre de red esté bien escrito y lo guarda en un string nomenclado NetName. En caso de que la validación no sea aceptable, se imprime a través del flujo *cerr* el siguiente mensaje de error:

```
error: missing NetworkName
```

2.3.2. Vector de Elementos de Red

Se utilizó la clase Vector de la biblioteca estándar. Por medio de los métodos de inserción, se logró la implementación de un vector dinámico: los elementos de red se validan y se generan de uno en uno, insertandose al final del vector. Con esto se logra que el tamaño del vector quede determinado en tiempo de ejecución. La siguiente imagen representa un esquema de lo que ocurre a medida que el programa lee línea por línea, la clave NetworkElement:



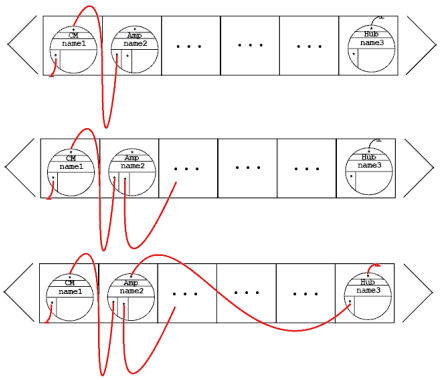
Básicamente se crea un objeto NetworkElement sin argumentos y se lo adjunta al final del vector, en cada iteración. Una vez finalizada la lectura de NetworkElements, el vector queda completo de objetos y queda determinada la cantidad de elementos de red disponibles en la topología.

2.3.3. Conexionado

Para realizar las conexiones correspondientes entre elementos, se procede con la localización y conexión de los elementos que se leen en el texto de entrada bajo el formato: `Connection <name1><name2>`. En este punto del programa, ya se dispone del vector de elementos de red completo de objetos con sus respectivos nombres y tipos, por lo que la estrategia pensada resuelve buscar por nombres y conectar los punteros padre y punteros hijo según corresponda.

[1] método istringstram iss(str)

Para lograr esto se implementó un ciclo que recorre el vector desde el primer elemento hasta el último, y que en cada iteración, busca los nombres $< name1 >$ y $< name2 >$ y los conecta. La figura siguiente es un esquema de lo que realiza el ciclo en este punto del programa:



Básicamente lee elemento a elemento buscando las claves leídas y conecta los elementos de a uno por vez. El orden no es necesariamente incremental, la imagen es sólo útil para fijar ideas.

2.4. Detección de loops

Para la detección de loops, se implementó una función recursiva que básicamente se encarga de recorrer el árbol creado previamente. Recorre en profundidad, saliendo de la misma en caso de detectar ciclos o árboles inconexos. Recibe un entero por referencia 'vertice' que guarda el numero de conexiones realizadas, y un arreglo de punteros a NetworkElement temporal, donde se guardan la direccion de memoria de los objetos recorridos. La función trabaja de manera recursiva e iterativa garanztizando que siempre que el arbol este armado puede vistar a todos los nodos del mismo. El valor final que queda guardado en la variable 'vertice' es la cantidad de nodos del arbol excepto la raiz (ver macro ROOT), luego de realizar la funcion recorrido()".

A partir de este método de recorrido, se implementaron luego pequeñas funciones encargadas de imprimir mensajes de salida si hay elementos repetidos en la topología, si hay ciclos, o bien si hay árboles conexos e inconexos. Las funciones mencionadas son: `validateIconnection(); isRepeaten(); validateCycle()`.

El detalle de implementación de esta función se puede encontrar en la sección CODE::NetworkElement Class.

2.5. Archivo de salida

El archivo de salida tiene un formato idéntico al archivo de entrada, pero ahora la información recibida tiene una estructura en memoria. El nombre de red está guardado en la variable NetName. La topología de red está guardada

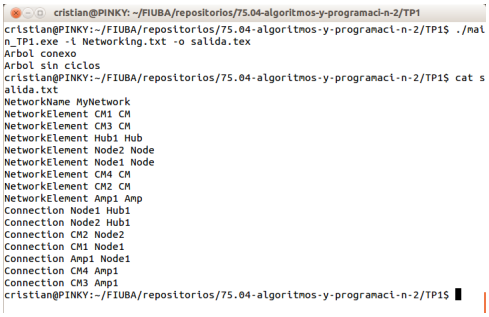
en el vector v, y a partir de recorrer el vector, se imprime el contenido de cada nodo en el siguiente formato:
`NetworkElement <name><type>`
Luego se recorre nuevamente el vector y se imprimen las conexiones en el siguiente formato:
`Connection <sons><name>`
Esta última línea merece una explicación, pues la impresión de `<sons>` se realiza de forma iterativa para todos los elementos de red hijos que tiene el nodo posicionado de nombre `<name>`.

3. Ejecuciones

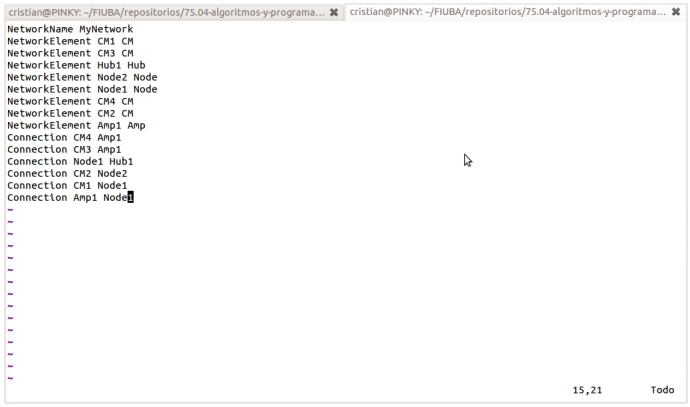
3.1. Compilación

La compilación del programa se realiza por medio de un Makefile. El detalle se encuentra en la sección CODE::Makefile. Como salida, se genera un archivo ejecutable de extension.exe.

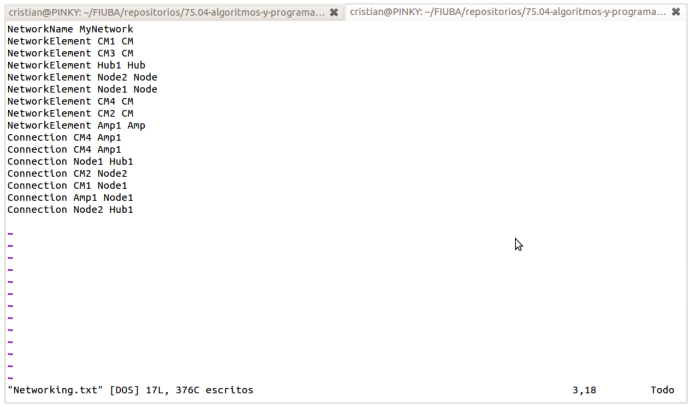
Para ejecutar el programa se necesita invocar con la siguiente línea de comandos: $./main_TP1.exe - i < entrada.txt > -o < salida.txt >$ Las pruebas que se realizaron contemplan archivos correctos de entrada, y algunos ejemplos que contienen topologías inconexas y topologías con ciclos. Los resultados se adjuntan a continuación:
Entrada correcta



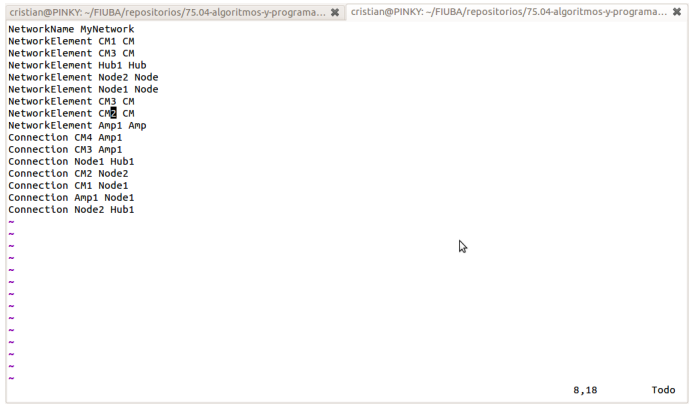
Entrada de malas conexiones
Entrada de árbol inconexo

de arbol inconexo.png 

Entrada de malas conexiones

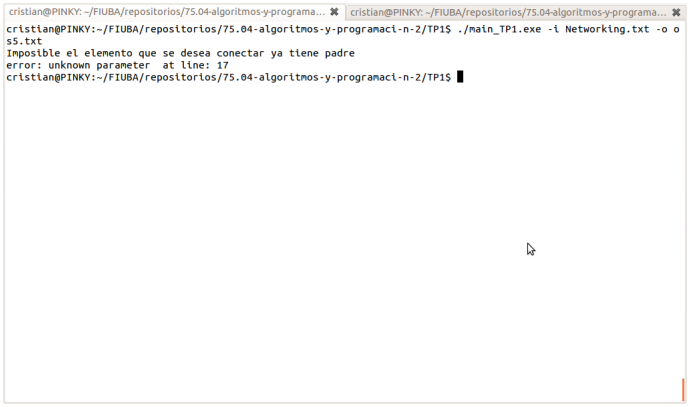
de malas conexiones.png 

Entrada de múltiple nodos

de multiple nodos.png 

Malas conexiones

conexiones.png

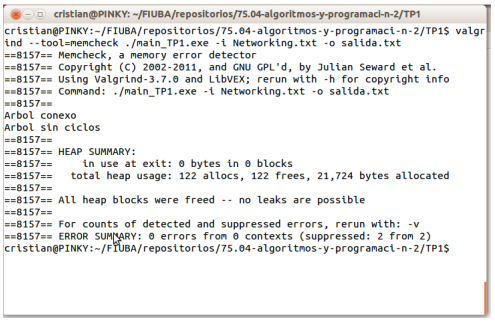


Múltiple nodos

nodos.png

3.2. Comprobación de uso de memoria con Valgrind

Utilizando la ejecucion del programa Valgrind bajo el siguiente formato: $valgrind --tool = memcheck./main_TP1.exe - iNetworking.txt - osalida.txt$ se obtiene el siguiente resultado:



que verifica que no hay fugas de memoria.

4. Códigos

CODE::Makefile

```
#compilacion y ejecucion de los archivos
CC       =  g++
CFLAGS   = -g -Wall  -pedantic
OBJS     = main_TP1.o  printers.o process.o dictionary.o
     NetworkElementClass.o cmdline.o options.o



muestras.exe:$(OBJS)
        $(CC)  $(OBJS) -o main_TP1.exe

main_TP1.o:main_TP1.cpp dictionary.hpp printers.hpp
    process.hpp  common.hpp cmdline.h options.hpp
    NetworkElementClass.hpp
        $(CC) main_TP1.cpp -c -o main_TP1.o $(CFLAGS)

printers.o:printers.cpp printers.hpp common.hpp
        $(CC) printers.cpp -c -o printers.o $(CFLAGS)

process.o:process.cpp process.hpp common.hpp printers.hpp
        $(CC) process.cpp -c -o process.o $(CFLAGS)

dictionary.o:dictionary.cpp dictionary.hpp cmdline.h
    options.hpp
        $(CC) dictionary.cpp -c -o dictionary.o $(CFLAGS)

NetworkElementClass.o:NetworkElementClass.cpp
    NetworkElementClass.hpp common.hpp
        $(CC) NetworkElementClass.cpp -c -o
            NetworkElementClass.o $(CFLAGS)

cmdline.o: cmdline.cc cmdline.h
        $(CC)  cmdline.cc -c -o cmdline.o $(CFLAGS)

options.: options.cpp options.hpp
        $(CC)  options.cpp -c -o options.o $(CFLAGS)



clean:
        rm *.o
```

CODE::main

```
#include <iostream>
```

```
2 #include <fstream>
3 #include <string>
4 #include <sstream>
5 #include <cstring>
6 #include"common.hpp"
7 #include "dictionary.hpp"
8 #include "process.hpp"
9 #include "NetworkElementClass.hpp"
10 #include "cmdline.h"
11 #include "options.hpp"
12 #include <vector>
13 #include <algorithm>
14
15 using namespace std;
16
17 istream *iss = 0;
18 ostream *oss = 0;
19 fstream ifs, ofs;
20 extern option_t options[];
21 extern string network_element_type[];
22
23 string getNetName(string);
24 bool NetworkElementType(string);
25
26 /**** MAIN ****/
27
28 int main(int argc,char *argv[])
29 {
30 //OPTIONS AND ARGUMENTS VALIDATION
31         cmdline cmdl(options);
32         cmdl.parse(argc, argv);
33
34 //NetworkName
35         string str;
36         string NetName;
37     getline(*iss,str);
38
39     NetName=getNetName(str);
40         if(NetName=="error: missing NetworkName"){
41                 cerr << NetName << endl;
42                 //cout << NetName << endl;
43                 return 1;
44                 }
45     size_t line=1;
46
47 //Vector of NetworkElements
48     vector <NetworkElement> v;
49     size_t i=0;
50
51     while( getline(*iss,str) )
```

```
52      {
53              line++;
54          //Setting Up the vector: NetworkElements
55              string aux;
56              istringstream iss(str);
57              iss >> aux;
58
59          //Wrong text
60              if(aux!="NetworkElement" && aux!= "Connection
                    ")
61              {
62                  cerr << "error: unknown parameter "<< aux
                        <<" at line: " << line << endl;
63                  //delete all memory assigned
64                  return 1;
65              }
66
67              if(aux=="NetworkElement")
68              {
69                  v.push_back(NetworkElement());
70                  iss >> aux;
71                  v[i].setName(aux);
72                  iss >> aux;
73                  if(!NetworkElementType(aux)){
74                          cerr << "error: unrecognized
                                Networkelement " << aux <<
                                endl;
75                          return 1;
76                  }
77                  v[i].setType(aux);
78                 // v[i].showContent(ofs);
79                  i++;
80              }
81
82
83          //Setting Up the Connections
84              if(aux=="Connection")
85              {
86                  string aux1, aux2;
87                  size_t i1, i2;
88                  bool son=false,father=false;
89                  iss >> aux1;
90                  iss >> aux2;
91                  for(size_t i=0; i<v.size();i++){
92
93
94                                          if(v[i].getName()
                                                ==aux1)
95                                          {
96                                                  i1=i;
```

```
97                                          if(son==false){son=true;}
98                                          else {son=false;}
99                                      }
100                 if(v[i].getName()==aux2)
101                                     {
102                                         i2=i;
103                                         if(father==false){father=true;}
104                                         else {father=false;}
105                 }
106         }
107
108
109         if(son==false)
110                                 {
111                                     cout << "Son node isn't found or multiple found"<<endl
112                                          << aux <<" Error at line: " << line <<endl;
113                                 }
114         if(father==false)
115                                 {
116                                     cout << "Father node isn't found or multiple found "<<endl
```

```
117                                                     << aux
                                                        <<"
                                                        Error
                                                         at
                                                        line:
                                                         " <<
                                                         line
                                                        <<
                                                        endl;
118                                     }
119
120
121
122                     //cout <<"index of son= "<<i1 <<" and
                            index of father= "<<i2 <<endl;
123                    // if(son==true && father==true)
124                      //    v[i2].connectToElement(v[i1]);
125                                     if(son==true && father==
                                            true)
126                                     {
127
128                                     if(v[i2].
                                            validateHierarchy(v[i1
                                            ])==true)
129                                 v[i2].connectToElement(v[i1]);
130                                     else
131                                     cout <<"Failure hierarchy
                                            . Unable to connect
                                            the elements"<<endl
132                                     << v[i2].getName() << "(
                                            father) with "<< v[i1
                                            ].getName() << "(son)"
                                            <<endl
133                                     << aux <<" Error at line:
                                             " << line << endl;
134                                     }
135
136             }
137
138     }
139 //ACA IMPRIME SEGUN EL ORDEN DEL ARREGLO ENTONCES SE
        DEBERIA IMPRIMIR SEGUN LA JERARQUIA ..!!!111
140             ofs << "NetworkName "<<NetName << endl;
141             for(size_t i=0; i< v.size(); i++)
142                                     v[i].showElements(ofs);
143             for(size_t i=0; i< v.size(); i++)
144                                     v[i].showConnections(ofs)
                                            ;
145
146
```

```
147         //Encuentro el hub1------
148         size_t rootPosition=FindRoot(v);
149
150         //Empieza las validaciones de ciclos e
                inconexiones
151
152 //      cout<<v.data()[rootPosition].getName()<<endl;
153         v.data()[rootPosition].validateIconnection(v.size
                ());
154         v.data()[rootPosition].isRepeaten(v);
155         v.data()[rootPosition].validateCycle();
156             return 0;
157 }
158
159 string getNetName(string str)
160 /*getNetName() valida el primer string
161 del archivo de entrada, que ya es asignado
162 como string previamente. */
163 {
164     string aux;
165     istringstream iss(str);
166     iss >> aux;
167     if(aux=="NetworkName"){
168         iss >> aux;
169         return aux;
170         }
171     else
172         return "error: missing NetworkName";
173 }
174
175
176 #define NET_TYPES 4
177 bool NetworkElementType(string aux)
178 {
179     for( size_t i=0;i<NET_TYPES;i++){
180         if(aux==network_element_type[i])
181             return true;
182     }
183     return false;
184 }
```

CODE::Dictionary

```
1 #ifndef DICTIONARY_HPP_INCLUDED
2 #define DICTIONARY_HPP_INCLUDED
3 #include<string>
4 #include"common.hpp"
5 #define MAX_LINES_DEFAULT 100
6
7
```

```
8
9
10
11
12 #endif // DICTIONARY_HPP_INCLUDED
```

```
1 #include"dictionary.hpp"
2 #include"cmdline.h"
3 #include"options.hpp"
4
5 //Diccionarios: HAY QUE ACORDAR UNA NOMNECLATURA PARA
      STRINGS , VARIABLES y ARREGLOS.
6 //Diccionarios: HAY QUE ACORDAR UNA NOMNECLATURA PARA
      STRINGS , VARIABLES y ARREGLOS.
7
8 string network_struct[]    = {"NetworkName","
      NetworkElement","Connection"};
9 string network_element_type[] = {"Hub","Node","Amp","CM"
      };
10 string network_element_name[MAX_LINES_DEFAULT];
11
12 option_t options[] = {
13         {1, "i", "input", "-", opt_input, OPT_DEFAULT},
14         {1, "o", "output", "-", opt_output, OPT_DEFAULT},
15         {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
16         {0, },
17 };
18
19
20
21 //extern static size_t   n, i;
```

CODE::NetworkElement Class

```
1 #ifndef NETWORK_ELEMENT_CLASS_HPP
2 #define NETWORK_ELEMENT_CLASS_HPP
3
4 #include<iostream>
5 #include<vector>
6 #include "common.hpp"
7 #define EXIST 0
8 #define NOT_EXIST 1
9 #define BACK_TREE 0;
10 #define DETECT_CYCLE 1
11 #define ROOT 1
12 #define ONE 1//valida que no se repitan mas de un
      NetworkElement
13
14
15
```

```
16 class errorsubindice
17 {
18         public:
19         errorsubindice()
20         {
21                 cout<<"Error de subindice"<<endl;
22         }
23 };
24
25
26 class NetworkElement
27 {
28         private:
29
30                 string name;
31                 string type;
32                 NetworkElement *father_;
33                 NetworkElement **sons; // Sera un punetro
                        a un arreglo dinamico de punteros, lo
                        declaro asi ya que genera error
                       declarar *sons[]
34                 size_t numberSons;
35
36         public:
37
38         /********************** CONSTRUCTORES
               *******************************/
39                 NetworkElement();
40                 NetworkElement(const string,const string)
                       ; // (NOMBRE, TIPO)
41
42                 // Deberia poder copiarse los nodos?
43                 // Constructor por copia:
44                 NetworkElement(const NetworkElement&);
45                 ~NetworkElement();
46
47         /************************** SET & GET
               *******************************/
48
49                 void setName (string n) {name=n;}
50                 void setType (string t) {type=t;}
51
52                 const string getName()const {return name
                       ;}
53                 const string getType()const {return type
                       ;}
54                 const NetworkElement* getFather()const {
                       return father_;}
55                 const size_t getNumberSons() const {
                       return numberSons;}
```

```
56              NetworkElement** getSons() const {return
                    sons;} // Revisar no me deja el
                    compilador retornar 'const
                    NetworkElement**'
57              const NetworkElement* getSons(const int)
                    const;
58
59
60              // getSons(): Sin argumentos, retorna el
                    puntero al arreglo de punteros a
                    NetworkElement hijos
61              // getSons(int): Con argumentos, retorna
                    el puntero de determinado hijo
62
63              /************************ OPERADORES
                    *********************************/
64
65          NetworkElement& operator = (const NetworkElement
                &); // operador asignacion
66          bool operator == (const NetworkElement&) const;
                // operador comparacion igualdad
67              bool operator != (const NetworkElement&)
                    const; // operador comparacion
                    desigualdad
68              NetworkElement& operator [ ](int);
69
70          /************************ MeTODOS
                ***********************************/
71
72              // Se asume que se conecta al ingresar:
                    hijo --> padre
73          NetworkElement& connectToElement(NetworkElement&)
                ;
74
75              // Funcion validacion de jerarquia de los
                    elementos
76              bool validateHierarchy(NetworkElement&);
77
78              void showContent(ostream&);
79              void showElements(ostream&);
80              void showConnections(ostream&);
81
82
83          /************************ MeTODOS  DE RECORRIDO
                DE ARBOL************************************/
84
85
86              friend int recorrido(NetworkElement* ,int
                    &,vector <NetworkElement*>&temp);
```

```
87                  friend int comparator(NetworkElement* ,
                        int&,vector <NetworkElement*>&temp);
88                  void validateCycle();
89                  void validateIconnection(int);
90                  void isRepeaten(vector <NetworkElement>&)
                        ;
91 };
92
93
94
95  #endif
```

```
1 #include <iostream>
2 #include "NetworkElementClass.hpp"
3
4
5
6 /********************************* CONSTRUCTORES
      ************************************/
7
8 NetworkElement :: NetworkElement()
9 {
10          name=" ";
11          type=" ";
12          father_=NULL;
13          sons=NULL;
14          numberSons=0;
15          //cout<<"Constructor sin argumentos"<<endl;
16 }
17
18 NetworkElement :: NetworkElement(const string n,const
      string t)
19 {
20          name=n;
21          type=t;
22          father_=NULL;
23          sons=NULL;
24          numberSons=0;
25          //cout<<"Constructor con argumentos"<<endl;
26 }
27
28 NetworkElement :: NetworkElement(const NetworkElement &
      element)
29 {
30          name=element.name;
31          type=element.type; // Por defecto dejo Cable
              Modem, luego se cambiara
32          father_=element.father_;
33          numberSons=0;
34          sons=NULL;
```

```
35          if(element.numberSons!=0)
36          {
37                  numberSons=element.numberSons;
38
39                  sons=new NetworkElement* [element.
                        numberSons];
40
41                  for(unsigned int i=0 ; i<numberSons ; i
                        ++)
42                  {
43                          sons[i]=element.sons[i];
44                  }
45          }
46
47  }
48
49  NetworkElement :: ~NetworkElement()
50  {
51          delete [] sons;
52
53  }
54
55  /************************************ SET & GET
        ****************************************/
56
57  const NetworkElement* NetworkElement :: getSons(const int
        subscript)const
58  {
59          if( (subscript < 0) || (subscript > (int)
                numberSons) ) throw errorsubindice();
60
61          else     return sons [subscript];
62
63  }
64
65  /************************************ OPERADORES
        ****************************************/
66
67  NetworkElement& NetworkElement :: operator = (const
        NetworkElement &Relement)
68  {
69          // Relement: element a la derecha del operador
                asignacion,
70          // que asigna al objeto sobre el que actua.
71
72          // Se chequea que la direccion de memoria del
                objeto
73          // pasado no sea la misma con el que estamos
                aplicando el
```

```
74          // metodo, para evitar auto asignaciones. Ejemplo
                : a=a;
75
76          if ( &Relement != this )
77          {
78                  name=Relement.name;
79                  type=Relement.type;
80                  father_=Relement.father_;
81
82                  if ( numberSons != Relement.numberSons )
83                  {
84                          NetworkElement **auxSons;
                                // Utilizo un elemento
                                auxiliar para los punteros a
                                los hijos. Sera un arreglo
                                dinamico de punteros
85
86                          auxSons = new NetworkElement* [
                                Relement.numberSons]; //se
                                pide espacio; si no se obtiene
                                 new lanza bad_alloc
87                          delete [] sons;
                                                // Si llego
                                aca es que obtuvo el espacio;
                                libera el anterior espacio
88                          numberSons = Relement.numberSons;
89                          sons = auxSons;

                                                // Apunta a la
                                 nueva zona
90
91                          for (unsigned int i = 0 ; i <
                                numberSons ; i++ )
92                                  sons[i] = Relement.sons[i
                                        ];
                                                // Copia el
                                        array en el nuevo
                                        objeto
93
94                          return *this;
                                                        // Al
                                retornar una referencia
                                permite x = y = z;
95                  }
96
97
98
99                  else
100                 {
```

```
101                         for (unsigned int i = 0 ; i <
                                numberSons ; i++ )
102                                 sons[i] = Relement.sons[i
                                        ];
103                         return *this;
104                 }
105
106         }
107
108         return *this;
109
110 }
111
112 bool NetworkElement :: operator == (const NetworkElement
        &element) const
113 {
114         // Solo estoy considerando la igualdad si los
                campos de nombre
115         // y tipo son los mismos. Se considera que nunca
                deberia haber
116         // dos elementos que se llamen iguales y que
                tengas hijos y
117         // padres distintos.
118
119         if (name == element.name && type == element.type)
                return true;
120
121         else return false;
122 }
123
124 bool NetworkElement :: operator != (const NetworkElement&
        element) const
125 {
126         // Idem que el caso del operador '==' pero
                comparando la desigualdad
127
128         if (name != element.name && type != element.type)
                return true;
129
130         else return false;
131 }
132
133 /************************************ MeTODOS
        ******************************************/
134
135 // El metodo connectToElement lo supongo su
        funcionamiento como que previo a elanzar padres
136 // e hijos respectivamente se encuentran validadas los
        casos de jerarquia donde determinados
```

```
137 // elementos no se puede unir, como un "nodo optico(node)
    " con un "cable modem (cm)"
138 //
139 // IMPORTANTE: Se supone que la coneccion correcta se
    realizara de la siguiente manera al
140 // invocar el metodo.
141 //                                                padre.
    connectToElement ( hijo )
142
143 NetworkElement& NetworkElement :: connectToElement (
    NetworkElement &element)
144 {
145         if(this != &element)
146         {
147                 // Se chequea que la direccion de memoria
                    del objeto
148                 // pasado no sea la misma con el que
                    estamos aplicando el
149                 // metodo, para evitar auto asignaciones.
                    Ejemplo: a=a;
150
151                 if(element.father_ ==NULL) // Valido que
                    el nuevo elemento no tenga padre ya
152                 {
153                         // Enlace: padre <-- hijo
154
155                         element.father_ = this; // Asigno
                            la direccion del objeto sobre
                            el que trabajo, al campo
                            padre del nuevo elemento (hijo
                            )
156
157                         // Enlace: hijo --> padre
158
159                         NetworkElement **auxSons;
                            // Utilizo un elemento
                            auxiliar para los punteros a
                            los hijos. Sera un arreglo
                            dinamico de punteros
160
161                         auxSons = new NetworkElement* [
                            numberSons+1]; // Se pide
                            espacio; si no se obtiene new
                            lanza bad_alloc
162
163
164                         if(sons!=NULL)
165                         {
166                                 for(unsigned int i=0 ; i<
                                    numberSons ; i++)
```

```
167                                     {
168                                             auxSons[i] = sons
                                                    [i];
169                                     }
170                             }
171                             auxSons[numberSons] = &element;
                                    // Guardo la direccion de
                                    memoria del que sera el nuevo
                                    hijo
172
173                             delete [] sons;         // Si
                                    llego aca es que obtuvo el
                                    espacio; libera el anterior
                                    espacio para sons
174                             sons = auxSons;         // Asigno
                                     a sons el puntero que apunta
                                    al nuevo array de hijos
175
176                             numberSons++;
177                     }
178
179                     else
180                     {
181                             // Buscar que devolver en caso de
                                     que no se pueda unir, todavia
                                     no se me ocurrio
182                             cout<<"Imposible el elemento que
                                    se desea conectar ya tiene
                                    padre"<<endl;
183                     }
184             }
185             return *this;
186 }
187
188 /*      Funcion que valida la jerarquia, evita que se
        realicen conexiones prohibidas del tipo
189         por ejemplo: CM1 --> CM2 o HUB1 --> CM2
190
191         padre.validateHierarchy(hijo)
192
193 */
194 bool NetworkElement :: validateHierarchy(NetworkElement &
        element)
195 {
196
197         if(type=="CM")
198         {       if(element.type=="CM" || element.type=="
                    Amp" || element.type=="Node" || element.type==
                    "Hub" ) return false; }
199
```

```
200         else if(type=="Amp" || type=="Node")
201         {       if(element.type=="Hub" || element.type=="
                Node") return false; }
202
203         else if(type=="Hub")
204         {       if(element.type=="Hub" || element.type=="
                Amp" || element.type=="CM") return false; }
205
206
207         return true;
208 }
209
210
211 void NetworkElement :: showContent(ostream& os)
212 {
213                 os//<<"***** Elemento de red *****"<<endl
214                 <<"NetworkElement "<<name//<<endl
215                 <<" "<<type<<endl;
216                 //<<"Cantidad de hijos: "<<numberSons<<
                        endl;
217 //      os << "number of sons: " <<numberSons << endl;
218         if(numberSons!=0)
219         {
220                 for(unsigned int i=0 ; i < numberSons; i
                        ++)
221                 os<<"Connection " << sons[i]->name <<" "
                        <<  name << endl;
222         }
223
224 }
225
226 void NetworkElement :: showElements(ostream& os)
227 {
228         os <<"NetworkElement "<<name
229            <<" "<<type<<endl;
230 }
231
232 void NetworkElement :: showConnections(ostream& os)
233 {
234         if(numberSons!=0)
235         {
236                 for(unsigned int i=0 ; i < numberSons; i
                        ++)
237                 os<<"Connection " << sons[i]->name <<" "
                        <<  name << endl;
238         }
239 }
240
241 void NetworkElement ::validateCycle()
242 {
```

```
243
244         int vertice=0;
245         vector <NetworkElement*> temp;
246         if((recorrido(this,vertice,temp))==DETECT_CYCLE)
247         {
248                 // asignarle elemento desde donde quieres
                        recorrer
249             // vertice contiene la cantidad de nodos que
250         // que pudieron ser recorridos.
251
252                 cout<< "Se en encontro un ciclo en:\t "<<
                    temp.data()[vertice]->getName()<<endl;
253
254         }
255         else
256                 cout<<"Arbol sin ciclos"<<endl;
257 }
258
259 void NetworkElement::validateIconnection(int numberNodes
    )
260 {
261
262         int vertice=0;
263         vector <NetworkElement*> temp;
264         if((recorrido(this,vertice,temp))!=DETECT_CYCLE)
265         {
266                 if(vertice + ROOT < numberNodes)
267                 {
268                         cout<< "Arbol inconexo."<<endl
269                                 << "Cantidad de vertices
                                        registrados:\t"<<
                                        vertice<<endl;
270                 }
271                 else
272                         cout<< "Arbol conexo"<<endl;
273         }
274 }
275
276
277 // Recorrer vector de elementos y revisar si hay
    elementos repetidos
278 void NetworkElement :: isRepeaten(vector <NetworkElement
    >& vectorElement)
279 {
280
281         int vertice=0,repeat=0;
282         vector <NetworkElement*> temp;
283         if((recorrido(this,vertice,temp))!=DETECT_CYCLE)
284         {
```

```
285                     for(size_t i=0;i<vectorElement.size();i
                            ++)
286                     {
287                             // En la posicion "vertice" del
                                    vector "temp" esta el elemento
                                     detectado como posible
288                             // candidato a estar repetido, ya
                                    que la funcion "recorrido" lo
                                    detecta como un ciclo.
289
290                             if(*(temp.data()[vertice])==
                                    vectorElement.data()[i])
                                    repeat++;
291                     }
292                     if(repeat>ONE){
293                             cout<< "Elemento repetido:\t "<<
                                    temp.data()[vertice]->getName
                                    ()<<endl;
294                     }
295
296             }
297
298
299 }
300
301
302
303 int recorrido(NetworkElement *v,int &vertice,vector <
        NetworkElement*> &temp){
304
305             if(comparator(v,vertice,temp)==EXIST){return
                    DETECT_CYCLE;}
306             temp.push_back(v);
307             //cout<<"Nombres guardados: "<<v->getName()<<endl
                    ;
308         if(temp[vertice]->numberSons==0){ return BACK_TREE;}
309             else
310             for(size_t i=0;i<v->numberSons;i++)
311                     {
312                             vertice++;
313                             if((recorrido(v->sons[i],vertice,
                                    temp))==DETECT_CYCLE){return
                                    DETECT_CYCLE;}
314             }
315                     return BACK_TREE;
316 }
317
318 //Esta funcion verifica si el elemeto esta repetido,
        entonces se deriva en un ciclo
```

```
319 int comparator(NetworkElement *v,int &vertice,vector<
    NetworkElement*>&temp){
320
321
322         for(int i=0;i<vertice;i++){
323
324                 if(v->name==temp.data()[i]->name)
325                 {
326                         temp.push_back(v);
327
328                         return EXIST;
329                 }
330
331         }
332
333         return NOT_EXIST;
334 }
```

CODE::cmdline

```
1 #ifndef _CMDLINE_H_INCLUDED_
2 #define _CMDLINE_H_INCLUDED_
3
4 #include <string>
5 #include <iostream>
6
7 #define OPT_DEFAULT   0
8 #define OPT_SEEN      1
9 #define OPT_MANDATORY 2
10
11 struct option_t {
12         int has_arg;
13         const char *short_name;
14         const char *long_name;
15         const char *def_value;
16         void (*parse)(std::string const &);
17         int flags;
18 };
19
20 class cmdline {
21         // Este atributo apunta a la tabla que describe
             todas
22         // las opciones a procesar. Por el momento, solo
             puede
23         // ser modificado mediante contructor, y debe
             finalizar
24         // con un elemento nulo.
25         //
26         option_t *option_table;
27
```

```
28          // El constructor por defecto cmdline::cmdline(), es
29          // privado, para evitar construir parsers sin opciones.
30          //
31          cmdline();
32          int do_long_opt(const char *, const char *);
33          int do_short_opt(const char *, const char *);
34  public:
35          cmdline(option_t *);
36          void parse(int, char * const []);
37  };
38
39  #endif
```

```
1   // cmdline - procesamiento de opciones en la linea de comando.
2   //
3   // $Date: 2012/09/14 13:08:33 $
4   //
5   #include <string>
6   #include <cstdlib>
7   #include <iostream>
8   #include "cmdline.h"
9
10  using namespace std;
11
12  cmdline::cmdline()
13  {
14  }
15
16  cmdline::cmdline(option_t *table) : option_table(table)
17  {
18  }
19
20  void
21  cmdline::parse(int argc, char * const argv[])
22  {
23  #define END_OF_OPTIONS(p)       \
24          ((p)->short_name == 0   \
25           && (p)->long_name == 0 \
26           && (p)->parse == 0)
27
28          // Primer pasada por la secuencia de opciones: marcamos
29          // todas las opciones, como no procesadas. Ver codigo de
30          // abajo.
31          //
```

```
32          for (option_t *op = option_table; !END_OF_OPTIONS
                (op); ++op)
33                  op->flags &= ~OPT_SEEN;
34
35          // Recorremos el arreglo argv. En cada paso,
                vemos
36          // si se trata de una opcion corta, o larga.
                Luego,
37          // llamamos a la funcion de parseo
                correspondiente.
38          //
39          for (int i = 1; i < argc; ++i) {
40                  // Todos los parametros de este programa
                        deben
41                  // pasarse en forma de opciones.
                        Encontrar un
42                  // parametro no-opcion es un error.
43                  //
44                  if (argv[i][0] != '-') {
45                          cerr << "Invalid non-option
                                argument: "
46                               << argv[i]
47                               << endl;
48                          exit(1);
49                  }
50
51                  // Usamos "--" para marcar el fin de las
52                  // opciones; todo los argumentos que
                        puedan
53                  // estar a continuacion no son
                        interpretados
54                  // como opciones.
55                  //
56                  if (argv[i][1] == '-'
57                      && argv[i][2] == 0)
58                          break;
59
60                  // Finalmente, vemos si se trata o no de
                        una
61                  // opcion larga; y llamamos al metodo que
                         se
62                  // encarga de cada caso.
63                  //
64                  if (argv[i][1] == '-')
65                          i += do_long_opt(&argv[i][2],
                                argv[i + 1]);
66                  else
67                          i += do_short_opt(&argv[i][1],
                                argv[i + 1]);
68          }
```

```
69
70          // Segunda pasada: procesamos aquellas opciones
                que,
71          // (1) no hayan figurado explicitamente en la
                linea
72          // de comandos, y (2) tengan valor por defecto.
73          //
74          for (option_t *op = option_table; !END_OF_OPTIONS
                (op); ++op) {
75  #define OPTION_NAME(op) \
76          (op->short_name ? op->short_name : op->long_name)
77                  if (op->flags & OPT_SEEN)
78                          continue;
79                  if (op->flags & OPT_MANDATORY) {
80                          cerr << "Option "
81                               << "-"
82                               << OPTION_NAME(op)
83                               << " is mandatory."
84                               << "\n";
85                          exit(1);
86                  }
87                  if (op->def_value == 0)
88                          continue;
89                  op->parse(string(op->def_value));
90          }
91  }
92
93  int
94  cmdline::do_long_opt(const char *opt, const char *arg)
95  {
96          // Recorremos la tabla de opciones, y buscamos la
97          // entrada larga que se corresponda con la opcion
                de
98          // linea de comandos. De no encontrarse,
                indicamos el
99          // error.
100         //
101         for (option_t *op = option_table; op->long_name
                != 0; ++op) {
102                 if (string(opt) == string(op->long_name))
                        {
103                         // Marcamos esta opcion como
                            usada en
104                         // forma explicita, para evitar
                            tener
105                         // que inicializarla con el valor
                            por
106                         // defecto.
107                         //
108                         op->flags |= OPT_SEEN;
```

```
109
110                         if (op->has_arg) {
111                                 // Como se trada de una
                                        opcion
112                                 // con argumento,
                                        verificamos que
113                                 // el mismo haya sido
                                        provisto.
114                                 //
115                                 if (arg == 0) {
116                                         cerr << "Option
                                                requires
                                                argument: "
117                                                 << "--"
118                                                 << opt
119                                                 << "\n";
120                                         exit(1);
121                                 }
122                                 op->parse(string(arg));
123                                 return 1;
124                         } else {
125                                 // Opcion sin argumento.
126                                 //
127                                 op->parse(string(""));
128                                 return 0;
129                         }
130                 }
131         }
132
133         // Error: opcion no reconocida. Imprimimos un
                mensaje
134         // de error, y finalizamos la ejecucion del
                programa.
135         //
136         cerr << "Unknown option: "
137              << "--"
138              << opt
139              << "."
140              << endl;
141         exit(1);
142
143         // Algunos compiladores se quejan con funciones
                que
144         // logicamente no pueden terminar, y que no
                devuelven
145         // un valor en esta ultima parte.
146         //
147         return -1;
148 }
149
```

```
150 int
151 cmdline::do_short_opt(const char *opt, const char *arg)
152 {
153         option_t *op;
154 
155         // Recorremos la tabla de opciones, y buscamos la
156         // entrada corta que se corresponda con la opcion
              de
157         // linea de comandos. De no encontrarse,
              indicamos el
158         // error.
159         //
160         for (op = option_table; op->short_name != 0; ++op
              ) {
161                 if (string(opt) == string(op->short_name)
                      ) {
162                         // Marcamos esta opcion como
                              usada en
163                         // forma explicita, para evitar
                              tener
164                         // que inicializarla con el valor
                               por
165                         // defecto.
166                         //
167                         op->flags |= OPT_SEEN;
168 
169                         if (op->has_arg) {
170                                 // Como se trata de una
                                      opcion
171                                 // con argumento,
                                      verificamos que
172                                 // el mismo haya sido
                                      provisto.
173                                 //
174                                 if (arg == 0) {
175                                         cerr << "Option
                                              requires
                                              argument: "
176                                           << "-"
177                                           << opt
178                                           << "\n";
179                                         exit(1);
180                                 }
181                                 op->parse(string(arg));
182                                 return 1;
183                         } else {
184                                 // Opcion sin argumento.
185                                 //
186                                 op->parse(string(""));
187                                 return 0;
```

```
188                         }
189                     }
190                 }
191
192                 // Error: opcion no reconocida. Imprimimos un
                        mensaje
193                 // de error, y finalizamos la ejecucion del
                        programa.
194                 //
195                 cerr << "Unknown option: "
196                      << "-"
197                      << opt
198                      << "."
199                      << endl;
200                 exit(1);
201
202                 // Algunos compiladores se quejan con funciones
                        que
203                 // logicamente no pueden terminar, y que no
                        devuelven
204                 // un valor en esta ultima parte.
205                 //
206                 return -1;
207 }
```

CODE::option

```
1 #ifndef OPTIONS_HPP_INCLUDED
2 #define OPTIONS_HPP_INCLUDED
3 #include<iostream>
4 #include<string>
5 #include <fstream>
6 #include<cstdlib>
7 using namespace std;
8 //HOLA
9 void opt_input(string const &);
10 void opt_output(string const &);
11 void opt_help(string const &);
12
13 #endif // OPTIONS_HPP_INCLUDED
```

```
1
2 #include "options.hpp"
3 using namespace std;
4 //HOLA
5 extern istream *iss;
6 extern ostream *oss;
7 extern fstream ifs;
8 extern fstream ofs;
9 //extern option_t options[];
```

```
10
11
12 void
13 opt_input(string const &arg)
14 {
15         // Si el nombre del archivos es "-", usaremos la
               entrada
16         // estandar. De lo contrario, abrimos un archivo
               en modo
17         // de lectura.
18         //
19         if (arg == "-") {
20                 iss = &cin;
21
22 //------meter las funciones de charle para entrada standar
23
24
25
26
27
28         } else {
29                 ifs.open(arg.c_str(), ios::in);
30                 iss = &ifs;
31         }
32
33         // Verificamos que el stream este OK.
34         //
35         if (!iss->good()) {
36                 cerr << "cannot open "
37                      << arg
38                      << "."
39                      << endl;
40                 exit(1);
41         }
42 }
43 void
44 opt_output(string const &arg)
45 {
46         // Si el nombre del archivos es "-", usaremos la
               salida
47         // estandar. De lo contrario, abrimos un archivo
               en modo
48         // de escritura.
49         //
50         if (arg == "-") {
51                 oss = &cout;
52
53
54
55 //------meter las funciones de charle para salida standar
```

```
56
57
58
59
60
61          } else {
62                  ofs.open(arg.c_str(), ios::out);
63                  oss = &ofs;
64          }
65
66          // Verificamos que el stream este OK.
67          //
68          if (!oss->good()) {
69                  cerr << "cannot open "
70                       << arg
71                       << "."
72                       << endl;
73                  exit(1);
74          }
75  }
76
77  void opt_help(string const &arg)
78  {
79          cout << "cmdline -f factor [-i file] [-o file]"
80               << endl;
81          exit(0);
82  }
```

CODE::process

```
1 #ifndef PROCESS_HPP_INCLUDED
2 #define PROCESS_HPP_INCLUDED
3
4 #include <string>
5 #include <sstream>
6 #include"common.hpp"
7 #include <vector>
8 #include "NetworkElementClass.hpp"
9
10 //Esta funcion encuentra el ROOT
11
12 int FindRoot(vector <NetworkElement>&);
13
14
15
16 #endif // PROCESS_HPP_INCLUDED
```

```
1 #include"process.hpp"
2 #include "printers.hpp"
3
```

```
4
5 /*
    ************************************************************************

6  FindRoot:La funcion recibe el arreglo de objetos
       NetworkElement y retorna la posicion
7  del ROOT
8
9 ************************************************************************
    */
10
11 int FindRoot(vector <NetworkElement> &v){
12         int i;
13         for(i=0;v.data()[i].getName()!="Hub1";i++);
14         return i;
15 }
```