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

May 8, 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

```
NetworkName MyNetwork
NetworkElement CM1 CM
NetworkElement CM3 CM
NetworkElement Hub1 Hub
NetworkElement Node2 Node
NetworkElement Node1 Node
NetworkElement CM4 CM
NetworkElement CM2 CM
NetworkElement Amp1 Amp
Connection CM4 Amp1
Connection CM3 Amp1
Connection Node1 Hub1
Connection CM2 Node2
Connection CM1 Node1
Connection Amp1 Node1
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 Testing

El resultado del testing realizado es el siguiente:

Ejecución del programa sin argumentos:

```
.../TP0_final# ./main_muestra_V2.exe
error 9:ERROR_NO_ARGS
```

Ejecución con argumento '-i' seguido de nada:

```
.../TP0_final# ./main_muestra_V2.exe -i
error 8:ERROR_ROUTE_NAME_INVALID
```

Ejecución con argumento '-o' seguido de nada:

```
.../TP0_final# ./main_muestra_V2.exe -o
error 8:ERROR_ROUTE_NAME_INVALID
```

Ejecución con argumento falso o inválido:

```
.../TP0_final# ./main_muestra_V2.exe -increible
error 12:ERROR_INVALID_ARG
```

Ejecución con argumento '-i' seguido de una ruta falsa:

```
.../TP0_final# ./main_muestra_V2.exe -i i
error 8:ERROR_ROUTE_NAME_INVALID
```

Ejecución con argumento '-i' con archivo de entrada y sin argumento de salida:

```
.../TP0_final# ./main_muestra_V2.exe -i Networking.txt
error 8:ERROR_ROUTE_NAME_INVALID
```

Ejecución con argumentos de entrada y salida correctos:

```
.../TP0_final# ./main_muestra_V2.exe -i Networking.txt -o O1.txt
```

Ejecución con entrada '/dev/null':

```
.../TP0_final# ./main_muestra_V2.exe -i /dev/null
error 14:ERROR_NULL_FILE
```

Ejecución de ejemplo con entrada '-i -' para ejecutar desde consola:

```
.../TP0_final# ./main_muestra_V2.exe -i -
I: NetworkName Red_cin
O: Red_cin
Ingrese elementos de red y conexiones. Luego termine con NetEND
I: NetworkElement CM1 CM
I: NetworkElement CM2 CM
I: NetworkElement Amp1 Amp
I: Connection CM1 Amp1
I: Connection CM2 Amp1
I: NetEND
O: 0 Hubs
O: 0 Nodes
O: 1 Amps
O: 2 CMs
O: 2 Connections
```

Para el archivo generado por el script de Python randnet.py, el resultado de la ejecución del programa se generó correctamente:

```
.../TP0_final# ./main_muestra_V2.exe -i randnet -o Out_randnet.txt
.../TP0_final# cat Out_randnet.txt
network_bkOHYTqJnqCKYbFIeVahnoAQQNbBdWfZaQuQounvQZecDfdkOKGPGJeTCTrFBMbbEuhQjXADFwRMljiJMp
13647 Hubs
13805 Nodes
13752 Amps
13756 CMs
11785 Connections
```

Se puede observar que el nombre de red en la primer línea es una cadena de caracteres aleatorios.

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    = -g -Wall  -pedantic
OBJS=main_muestra_V2.o file_load.o printers.o process.o dictionary.o argume



muestras.exe:$(OBJS)
        $(CC)  $(OBJS) -o main_muestra_V2.exe
```

```
main_muestra_V2.o:main_muestra_V2.cpp dictionary.hpp printers.hpp process.h
	$(CC) main_muestra_V2.cpp -c -o main_muestra_V2.o $(CFLAGS)

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

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


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


clean:
	rm *.o
```

main

```
/*****************************************************************************
TP - Algoritmos II - Lic. Calvo
Jueves 8 de Mayo 2014

Docentes:
    Lucio Santi, Leandro Santi
Alumnos:
    Carlos Carreno Romano <charlieromano@gmail.com>
    Federico Verstraeten <federico.verstraeten@gmail.com>
    Cristian Zozimo Aranda Cordero <cristian.pinky@gmail.com>

*****************************************************************************


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

```
#include"arguments.hpp"

int number_of_elements[5];

/**** MAIN ****/


int main(int argc,char *argv[])
{
        char *route_in=NULL, *route_out=NULL;
        string **lines;
        size_t number_lines;
        size_t i;
        status_t f_, status;
        ifstream file_in;


        f_=validateArgument(argc,argv,route_in,route_out);

        if(f_==OK)
        {
                file_in.open( route_in ); //Flujo archivo file_in, abierto
                if (!file_in.is_open())
                {
                        printErrorMessage(ERROR_INVALID_INPUT_ROUTE,cout);
                        return 1;
                }


                ofstream file_out ( route_out , ios_base::out );//Flujo arc
                loadFileMemory( file_in , &lines , number_lines );

                printNetworkName( (*lines[0]) , file_out) ; // esta tiene e
                for( i=1 ; i<number_lines ; i++ )
                        processLine( (*lines[i]) );

                printElements( number_of_elements , file_out );
                close_all_stream_file( file_in , file_out );
                eraseFileMemory( &lines , number_lines );
        }

        if(f_==OK_INPUT_CIN)
        {
        inputFromConsole();
        printElements( number_of_elements , cout );
        }

        else
        {
        printErrorMessage(f_,cout);
```

```
        return ERROR_ARG;
        }
    return 0;
}
```

headers

```
#ifndef COMMON_HPP
#define COMMON_HPP
using namespace std;
typedef enum{
                OK,
                OK_INPUT,
                OK_INPUT_CIN,
                OK_INPUT_CIN_PROCESSING,
                OK_OUTPUT,
                OK_OUTPUT_COUT,
                OK_ROUTE_NAME,
                ERROR_NULL_POINTER,
                ERROR_MEMORY_NO_AVAILABLE,
                ERROR_TEXT_LINE_INVALID,
                ERROR_ARG,
                ERROR_ROUTE_NAME_INVALID,
                ERROR_NO_ARGS,
                ERROR_INVALID_INPUT_ROUTE,
                ERROR_INVALID_OUTPUT_ROUTE,
                ERROR_INVALID_ARGUMENT,
                ERROR_STREAM_OUT,
                ERROR_NULL_FILE
                }status_t;


#endif

#ifndef DICTIONARY_HPP_INCLUDED
#define DICTIONARY_HPP_INCLUDED
#include<string>
#include"common.hpp"
#define MAX_LINES_DEFAULT 100




#endif // DICTIONARY_HPP_INCLUDED

#ifndef FILE_LOAD_HPP
#define FILE_LOAD_HPP
#include<iostream>
```

```
#include<fstream>
#include<string>
#include"common.hpp"
#define INIT_CHOP 30
#define CHOP_SIZE 20
//Prototipos
status_t loadFileMemory(ifstream &file,string ***lines,size_t &size);
status_t eraseFileMemory(string ***lines,size_t &size);
#endif

#ifndef PRINTERS_HPP
#define PRINTERS_HPP
#include"common.hpp"
#include<iostream>
#include <string>
#include <sstream>
#include<fstream>


void printElements(int *, ostream&);
void printNetworkName(string, ostream&);
void printErrorMessage(status_t, ostream&);
void printString(string, ostream&);

#endif // PRINTERS_HPP_INCLUDED

#ifndef PROCESS_HPP_INCLUDED
#define PROCESS_HPP_INCLUDED

#include <string>
#include <sstream>
#include"common.hpp"
void processLine(string);
void inputFromConsole(void);

#endif // PROCESS_HPP_INCLUDED

#ifndef ARGUMENTS_HPP
#define ARGUMENTS_HPP
#include<fstream>
#include<iostream>
#include"common.hpp"
/**** DEFINICIONES ****/
#define SIG_ARG_POS 1
#define CHAR_VOID 0

/**** PROTOTIPOS ****/

status_t read_argument(const char  arg[]);
status_t route_verification(char arg[],char* &route);
```

```
void close_all_stream_file(ifstream& ,ofstream& );
status_t validateArgument(int argc,char *argv[],char* &route_in,char* &rout




#endif
```

includes

```
#include"file_load.hpp"


//Especificaciones de la funcion:
//le pasan una referencia al archivo ya abierto previamente.
//devuelve por interfaz un arreglo de punteros a objeto string cargados con
//devuelve la cantindad de lineas leidas.

status_t loadFileMemory(ifstream &file,string ***lines,size_t &size){

        size_t alloc_size,i;
        string **aux,str;

        if(lines==NULL)return ERROR_NULL_POINTER;
        if(((*lines)=new string*[INIT_CHOP])==NULL)return ERROR_MEMORY_NO_A

        size=0;
        alloc_size=INIT_CHOP;

        while(getline(file,str)){

                if(size==alloc_size){

                        if((aux=new string*[CHOP_SIZE+alloc_size])==NULL){

                                for(i=0;i<size;i++){
                                        delete(*lines)[i];
                                }
                                delete[](*lines);
                                return ERROR_MEMORY_NO_AVAILABLE;
                        }
                        alloc_size+=CHOP_SIZE;
                        for(i=0;i<size;i++){

                                aux[i]=(*lines)[i];
                        }
                        delete [](*lines);//elimino el viejo arreglo
                        (*lines)=aux;
```

```
                }

                (*lines)[size++]=new string(str);
        }
 return OK;
}

//Especificaciones de funcion.
//Borra los string y luego el arreglo de punteros  a string.
status_t eraseFileMemory(string ***lines,size_t &size){

        size_t i;

        if(lines==NULL)return ERROR_NULL_POINTER;
        for(i=0;i<size;i++){
                delete(*lines)[i];
        }
        delete[](*lines);
        *lines=NULL;
return OK;
}

#include"printers.hpp"
extern string network_struct[];
//esta funcion asume que se le pasa un string
//que contiene "NetworkName <nombre>",
//donde nombre es el nombre a imprimir

void printNetworkName(string name_line, ostream& os)
{
    string aux, network_name;
    istringstream iss(name_line);
    iss >> aux;
    if(aux == network_struct[0])
    {
        iss >> network_name;
        os << network_name << "\n";
    }
    else     os << "no hay nombre de red" << "\n";
}



void printElements(int number_of_elements[], ostream& os)
{
    os << number_of_elements[0] << " Hubs"<<"\n";
    os << number_of_elements[1] << " Nodes"<<"\n";
    os << number_of_elements[2] << " Amps"<<"\n";
    os << number_of_elements[3] << " CMs"<<"\n";
    os << number_of_elements[4] << " Connections"<<"\n";
```

```
}

void printErrorMessage(status_t error_type, ostream& os)
{
    switch(error_type){
        case OK:
            break;
        case ERROR_NULL_POINTER:
            { os << "error " << error_type << ":ERROR_NULL_POINTER"<< "\n";
        case ERROR_MEMORY_NO_AVAILABLE:
            { os << "error " << error_type << ":ERROR_MEMORY_NO_AVAILABLE"<
        case OK_INPUT:
            break;
        case OK_OUTPUT:
            break;
        case OK_INPUT_CIN:
            break;
        case OK_OUTPUT_COUT:
            break;
        case ERROR_TEXT_LINE_INVALID:
            { os << "error " << error_type << ":ERROR_TEXT_LINE_INVALID"<<
        case ERROR_ROUTE_NAME_INVALID:
            { os << "error " << error_type << ":ERROR_ROUTE_NAME_INVALID"<<
        case ERROR_ARG:
            { os << "error " << error_type << ":ERROR_ARG"<< "\n"; break; }
        case OK_ROUTE_NAME:
            break;
        case ERROR_NO_ARGS:
            { os << "error " << error_type << ":ERROR_NO_ARGS"<< "\n"; brea
        case ERROR_INVALID_INPUT_ROUTE:
            { os << "error " << error_type << ":ERROR_INVALID_INPUT_ROUTE"<
        case ERROR_INVALID_OUTPUT_ROUTE:
            { os << "error " << error_type << ":ERROR_INVALID_OUTPUT_ROUTE"
        case ERROR_INVALID_ARGUMENT:
            { os << "error " << error_type << ":ERROR_INVALID_ARG"<< "\n";
        case ERROR_STREAM_OUT:
            { os << "error " << error_type << ":ERROR_STREAM_OUT"<< "\n"; b
        case ERROR_NULL_FILE:
            { os << "error " << error_type << ":ERROR_NULL_FILE"<< "\n"; br
        case OK_INPUT_CIN_PROCESSING:
            break;


        default:break;
    }
}
void printString(string s, ostream& os){    os << s << "\n"; }

#include"process.hpp"
#include "printers.hpp"
```

```
extern string network_struct[];
extern string network_element_type[];
extern string network_element_name[];
extern int number_of_elements[];// = {"Number_of_Hubs","Number_of_Nodes", "

//Esta funcion recibe una linea del texto
//Networking asumiendo que viene 'bien escrita',
//esto es, que siempre que viene Networking
// vienen 2 palabras mas, y la ultima es
// la que se necesita computar. Si recibe Connection
// entonces solo se incrementa la cantidad de conexiones.

void processLine(string text_line)
{
    istringstream iss(text_line);
    string word[3];
    iss >> word[0];
    if(word[0]==network_struct[1])//network_struct[] es un diccionario glob
    {
        iss >> word[1];
        iss >> word[2];
        size_t i;
        for(i=0;i<4;i++)
        {
         if(word[2]==network_element_type[i]) number_of_elements[i]++;
        }//number_of_elements[] es un array global que guarda las cantidade
    }
    if(word[0]==network_struct[2])   number_of_elements[4]++;
    else if( (word[0]!=network_struct[1]) && (word[0]!=network_struct[2]) )
    {
        status_t error_process=ERROR_TEXT_LINE_INVALID;
        printErrorMessage(error_process, cout);
        printString(word[0], cout);
    }
}

void inputFromConsole(void)
{
        string line, NetName;

        getline(cin,NetName);
        printNetworkName(NetName,cout);
        cout<<"Ingrese␣elementos␣de␣red␣y␣conexiones.␣Luego␣termine␣con␣Net
        while(getline(cin, line))
        {
            if(line=="NetEND") break;
            processLine(line);

        }
```

```
}

#include"arguments.hpp"
#include"common.hpp"

/**** FUNCIONES ****/

status_t read_argument(char const arg[])
{

                /*  Todos los parametros de este programa deben
            pasarse en forma de opciones, con el prefijo '-'.
            Encontrar un
            parametro no-opcion es un error.
                */

                if (arg[0]== '-')
                {
            if(arg[1]=='i' && arg[2]==CHAR_VOID)
                        //Con CHAR_VOID verifico que no se ingresen cosas c
                        //valide, es decir el caracter seguid de "-i" sea v
                return OK_INPUT;

            else if(arg[1]=='o' && arg[2]==CHAR_VOID)
                return OK_OUTPUT;


                }

    return ERROR_INVALID_ARGUMENT; //Consideramos que deberia ir ERROR_INVA
}

status_t route_verification(char arg[],char* &route)
{
    if (string(arg)=="/dev/null")
        // Valida como caso particular que la entrada sea /dev/null
    {
        return ERROR_NULL_FILE;
    }
    if ( string(arg)=="-")
        return OK_INPUT_CIN;
        else route=arg;

    return OK_ROUTE_NAME;
}

void close_all_stream_file(ifstream &file_in, ofstream &file_out)
{
    file_in.close();
```

```
    file_out.close();
}


status_t validateArgument(int argc,char *argv[],char* &route_in,char* &rout
{
        if(argc==1) return ERROR_NO_ARGS;
        //Salvo caso en que no se ingresen argumentos, ejemplo ./programa.e

    status_t status;


        for (int i = 1; i < argc; ++i)
        {

                status=read_argument(argv[i]);

        if((status!=OK_INPUT) && (status!=OK_OUTPUT)) return status;
                //validacion contra '-i' y '-o', salva por ejemplo '-increi

                if(status==OK_INPUT && (i+1)!=argc)
        {
                        status=route_verification(argv[i+1],route_in);
            if(status!=OK_ROUTE_NAME) return status;

                        else i+= SIG_ARG_POS;
            //El argumento siguiente debe contener la ruta del archivo, lo
        }

        else if(status==OK_OUTPUT && (i+1)!=argc)
        {
                        status=route_verification(argv[i+1],route_out);
            if(status!=OK_ROUTE_NAME) return status;

                        else i+= SIG_ARG_POS;
        }


    }
        // Si el ciclo finaliza correctamente, la verificacion de los argum

    if(route_in!=NULL && route_out!=NULL)return OK; // Esta linea VERY IMPO




//-------------------------------------------------------------------------

                /*antes de salir se puede validar -i -*/
```

```
/*
struct noop {
    void operator()(...) const {}
};

// ...

shared_ptr<istream> input;
if (filename == "-")
    input.reset(&cin, noop());
else
    input.reset(new ifstream(filename.c_str()));
*/

//------------------------------------------------------------------------


        return ERROR_ROUTE_NAME_INVALID;
}
```