Trabajo Práctico 0:
Programación C++
75.04 - Algoritmos y Programación II
Universidad de Buenos Aires
Facultad de Ingenieria
1er Cuatrimestre 2014

Carlos Germán Carreño Romano 90392
Cristian Aranda Zózimo Cordero 93631,
Federico Verstraeten 92892

April 10, 2014

Contents

0.1 Introducción

El desarrollo de este trabajo práctico tiene como objetivo ejercitar conceptos básicos de programación en C++. Para el mismo, la temática propuesta propone estructuras de datos basadas en modelos de redes HFC, que son ampliamente utilizadas en sistemas de telecomunicaciones, teniendo por tanto implicancia directa en el consumo masivo. En particular, las redes HFC son estándar en la transmisión de CATV e Internet.

0.2 Programa

El programa propuesto consiste básicamente en estructurar datos, aprender el manejo de flujos(stremas) en C++, el uso de consola o línea de comandos. El formato determinado para los archivos de entrada describe una red HFC a partir de un nombre (nomenclado como *NetworkName*), y los elementos de red y conexiones(nomenclados como *NetworkElement* y *Connection* respectivamente), bajo la estructura de un esquema de árbol donde la jerarquía de elementos se describe en el enunciado. A partir del archivo recibido, que hace la descripción mencionada en formato de texto, se debe generar un archivo de salida computando la información existente, respecto al nombre de red y la cantidad de elementos y conexiones.

Para el desarrollo del programa, se optó por trabajar en módulos de software que puedan interactuar entre sí, con una previa discusión de las condiciones de borde que debería cumplir cada módulo. El esquema de desarrollo se centró entonces en 3 líneas; por un lado la ejecución y validación de los argumentos de entrada mediante línea de comandos, por otro la apertura del archivo de entrada y construcción de un arreglo de punteros a strings para cargar en memoria el

contenido del archivo[1], y por último, el procesamiento de los datos de entrada para imprimirlos en un flujo de salida.

0.2.1 Entrada

En esta sección se explica la primer parte del programa. La entrada es en principio un archivo de texto que contiene información de red. El archivo de texto de entrada utilizado como ejemplo base es el siguiente:

Code

```
1  NetworkName MyNetwork
2  NetworkElement CM1 CM
3  NetworkElement CM3 CM
4  NetworkElement Hub1 Hub
5  NetworkElement Node2 Node
6  NetworkElement Node1 Node
7  NetworkElement CM4 CM
8  NetworkElement CM2 CM
9  NetworkElement Amp1 Amp
10 Connection CM4 Amp1
11 Connection CM3 Amp1
12 Connection Node1 Hub1
13 Connection CM2 Node2
14 Connection CM1 Node1
15 Connection Amp1 Node1
16 Connection Node2 Hub1
```

Este primer ejemplo contiene la información correcta que se debe recibir.

El objetivo de la primera sección es realizar la lectura de la línea de comandos, procesarla, validarla, y realizar la apertura de flujos de entrada y salida si corresponde. La función que se desarrolló en este punto es la siguiente:

donde los argumentos de entrada son:

y la salida es:

que funciona como....

Las opciones de validación consideradas son... - - - Se optó por esta validación asumiendo que....

Las opciones en caso de que no se ingrese correctamente un archivo de entrada, esto es, ... muestran los siguientes mensajes de error, que se imprimen por el flujo....

En esta etapa del trabajo, esta sección se encuentra en una versión funcional. Se propone mejorar el funcionamiento en etapas posteriores considerando más opciones, y una modularización en base a clases.....

Una vez validados los flujos de entrada y salida, referenciados como file in y file out respectivamente, se carga en memoria la información del archivo abierto (file in) y se deja abierto el flujo de salida para el procesado de la información. Esto se corresponde con las secciones de Carga en Memoria y de Procesado de la Información que se detallan a continuación.

[1] la elección de esta estrategia se explica más adelante

0.2.2 Carga en Memoria

Se optó por cargar en memoria dinámica el texto completo recibido. El procedimiento fue cargar los archivos en un arreglo de punteros a string dinámico, nomenclado como *lines*, un string por línea de texto, con una estrategia de crecimiento geométrico. Esto último hace referencia a un modelo de incremento de memoria dinámica que permite incrementar la memoria para los strings en tiempo de ejecución. La hipótesis que fundamenta esta decisión es que a priori, el archivo de entrada puede contener 3(o menos) ó 300(o más) líneas de información, y que el procesamiento que debe ser línea a línea, si se realiza sobre el flujo de entrada, puede tener un costo elevado en tiempo de ejecución si el archivo es grande. Además, estableciendo un tamaño de asignación inicial (INIT_ CHOP) se puede tener noción de la cantidad de memoria inicial que se deberá requerir para trabajar con un archivo promedio. Estas suposiciones se reflejan en dos funciones con los siguientes prototipos:

La función *load file memory()* recibe como primer argumento una referencia a un archivo previamente abierto por el flujo if stream; como segundo argumento la dirección de memoria de un arreglo de strings, que se decidió para devolver por referencia la variable que contiene el arreglo dinámico de strings (*lines*); y como tercer argumento, devuelve por referencia el tamaño del arreglo (size) que indica el número de líneas que contiene el texto.

0.2.3 Proceso

El formato del archivo de texto indicado en la sección Entrada, contiene en la primer línea el nombre de la red, y a partir de la segunda línea hasta el final, contiene la información de los elementos de red (NetworkElement) y de las conexiones entre ellos (Connections). Este es el tipo de formato esperado como flujo de entrada, y en base a éste se pensó y codificó una primer versión que asume que el texto está bien redactado, es decir, empieza con la palabra NetworkName y sigue con las líneas que comienzan con las palabras NetworkElement o Connection. En el caso de NetworkElement, también se asumió que la linea está bien redactada y se leen 3 palabras, siendo la última el tipo de elemento de red a contabilizar. Una línea 'bien redactada' consta de 3 palabras y tiene la siguiente forma:

```
NetworkElement <name> <NetworkElementType>
```

donde la segunda palabra es el nombre del elemento de red, y la tercer palabra es el tipo de elemento de red. Ésta última es la que se requiere para computar la cantidad de elementos distintos que forman la red. Para el caso que se lea Connection, la línea bien redactada tiene la forma:

```
Connection <name1> <name2>
```

y en este caso sólo se computa que hay una conexión. Las función que se codificó para esta sección es la siguiente:

```
void processLine(string textline);
```

que recibe como argumento de entrada el string que corresponde a cada línea de texto del flujo file in. Esta función, convierte el string en un stream y utiliza el operador >> sobrecargado, por medio de la biblioteca $< sstream >$, para compararlo contra dos diccionarios que contienen las palabras claves que se requieren identificar. Estos diccionarios se detallan a continuación:

```
 string network_struct[] = "NetworkName","NetworkElement","Connection";
string network_element_type[] = "Hub","Node","Amp","CM";
int number_of_elements[5];
// = "Number_of_Hubs","Number_of_Nodes", "Number_of_Amps","Number_of_CM",
"Number_of_Connections";
```

La última declaración corresponde a un arreglo de variables int que guardan las cantidades de elementos de red, como indica el comentario. Con este arreglo actualizado al procesar las líneas de texto con la función processLine() sólo resta imprimir por flujo de salida los datos. Este trabajo se explica en la sección de Salida.

Mejoras

Adicionalmente al cómputo de elementos de red y de conexiones, se validó la cantidad de líneas que tiene el archivo al cargarlo en memoria en un arreglo de strings. Con este dato se pretende en una segunda versión, hacer un recorrido de líneas y elaborar un algoritmo de recorrido del arreglo de strings para tener mayor control de información. Además, se pretende vincular los nombres y la cantidad de elementos distintos con una lógica que permita asociar la jerarquía de elementos de red para validar que no haya por ejemplo, un número distinto de las conexiones posibles entre elementos.
Un punto clave a mejorar en esta sección se corresponde también con independizar la función de los diccionarios globales y de la variable global Number of elements, que no fueron utilizados como argumentos.

0.2.4 Salida

En esta sección se dispone de: el flujo de entrada cargado en memoria dinámica; el procesado de la información listo; y el flujo de salida abierto. Por lo que resta es imprimir la información guardada del proceso en el flujo de salida. Para realizar esto se codificaron dos funciones de impresión, de las cuales se detallan los prototipos a continuación:

```
void printNetworkName(string name_line, ostream& os);
void printElements(int number_of_elements[], ostream& os);
```

La primera imprime el nombre de la red y la segunda las líneas que contienen la información requerida en el enunciado a través del flujo de salida os. Un ejemplo de impresión de estas funciones que se corresponde con la información del archivo Networking.txt detallado en la sección Entrada, se muestra a continuación:

Code

```
MyNetwork
1 Hubs
2 Nodes
1 Amps
4 CMs
7 Connections
```

Este es el formato de salida esperado.
En la sección que continúa se muestran resultados de ejecución para distintas entradas.

0.3 Ejecuciones del programa y resultados

0.4 Código fuente

Se detalla a continuación el esquema de codificación, presentando primero el Makefile utilizado y luego cada archivo componente del programa principal.

Makefile

```
#compilacion y ejecucion de los archivos
CC       =  g++
CFLAGS   =  -Wall -pedantic
OBJS=main_muestra.o file_load.o


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

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

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

clean:
        rm *.o
```

main

```
/**** BIBLIOTECAS ****/
#include <iostream>
#include <fstream>
#include <string>
#include"file_load.hpp"
#include"common.hpp"

/**** DEFINICIONES ****/
```

```
9  #define SIG_ARG_POS 1
10 #define CHAR_VOID 0
11
12 using namespace std;
13 #include <sstream>
14
15
16
17 #define MAX_LINES_DEFAULT 100
18
19
20
21 //Prototipos
22
23 void uploadLine(string, string **);
24
25 void printLine(string);
26
27 void processLine(string);
28
29 void printElements(int *, ostream&);
30 void printNetworkName(string, ostream&);
31 //int  searchNetworkName(string**);
32
33
34 //Diccionarios: HAY QUE ACORDAR UNA NOMNECLATURA PARA
       STRINGS , VARIABLES y ARREGLOS.
35 string network_struct[]    = {"NetworkName","
       NetworkElement","Connection"};
36
37 string network_element_type[] = {"Hub","Node","Amp","CM"
       };
38 string network_element_nam[MAX_LINES_DEFAULT];
39
40
41 int number_of_elements[5];// = {"Number_of_Hubs","
       Number_of_Nodes", "Number_of_Amps","Number_of_CM", "
       Number_of_Connections"};
42
43 static size_t    n=0, i=0;
44
45
46 /**** PROTOTIPOS ****/
47
48 status_t read_argument(const char  arg[]);
49 status_t route_verification(char arg[],char** route);
50 void close_all_stream_file(ifstream& ,ofstream& );
51 int validateArgument(int argc,char *argv[],char* &
       route_in,char* &route_out);
52
```

```
53 /**** MAIN ****/
54
55 int main(int argc,char *argv[])
56 {
57     char *route_in=NULL, *route_out=NULL;
58     string **lines;
59     size_t number_lines;
60     size_t i;
61     int f_;
62
63     f_=validateArgument(argc,argv,route_in,route_out);
64
65     if(f_==1)//route_in!=NULL && route_out!=NULL
66     {
67         ifstream file_in (route_in); //Flujo archivo
              file_in, abierto
68
69         ofstream file_out (route_out,ios_base::out);//
              Flujo archivo file_out, abierto
70
71        load_file_memory(file_in,&lines,number_lines);
72         cout<< "Funciono el hibrido wachin jaja alto
              programa loro ahh y tenes " <<number_lines <<
              endl;
73        /* for(i=0;i<number_lines;i++){
74                file_out<<*lines[i]<<endl;
75         }
76         */
77         printNetworkName((*lines[0]),file_out); // esta
              tiene el contenido del nombre de la red
78         for(i=1;i<number_lines;i++)     processLine((*
              lines[i]));
79
80         printElements(number_of_elements, file_out);
              //esta funcion printElements() es la que se
              encarga de la salida por pantalla.
81
82          close_all_stream_file(file_in,file_out);
83
84     }
85     else
86     {
87         cout<<"Error: no se pudieron abrir los stream
              correctamente"<<endl;
88         if(route_in==NULL)
89             cout<<"Error: stream entrada falso"<<endl;
90         if(route_out==NULL)
91             cout<<"Error: stream salida falso"<<endl;
92
93
```

```
94      }
95          erase_file_memory(&lines,number_lines);
96
97          return 0;
98
99  }
100
101
102 /**** FUNCIONES ****/
103
104 status_t read_argument(char const arg[])
105 {
106
107                 // Todos los parametros de este programa
                        deben
108                 // pasarse en forma de opciones.
                        Encontrar un
109                 // parametro no-opcion es un error.
110                 //
111                 if (arg[0]== '-')
112                 {
113             if(arg[1]=='i' && arg[2]==CHAR_VOID) //Con
                    CHAR_VOID verifico que no se ingresen
                    cosas como "-increible" y lo
114                  return OK_INPUT;            //valide, es
                        decir el caracter seguid de "-i" sea
                        vacio o blanco.
115                 //cout<<"entrada\n";
116             else if(arg[1]=='o' && arg[2]==CHAR_VOID)
117                  return OK_OUTPUT;
118                  //cout<<"salida\n";
119
120                 }
121
122
123     return ARG_ERR;
124 }
125
126 status_t route_verification(char arg[],char** route)
127 {
128     if (arg[0]!= '-')
129     {
130         (*route)=arg; //Dejo apuntando route al string
                que posee el nombre de la ruta
131         return ARG_OK;
132     }
133
134     return ARG_ERR;
135 }
136
```

```
137 void close_all_stream_file(ifstream &file_in , ofstream &
    file_out)
138 {
139     file_in.close();
140     file_out.close();
141 }
142
143
144
145 void uploadLine(string line , string** lines_array)
146 {
147
148             lines_array[n]=new string;
149
150             (*lines_array[n])=line;
151
152             printLine((*lines_array[n]));
153
154             n++;
155
156 }
157
158
159
160
161
162 void processLine(string text_line)
163 {
164
165     istringstream iss(text_line);
166     string word[3];
167
168     iss >> word[0];
169     if(word[0]==network_struct[1])
170     {
171         iss >> word[1];
172         iss >> word[2];
173         size_t i;
174         for(i=0;i<4;i++)
175         {
176          if(word[2]==network_element_type[i])
                number_of_elements[i]++;
177         }
178     }
179     if(word[0]==network_struct[2])    number_of_elements
        [4]++;
180
181
182 }
183
```

```
184
185 void printLine(string line)
186 {
187
188     cout << line << "\n";
189
190 }
191 /*
192 int  searchNetworkName(string* lines_array)
193 {
194
195 }*/
196
197 void printNetworkName(string name_line, ostream& os)
198 //esta funcion asume que se le pasa un string
199 //que contiene "NetworkName <nombre>",
200 //donde nombre es el nombre a imprimir
201 {
202     string aux, network_name;
203     istringstream iss(name_line);
204     iss >> aux;
205     if(aux == network_struct[0])
206     {
207         iss >> network_name;
208         os << network_name << "\n";
209     }
210     else      cout << "no hay nombre de red" << "\n";
211 }
212
213
214 void printElements(int number_of_elements[], ostream& os)
215 {
216
217         os << number_of_elements[0] << " Hubs"<<"\n";
218
219         os << number_of_elements[1] << " Nodes"<<"\n";
220
221         os << number_of_elements[2] << " Amps"<<"\n";
222
223         os << number_of_elements[3] << " CMs"<<"\n";
224         os << number_of_elements[4] << " Connections"<<"\
            n";
225
226 }
227
228 int validateArgument(int argc,char *argv[],char* &
    route_in,char* &route_out)
229 {
230         if(argc==1)
231         {
```

```
232                 cout<<"Error: no hay argumentos"<<endl;
233                 return 0;
234         }
235         for (int i = 1; i < argc; ++i)
236         {
237                 if(read_argument(argv[i])==OK_INPUT && (i
                        +1)!=argc)
238                 {
239
240                 if(route_verification(argv[i+1],&route_in
                        )!=ARG_ERR) i = i + SIG_ARG_POS;
241                 //El argumento siguiente debe contener la
                         ruta del archivo, lo valido y lo
                        apunto con la varuiable route
242
243                 else
244                 {
245                     cerr << "Ruta entrada invalida: "
246                          << argv[i+1]
247                          << endl;
248                      //break;
249                      return 0;
250                  }
251
252         }
253
254         else if(read_argument(argv[i])==OK_OUTPUT && (i
                +1)!=argc)
255         {
256             if(route_verification(argv[i+1],&route_out)!=
                    ARG_ERR) i = i + SIG_ARG_POS;
257
258             else
259             {
260                 cerr << "Ruta salida invalida: "
261                  << argv[i+1]
262                  << endl;
263                 //break;
264                 return 0;
265             }
266
267         }
268
269         else //if (read_argument(argv[i])==ARG_ERR)
270         {
271             cerr << "Argumento invalido: "
272                             << argv[i]
273                             << endl;
274             //break;
275             return 0;
```

```
276             }
277
278         }
279
280         return 1;
281     }
```

headers

```
1  #ifndef COMMON_HPP
2  #define COMMON_HPP
3  using namespace std;
4  typedef enum{OK,ERROR_NULL_POINTER,
       ERROR_MEMORY_NO_AVAILABLE}status_t;
5
6
7  #endif
```

```
1  #ifndef DICTIONARY_HPP_INCLUDED
2  #define DICTIONARY_HPP_INCLUDED
3
4  string network_struct[]    = {"NetworkName","
       NetworkElement","Connection"};
5  string network_element_type[] = {"Hub","Node","Amp","CM"
       };
6  string network_element_name[MAX_LINES_DEFAULT];
7  int number_of_elements[5];// = {"Number_of_Hubs","
       Number_of_Nodes", "Number_of_Amps","Number_of_CM", "
       Number_of_Connections"};
8
9  #endif // DICTIONARY_HPP_INCLUDED
```

```
1  #ifndef FILE_LOAD_HPP
2  #define FILE_LOAD_HPP
3
4  #include<iostream>
5  #include<fstream>
6  #include<string>
7  #include"common.hpp"
8  #define INIT_CHOP 30
9  #define CHOP_SIZE 20
10 //Prototipos
11 status_t loadFileMemory(ifstream &file,string ***lines,
       size_t &size);
12 status_t eraseFileMemory(string ***lines,size_t &size);
13 #endif
```

```
1  #ifndef PRINTERS_HPP_INCLUDED
2  #define PRINTERS_HPP_INCLUDED
3
```

```
4  void printElements(int *, ostream&);
5  void printNetworkName(string , ostream&);
6
7  #endif // PRINTERS_HPP_INCLUDED
```

```
1  #ifndef PROCESS_HPP_INCLUDED
2  #define PROCESS_HPP_INCLUDED
3
4  void processLine(string);
5
6  #endif // PROCESS_HPP_INCLUDED
```

codes

```
1  #include"file_load.hpp"
2  #include"common.hpp"
3
4  using namespace std;
5
6  //Especificaciones de la funcion:
7  //le pasan una referencia al archivo ya abierto
       previamente.
8  //devuelve por interfaz un arreglo de punteros a objeto
       string cargados con el archivo file(una linea por
       string).
9  //devuelve la cantindad de lineas leidas.
10
11 status_t loadFileMemory(ifstream &file,string ***lines,
       size_t &size){
12
13         size_t alloc_size,i;
14         string **aux,str;
15
16         if(lines==NULL)return ERROR_NULL_POINTER;
17         if(((*lines)=new string*[INIT_CHOP])==NULL)return
               ERROR_MEMORY_NO_AVAILABLE;
18
19         size=0;
20         alloc_size=INIT_CHOP;
21
22         while(getline(file,str)){
23
24                 if(size==alloc_size){
25
26                         if((aux=new string*[CHOP_SIZE+
                               alloc_size])==NULL){
27
28                                 for(i=0;i<size;i++){
29                                         delete(*lines)[i
                                               ];
```

```
30                                      }
31                                      delete[](*lines);
32                                      return
                                            ERROR_MEMORY_NO_AVAILABLE
                                            ;
33                              }
34                              alloc_size+=CHOP_SIZE;
35                              for(i=0;i<size;i++){
36
37                                      aux[i]=(*lines)[i];
38                              }
39                              delete []((*lines);//elimino el
                                    viejo arreglo
40                              (*lines)=aux;
41
42                      }
43
44                      (*lines)[size++]=new string(str);
45              }
46      return OK;
47      }
48
49      //Especificaciones de funcion.
50      //Borra los string y luego el arreglo de punteros  a
            string.
51      status_t eraseFileMemory(string ***lines,size_t &size){
52
53              size_t i;
54
55              if(lines==NULL)return ERROR_NULL_POINTER;
56              for(i=0;i<size;i++){
57                      delete(*lines)[i];
58              }
59              delete[](*lines);
60              *lines=NULL;
61      return OK;
62      }
```

```
1   void printNetworkName(string name_line, ostream& os)
2   //esta funcion asume que se le pasa un string
3   //que contiene "NetworkName <nombre>",
4   //donde nombre es el nombre a imprimir
5   {
6       string aux, network_name;
7       istringstream iss(name_line);
8       iss >> aux;
9       if(aux == network_struct[0])
10      {
11          iss >> network_name;
12          os << network_name << "\n";
```

```
13      }
14      else      os << "no hay nombre de red" << "\n";
15  }
16
17
18
19  void printElements(int number_of_elements[], ostream& os)
20  {
21      os << number_of_elements[0] << " Hubs"<<"\n";
22      os << number_of_elements[1] << " Nodes"<<"\n";
23      os << number_of_elements[2] << " Amps"<<"\n";
24      os << number_of_elements[3] << " CMs"<<"\n";
25      os << number_of_elements[4] << " Connections"<<"\n";
26  }
```

```
1   //Esta funcion recibe una linea del texto
2   //Networking asumiendo que viene 'bien escrita',
3   //esto es, que siempre que viene Networking
4   // vienen 2 palabras mas, y la ultima es
5   // la que se necesita computar. Si viene connection
6   // entonces solo se incrementa la cantidad de conexiones.
7
8   void processLine(string text_line)
9   {
10      istringstream iss(text_line);
11      string word[3];
12      iss >> word[0];
13      if(word[0]==network_struct[1])//network_struct[] es
            un diccionario global
14      {
15          iss >> word[1];
16          iss >> word[2];
17          size_t i;
18          for(i=0;i<4;i++)
19          {
20           if(word[2]==network_element_type[i])
                number_of_elements[i]++;
21          }//number_of_elements[] es un array global que
                guarda las cantidades
22      }
23      if(word[0]==network_struct[2])    number_of_elements
            [4]++;
24
25  }
```

0.5 Conclusiones