Algoritmos y programación II (95.12)

Integrantes:

Freites, Lucas # 96138

Mancini, Lucas # 98091

Rizzo, Gonzalo Gabriel # 96772

|  |  |
| --- | --- |
| Entrega/Fecha | 20/10/16 |
| Nota |  |

**Introducción**

En el presente trabajo se busca ejercitar los conceptos básicos de programación en C++ asentando los mismos en un programa. El programa en cuestión consiste en un software capaz de procesar señales moduladas en amplitud de acuerdo al flujo de cómputo que figura a continuación:

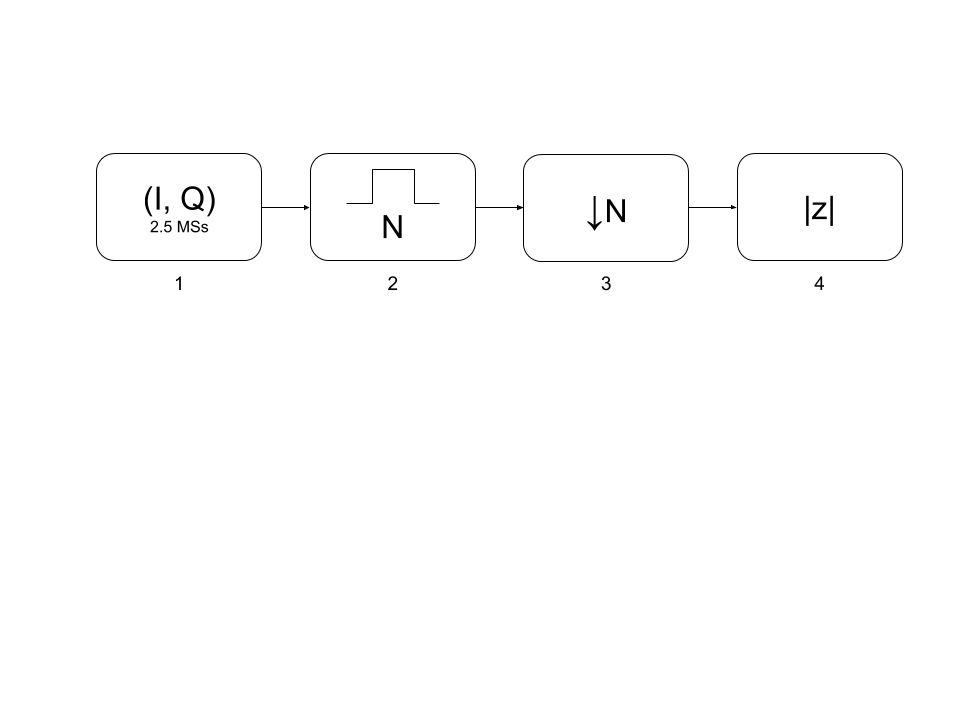


Figura 1: Diagrama de flujo de procesamiento de señales de RF

Donde a los bloques operativos les corresponden las siguientes acciones:

1. El programa recibe un stream de números complejos (I, Q) que digitalizan una porción de 2.5 MHz del espectro de RF, centrado en la frecuencia de la estación de radio que queremos demodular
2. Moving avarage: Promedia las últimas N muestras recibidas en cada instante de tiempo
3. Decimador: Reduce la velocidad de muestreo de señal de entrada, seleccionando 1 de cada N muestras adyacentes
4. Módulo: Calcula el valor absoluto de la entrada compleja

**Desarrollo**

1. **Invocación**

Como se especifica en la guía de trabajo, la interacción con el programa se realiza mediante línea de comandos. A continuación se especifica las opciones disponibles a incluir durante su invocación en la línea de comandos y las respuestas del programa ante los mismos. En el caso que las opciones no sean especificadas, o a continuación de las mismas se reciba “-”, las opciones (en el caso que se indique) toman su valor por defecto:

* Entrada:
  + Flag: “-i”.
  + Por defecto se tomará como entrada el flujo estándar *cin.* El ingreso de datos se detiene al encontrar el EOF (*“end of file”.* Linux: Ctrl+D. Win: Ctrl+Z)
  + A continuación de esta bandera, se especifica el nombre del archivo donde se hará la recepción de datos que serán procesados por el programa.
  + Los requisitos que debe cumplir el archivo de entrada son:
    - Debe ser un archivo plano en formato *txt*.
    - Los números complejos deben estar en el formato de par ordenado, o en el caso de solo poseer la parte real, un número. Los mismos deben estar separados unos de otros por espacios en blanco (tabulación, espacio o salto de línea).
    - No se permiten caracteres. En ese caso se informará en el archivo histórico la ocurrencia de los mismos.
    - No se permiten expresiones matemáticas.
* Salida:
  + Flag: “-o”.
  + Por defecto se tomará como entrada el flujo estándar *cout.*
  + A continuación de esta bandera, se especifica el nombre del archivo donde se despliega el resultado final del programa.
  + En el caso de que el archivo exista previamente, se incluyen los nuevos datos a continuación.
* Longitud del filtro pasabajos:
  + Flag: “-N”
  + Por defecto de N=500.
  + A continuación de esta bandera, nos permite configurar la longitud del filtro pasabajos y el factor decimación.
  + Los requisitos que debe cumplir el valor de *N* son:
    - Debe ser un entero positivo. En caso de no serlo, se despliega por pantalla un mensaje de error y se termina el programa.
    - No puede se permiten expresiones matemáticas.
* Archivo histórico para errores:
  + Flag: “-log”
  + Por defecto se crea un archivo histórico en el directorio donde se ejecuta el programa.
  + A continuación de esta bandera, se especifica el nombre del archivo donde se quisiera informar sobre la terminación y/o el estado del programa. En el archivo se incluirá fecha y hora de la ejecución del programa y, en caso de surgir un evento inesperado, se informará del mismo. Los errores previstos se especifican en la tabla **1.**
  + En el caso que el archivo exista previamente, se sobreescribe el mismo.

|  |  |
| --- | --- |
| **Evento** | **Respuesta del programa** |
| Error de escritura/lectura en stream | -Terminación del programa.  -Especificación del error (falla en lectura o escritura) en archivo log. |
| Error en la lectura de datos | -Terminación del programa.  -Especificación de número de operación fallida en archivo log. |
| Error en cierre de archivo de lectura (ausencia de EOF) | -Terminación del programa.  -Especificación del error en archivo log. |
| Terminación exitosa del programa | -Mensaje de proceso exitoso en archivo log. |

Tabla 1: Especificaciones de errores previstos y respuesta del programa

* Ayuda:
  + Flag: “-h”.
  + Es una opción que indica cómo es la correcta invocación del programa, especificando el nombre del mismo, seguido de las opciones posibles. A continuación de las mismas se indica de qué tipo es el argumento que las mismas reciben.

Un ejemplo de la invocación del programa sería:

$/tp0 -i input.txt -o output.txt -N 300 -log log.txt

**B. Procesamiento**

La función principal del programa, *main*, consta de 3 procesos fundamentales:

* *cmdline cmdl(options);*

Se crea un objeto del tipo *cmdline* para manejar los argumentos obtenidos en la invocación del programa. El mismo se inicializa con la tabla *options* del tipo *options\_t* declarada globalmente, la cuál contiene un arreglo con opciones estandarizadas.

* *cmdl.parse(argc,argv);*

Se instancia el objeto *cmdl* para que se parseen los respectivos argumentos con los que se invocó el programa.

* *RF\_process(iss,oss);*

En esta función, se realiza el procesamiento de los números complejos. A la hora de diseñar el algoritmo que procesaría los datos, surgieron varias implementaciones:

1. Se toman los números complejos de la entrada y cada N (factor decimación) muestras, se toma el promedio de las N muestras anteriores. Seguidamente, de ese promedio calculado, se toma el módulo del mismo y se despliega el resultado en el archivo de salida. En este caso, los bloques decimador y promediador se realizan en una sola acción.
2. Se toman los números complejos de la entrada mientras se llena un arreglo de N posiciones con las correspondientes muestras. Se toma el promedio de todas las muestras en el arreglo. Luego se toma un valor al azar del arreglo, se le calcula el promedio y se despliega el resultado en el archivo de salida. Luego se vuelve a cargar el arreglo con las N muestras siguientes en el archivo.
3. Se realiza las mismas acciones que en el caso 2. pero en vez que el arreglo se renueve por completo, solamente se renueva con la siguiente muestra en el archivo de entrada y se elimina el primer dato que entró en el arreglo en la leída anterior. De esta forma el arreglo es una estructura de dato del tipo “cola” ya que cumple con la estructura *FIFO (First In, First Out*).

Finalmente se optó por la implementación 1. debido a que consideramos que la misma era la más eficiente en cuestión de velocidad de procesamiento de las muestras. La misma no requiere el uso de memoria excesiva al almacenar resultados de operaciones calculadas innecesariamente que luego serían descartadas en la etapa de decimación como ocurre en el caso 2 y 3, donde calcularíamos una serie de promedios que serán perdidos.

**Documentación**

A Continuación se presentan en forma impresa todos los scripts utilizados para el trabajo:

***main.cc:***

#include <fstream>

#include <iomanip>

#include <iostream>

#include <sstream>

#include <cstdlib>

#include <cmdline.h>

#include "complejo.h"

#include "log\_msg.h"

#include "log\_printer.cc"

#define DEFAULT\_N 500

using namespace std;

static void opt\_input(string const &);

static void opt\_output(string const &);

static void opt\_samples\_length(string const &);

static void opt\_log\_file(string const &);

static void opt\_help(string const &);

void RF\_process(istream \*, ostream \*);

// Tabla de opciones de línea de comando. El formato de la tabla

// consta de un elemento por cada opción a definir. A su vez, en

// cada entrada de la tabla tendremos:

//

// o La primera columna indica si la opción lleva (1) o no (0) un

// argumento adicional.

//

// o La segunda columna representa el nombre corto de la opción.

//

// o Similarmente, la tercera columna determina el nombre largo.

//

// o La cuarta columna contiene el valor por defecto a asignarle

// a esta opción en caso que no está explícitamente presente

// en la línea de comandos del programa. Si la opción no tiene

// argumento (primera columna nula), todo esto no tiene efecto.

//

// o La quinta columna apunta al método de parseo de la opción,

// cuyo prototipo debe ser siempre void (\*m)(string const &arg);

//

// o La ultima columna sirve para especificar el comportamiento a

// adoptar en el momento de procesar esta opción: cuando la

// opción es obligatoria, deberá activarse OPT\_MANDATORY.

//

// Además, la última entrada de la tabla debe contener todos sus

// elementos nulos, para indicar el final de la misma.

//

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Elementos globales \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static option\_t options[] = {

{1, "i", "input", "-", opt\_input, OPT\_DEFAULT},

{1, "o", "output", "-", opt\_output, OPT\_DEFAULT},

{1, "N", "samples\_length", "-", opt\_samples\_length, OPT\_DEFAULT},

{1, "log", "log\_file", "-", opt\_log\_file, OPT\_DEFAULT},

{0, "h", "help", NULL, opt\_help, OPT\_DEFAULT},

{0, },

};

static int samples\_length;

static istream \*iss = 0; // Input Stream (clase para manejo de los flujos de entrada)

static ostream \*oss = 0; // Output Stream (clase para manejo de los flujos de salida)

static ostream \*log = 0;

static fstream ifs; // Input File Stream (derivada de la clase ifstream que deriva de istream para el manejo de archivos)

static fstream ofs; // Output File Stream (derivada de la clase ofstream que deriva de ostream para el manejo de archivos)

static fstream logfs;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void

opt\_input(string const &arg)

{

// Si el nombre del archivos es "-", usaremos la entrada

// estándar. De lo contrario, abrimos un archivo en modo

// de lectura.

//

if (arg == "-") {

iss = &cin; // Establezco la entrada estandar cin como flujo de entrada

}

else {

ifs.open(arg.c\_str(), ios::in); // c\_str(): Returns a pointer to an array that contains a null-terminated

// sequence of characters (i.e., a C-string) representing

// the current value of the string object.

iss = &ifs;

}

// Verificamos que el stream este OK.

//

if (!iss->good()) {

cerr << "cannot open "

<< arg

<< "."

<< endl;

exit(1);

}

}

static void

opt\_output(string const &arg)

{

// Si el nombre del archivos es "-", usaremos la salida

// estandar. De lo contrario, abrimos un archivo en modo

// de escritura.

//

if (arg == "-") {

oss = &cout; // Establezco la salida estandar cout como flujo de salida

} else {

ofs.open(arg.c\_str(), ios::app);

oss = &ofs;

}

// Verificamos que el stream este OK.

//

if (!oss->good()) {

cerr << "cannot open "

<< arg

<< "."

<< endl;

exit(1); // EXIT: Terminación del programa en su totalidad

}

}

static void

opt\_samples\_length(string const &arg)

{

if (arg == "-")

samples\_length = DEFAULT\_N; //Determina el valor por defecto N = 500

else {

istringstream iss(arg);

// Intentamos extraer el factor de la línea de comandos.

// Para detectar argumentos que únicamente consistan de

// números enteros, vamos a verificar que EOF llegue justo

// después de la lectura exitosa del escalar.

//

if (!(iss >> samples\_length) || !iss.eof() || samples\_length<=0) {

cerr << "non-integer: " << arg << "." << endl;

exit(1);

}

if (iss.bad()) {

cerr << "cannot read integer factor."

<< endl;

exit(1);

}

}

}

static void

opt\_help(string const &arg)

{

cout << "tp0 [-N nsamples] [-i file] [-o file] [-log file]"

<< endl;

exit(0);

}

static void

opt\_log\_file(string const &arg){

if (arg == "-"){

logfs.open("result.log", ios::app);

log = &ofs;

} else {

logfs.open(arg.c\_str(), ios::app);

log = &ofs;

}

if (!log->good()) {

cerr << "cannot open "

<< arg

<< "."

<< endl;

exit(1);

}

}

void

RF\_process(istream \*is, ostream \*os)

{

complejo num, aux;

bool is\_EOF=false;

int i,j=0;

while(is\_EOF==false){

num=0;

for(i=0; i<samples\_length;i++){

if( !(\*is >> aux )){

is\_EOF = true;

break;

}

num = num + aux;

j++;

}

if( i != 0 ){

num = num / i;

\*os << num.abs() << "\n";

}

}

if (os->bad()) {

log\_msg\_printer(&logfs,LOG\_UNSUCCESSFULL\_WRITE\_MSG);

exit(1);

}

if (is->bad()) {

log\_msg\_printer(&logfs,LOG\_UNSUCCESSFULL\_READ\_MSG,j++);

exit(1);

}

if (!is->eof()) {

log\_msg\_printer(&logfs,LOG\_NOT\_EOF\_MSG);

exit(1);

}

log\_msg\_printer(&logfs,LOG\_SUCCESSFULL\_MSG);

}

int

main(int argc, char \* const argv[])

{

cmdline cmdl(options); // Objeto con parametro tipo option\_t (struct) declarado globalmente. Ver línea 51 main.cc

cmdl.parse(argc, argv); // Metodo de parseo de la clase cmdline

RF\_process(iss, oss); // Función externa encargada de procesar y desplegar los datos

}

***cmdline.h:***

#ifndef \_CMDLINE\_H\_INCLUDED\_

#define \_CMDLINE\_H\_INCLUDED\_

#include <string>

#include <iostream>

#define OPT\_DEFAULT 0

#define OPT\_SEEN 1

#define OPT\_MANDATORY 2

struct option\_t {

int has\_arg;

const char \*short\_name;

const char \*long\_name;

const char \*def\_value;

void (\*parse)(std::string const &); // Puntero a función de opciones

int flags;

};

class cmdline {

// Este atributo apunta a la tabla que describe todas

// las opciones a procesar. Por el momento, sólo puede

// ser modificado mediante contructor, y debe finalizar

// con un elemento nulo.

option\_t \*option\_table;

// El constructor por defecto cmdline::cmdline(), es

// privado, para evitar construir "parsers" (analizador

// sintáctico, recibe una palabra y lo interpreta en

// una acción dependiendo su significado para el programa)

// sin opciones. Es decir, objetos de esta clase sin opciones.

cmdline();

int do\_long\_opt(const char \*, const char \*);

int do\_short\_opt(const char \*, const char \*);

public:

cmdline(option\_t \*);

void parse(int, char \* const []);

};

#endif

***cmdline.cc:***

// cmdline - procesamiento de opciones en la línea de comando.

#include <string>

#include <cstdlib>

#include <iostream>

#include <cmdline.h>

using namespace std;

cmdline::cmdline()

{

}

cmdline::cmdline(option\_t \*table) : option\_table(table)

{

/\*

- Lo mismo que hacer:

option\_table = table;

Siendo "option\_table" un atributo de la clase cmdline

y table un puntero a objeto o struct de "option\_t".

Se estaría contruyendo una instancia de la clase cmdline

cargandole los datos que se hayan en table (la tabla con

las opciones, ver el código en main.cc)

\*/

}

void

cmdline::parse(int argc, char \* const argv[])

{

#define END\_OF\_OPTIONS(p) \

((p)->short\_name == 0 \

&& (p)->long\_name == 0 \

&& (p)->parse == 0)

// Primer pasada por la secuencia de opciones: marcamos

// todas las opciones, como no procesadas. Ver código de

// abajo.

//

for (option\_t \*op = option\_table; !END\_OF\_OPTIONS(op); ++op)

op->flags &= ~OPT\_SEEN;

// Recorremos el arreglo argv. En cada paso, vemos

// si se trata de una opción corta, o larga. Luego,

// llamamos a la función de parseo correspondiente.

//

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

// Todos los parámetros de este programa deben

// pasarse en forma de opciones. Encontrar un

// parámetro no-opción es un error.

//

if (argv[i][0] != '-') {

cerr << "Invalid non-option argument: "

<< argv[i]

<< endl;

exit(1);

}

// Usamos "--" para marcar el fin de las

// opciones; todo los argumentos que puedan

// estar a continuación no son interpretados

// como opciones.

//

if (argv[i][1] == '-'

&& argv[i][2] == 0)

break;

// Finalmente, vemos si se trata o no de una

// opción larga; y llamamos al método que se

// encarga de cada caso.

//

if (argv[i][1] == '-')

i += do\_long\_opt(&argv[i][2], argv[i + 1]);

else

i += do\_short\_opt(&argv[i][1], argv[i + 1]);

}

// Segunda pasada: procesamos aquellas opciones que,

// (1) no hayan figurado explícitamente en la línea

// de comandos, y (2) tengan valor por defecto.

//

for (option\_t \*op = option\_table; !END\_OF\_OPTIONS(op); ++op) {

#define OPTION\_NAME(op) \

(op->short\_name ? op->short\_name : op->long\_name)

if (op->flags & OPT\_SEEN)

continue;

if (op->flags & OPT\_MANDATORY) {

cerr << "Option "

<< "-"

<< OPTION\_NAME(op)

<< " is mandatory."

<< "\n";

exit(1);

}

if (op->def\_value == 0)

continue;

op->parse(string(op->def\_value));

}

}

int

cmdline::do\_long\_opt(const char \*opt, const char \*arg)

{

// Recorremos la tabla de opciones, y buscamos la

// entrada larga que se corresponda con la opción de

// línea de comandos. De no encontrarse, indicamos el

// error.

//

for (option\_t \*op = option\_table; op->long\_name != 0; ++op) {

if (string(opt) == string(op->long\_name)) {

// Marcamos esta opción como usada en

// forma explícita, para evitar tener

// que inicializarla con el valor por

// defecto.

//

op->flags |= OPT\_SEEN;

if (op->has\_arg) {

// Como se trada de una opción

// con argumento, verificamos que

// el mismo haya sido provisto.

//

if (arg == 0) {

cerr << "Option requires argument: "

<< "--"

<< opt

<< "\n";

exit(1);

}

op->parse(string(arg));

return 1;

} else {

// Opción sin argumento.

//

op->parse(string(""));

return 0;

}

}

}

// Error: opción no reconocida. Imprimimos un mensaje

// de error, y finalizamos la ejecución del programa.

//

cerr << "Unknown option: "

<< "--"

<< opt

<< "."

<< endl;

exit(1);

// Algunos compiladores se quejan con funciones que

// lógicamente no pueden terminar, y que no devuelven

// un valor en esta última parte.

//

return -1;

}

int

cmdline::do\_short\_opt(const char \*opt, const char \*arg)

{

option\_t \*op;

// Recorremos la tabla de opciones, y buscamos la

// entrada corta que se corresponda con la opción de

// línea de comandos. De no encontrarse, indicamos el

// error.

//

for (op = option\_table; op->short\_name != 0; ++op) {

if (string(opt) == string(op->short\_name)) {

// Marcamos esta opción como usada en

// forma explícita, para evitar tener

// que inicializarla con el valor por

// defecto.

//

op->flags |= OPT\_SEEN;

if (op->has\_arg) {

// Como se trata de una opción

// con argumento, verificamos que

// el mismo haya sido provisto.

//

if (arg == 0) {

cerr << "Option requires argument: "

<< "-"

<< opt

<< "\n";

exit(1);

}

op->parse(string(arg));

return 1;

} else {

// Opción sin argumento.

//

op->parse(string(""));

return 0;

}

}

}

// Error: opción no reconocida. Imprimimos un mensaje

// de error, y finalizamos la ejecución del programa.

//

cerr << "Unknown option: "

<< "-"

<< opt

<< "."

<< endl;

exit(1);

// Algunos compiladores se quejan con funciones que

// lógicamente no pueden terminar, y que no devuelven

// un valor en esta última parte.

//

return -1;

}

***complejo.h:***

#ifndef \_COMPLEJO\_H\_INCLUDED\_

#define \_COMPLEJO\_H\_INCLUDED\_

#include <iostream>

class complejo {

double re\_, im\_;

public:

complejo();

complejo(double);

complejo(double, double);

complejo(const complejo &);

complejo const &operator=(complejo const &);

complejo const &operator\*=(complejo const &);

complejo const &operator+=(complejo const &);

complejo const &operator-=(complejo const &);

~complejo();

double re() const;

double im() const;

double abs() const;

double abs2() const;

complejo const &conjugar();

complejo const conjugado() const;

bool zero() const;

friend complejo const operator+(complejo const &, complejo const &);

friend complejo const operator-(complejo const &, complejo const &);

friend complejo const operator\*(complejo const &, complejo const &);

friend complejo const operator/(complejo const &, complejo const &);

friend complejo const operator/(complejo const &, double);

friend bool operator==(complejo const &, double);

friend bool operator==(complejo const &, complejo const &);

friend std::ostream &operator<<(std::ostream &, const complejo &);

friend std::istream &operator>>(std::istream &, complejo &);

};

#endif

***complejo.cc:***

#include "complejo.h"

#include <iostream>

#include <cmath>

using namespace std;

complejo::complejo() : re\_(0), im\_(0){}

complejo::complejo(double r) : re\_(r), im\_(0){}

complejo::complejo(double r, double i) : re\_(r), im\_(i){}

complejo::complejo(complejo const &c) : re\_(c.re\_), im\_(c.im\_){}

complejo const &

complejo::operator=(complejo const &c){

re\_ = c.re\_;

im\_ = c.im\_;

return \*this;

}

complejo const &

complejo::operator\*=(complejo const &c){

double re = re\_ \* c.re\_

- im\_ \* c.im\_;

double im = re\_ \* c.im\_

+ im\_ \* c.re\_;

re\_ = re, im\_ = im;

return \*this;

}

complejo const &

complejo::operator+=(complejo const &c){

double re = re\_ + c.re\_;

double im = im\_ + c.im\_;

re\_ = re, im\_ = im;

return \*this;

}

complejo const &

complejo::operator-=(complejo const &c){

double re = re\_ - c.re\_;

double im = im\_ - c.im\_;

re\_ = re, im\_ = im;

return \*this;

}

complejo::~complejo(){}

double

complejo::re() const{

return re\_;

}

double complejo::im() const{

return im\_;

}

double

complejo::abs() const{

return std::sqrt(re\_ \* re\_ + im\_ \* im\_);

}

double

complejo::abs2() const{

return re\_ \* re\_ + im\_ \* im\_;

}

complejo const &

complejo::conjugar(){

im\_\*= -1;

return \*this;

}

complejo const

complejo::conjugado() const{

return complejo(re\_, -im\_);

}

bool

complejo::zero() const{

#define ZERO(x) ((x) == +0.0 && (x) == -0.0)

return ZERO(re\_) && ZERO(im\_) ? true : false;

}

complejo const

operator+(complejo const &x, complejo const &y){

complejo z(x.re\_ + y.re\_, x.im\_ + y.im\_);

return z;

}

complejo const

operator-(complejo const &x, complejo const &y){

complejo r(x.re\_ - y.re\_, x.im\_ - y.im\_);

return r;

}

complejo const

operator\*(complejo const &x, complejo const &y){

complejo r(x.re\_ \* y.re\_ - x.im\_ \* y.im\_,

x.re\_ \* y.im\_ + x.im\_ \* y.re\_);

return r;

}

complejo const

operator/(complejo const &x, complejo const &y){

return x \* y.conjugado() / y.abs2();

}

complejo const

operator/(complejo const &c, double f){

return complejo(c.re\_ / f, c.im\_ / f);

}

bool

operator==(complejo const &c, double f){

bool b = (c.im\_ != 0 || c.re\_ != f) ? false : true;

return b;

}

bool

operator==(complejo const &x, complejo const &y){

bool b = (x.re\_ != y.re\_ || x.im\_ != y.im\_) ? false : true;

return b;

}

ostream &

operator<<(ostream &os, const complejo &c)

{

return os << "("

<< c.re\_

<< ", "

<< c.im\_

<< ")";

}

istream &

operator>>(istream &is, complejo &c){

int good = false;

int bad = false;

double re = 0;

double im = 0;

char ch = 0;

if (is >> ch

&& ch == '(') {

if (is >> re

&& is >> ch

&& ch == ','

&& is >> im

&& is >> ch

&& ch == ')')

good = true;

else

bad = true;

} else if (is.good()) {

is.putback(ch);

if (is >> re)

good = true;

else

bad = true;

}

if (good)

c.re\_ = re, c.im\_ = im;

if (bad)

is.clear(ios::badbit);

return is;

}

***log\_msg.h:***

#ifndef \_LOG\_MSG\_H\_INCLUDED\_

#define \_LOG\_MSG\_H\_INCLUDED\_

#define LOG\_SUCCESSFULL\_MSG "The operation was successfully complete"

#define LOG\_UNSUCCESSFULL\_READ\_MSG "The operation failed in the sample number "

#define LOG\_UNSUCCESSFULL\_WRITE\_MSG "Cannot write to output stream."

#define LOG\_NOT\_EOF\_MSG "Cannot find EOF on input stream "

#define LOG\_NOT\_VALID\_FACTOR\_MSG "Not valid factor"

#endif

***log\_printer.cc:***

#include <string>

#include <cstdlib>

#include <iostream>

#include "log\_msg.h"

using namespace std;

void log\_msg\_printer(fstream \*log,const char\* msg)

{

\*log<< msg << endl;

}

void log\_msg\_printer(fstream \*log,const char\* msg, int failed\_operation)

{

\*log<< msg << failed\_operation << endl;

}

***Makefile:***

CXXARGS = -g -Wall

CXXFLAGS = -I. $(CXXARGS)

all: tp0.exe

tp0.exe: main.cc cmdline.cc cmdline.h

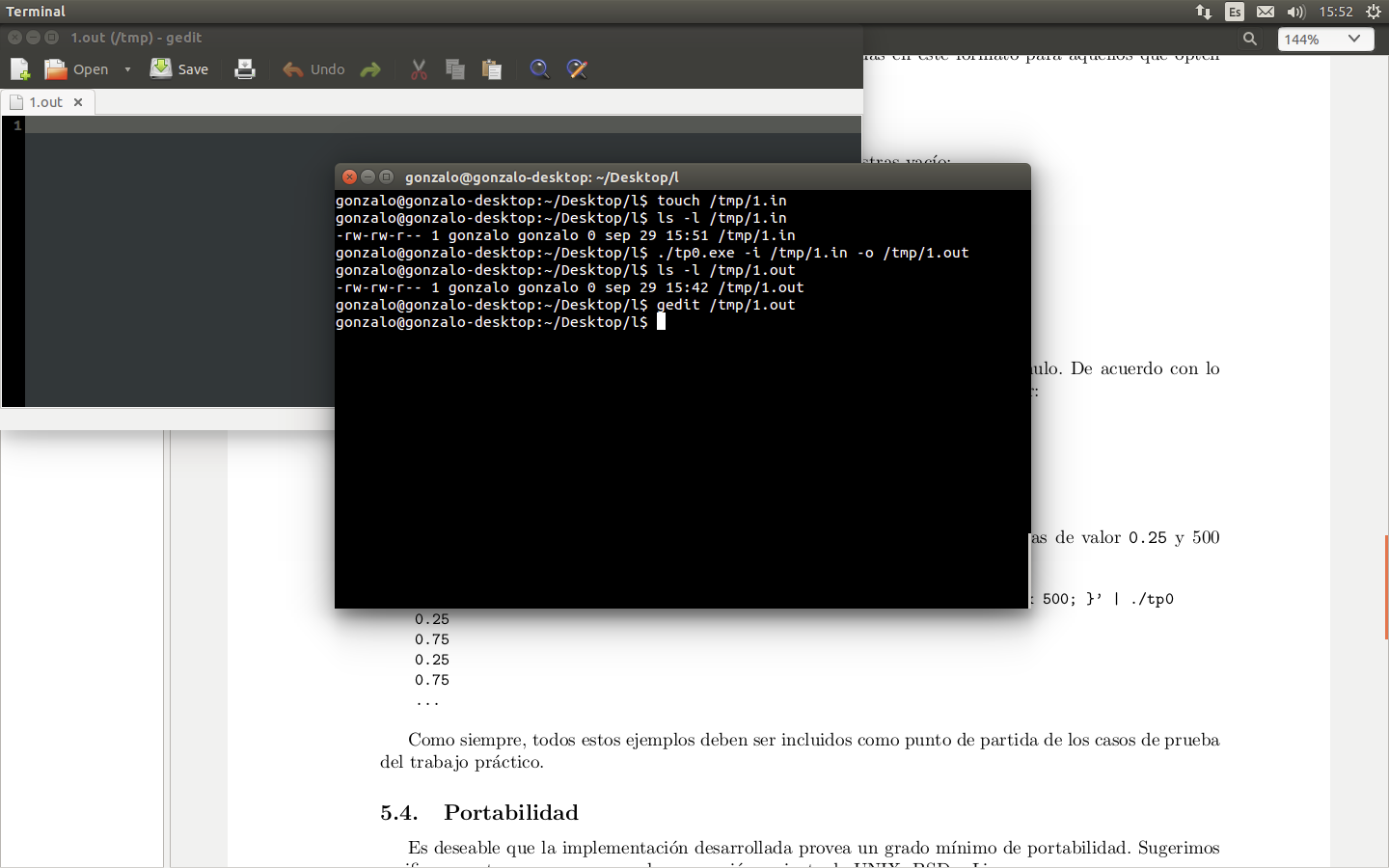
$(CXX) $(CXXFLAGS) -o tp0.exe cmdline.cc main.cc complejo.cc

clean:

$(RM) -vf \*.o \*.exe \*.t \*.out \*.err

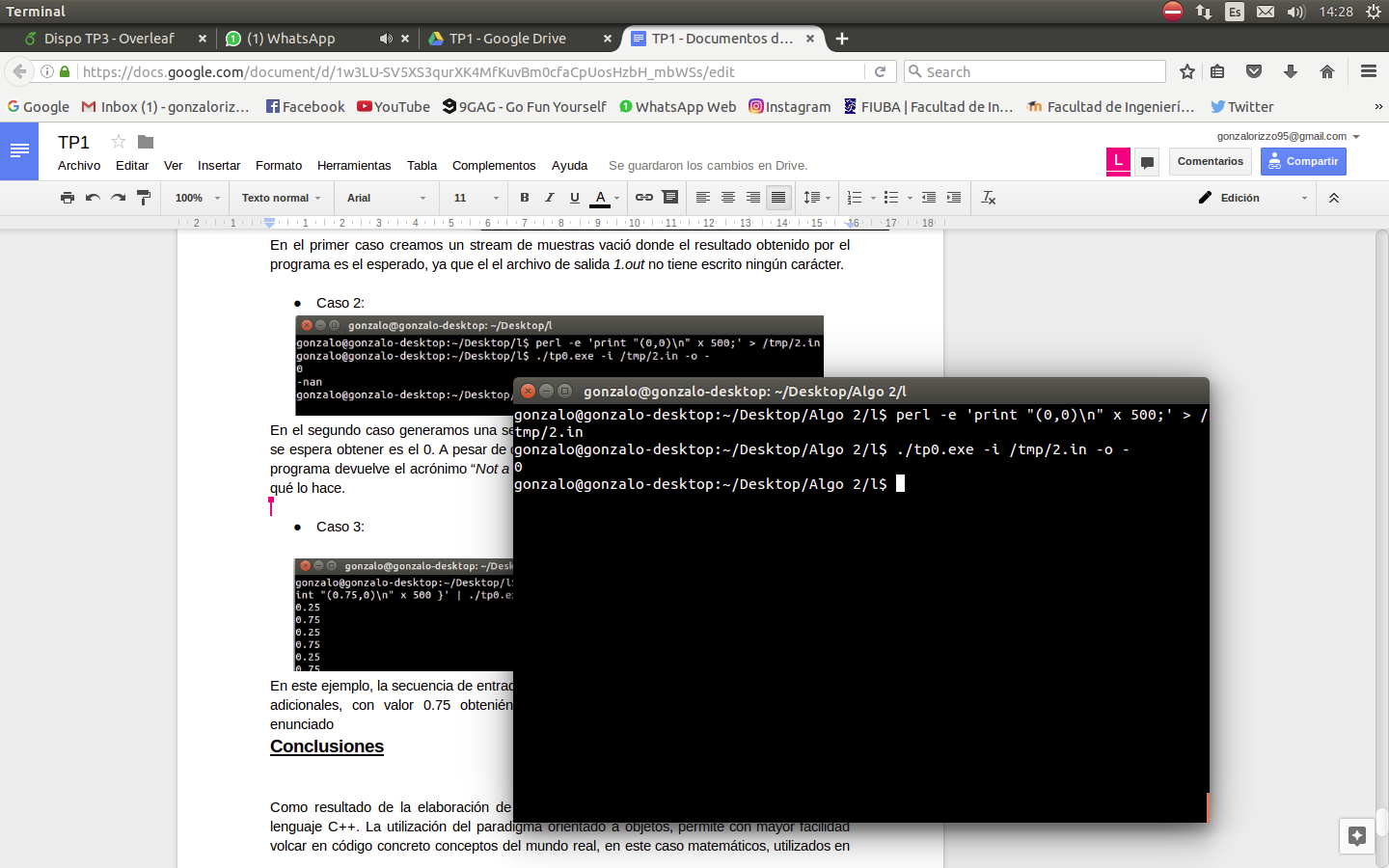
**Corridas de prueba**

* Caso 1:



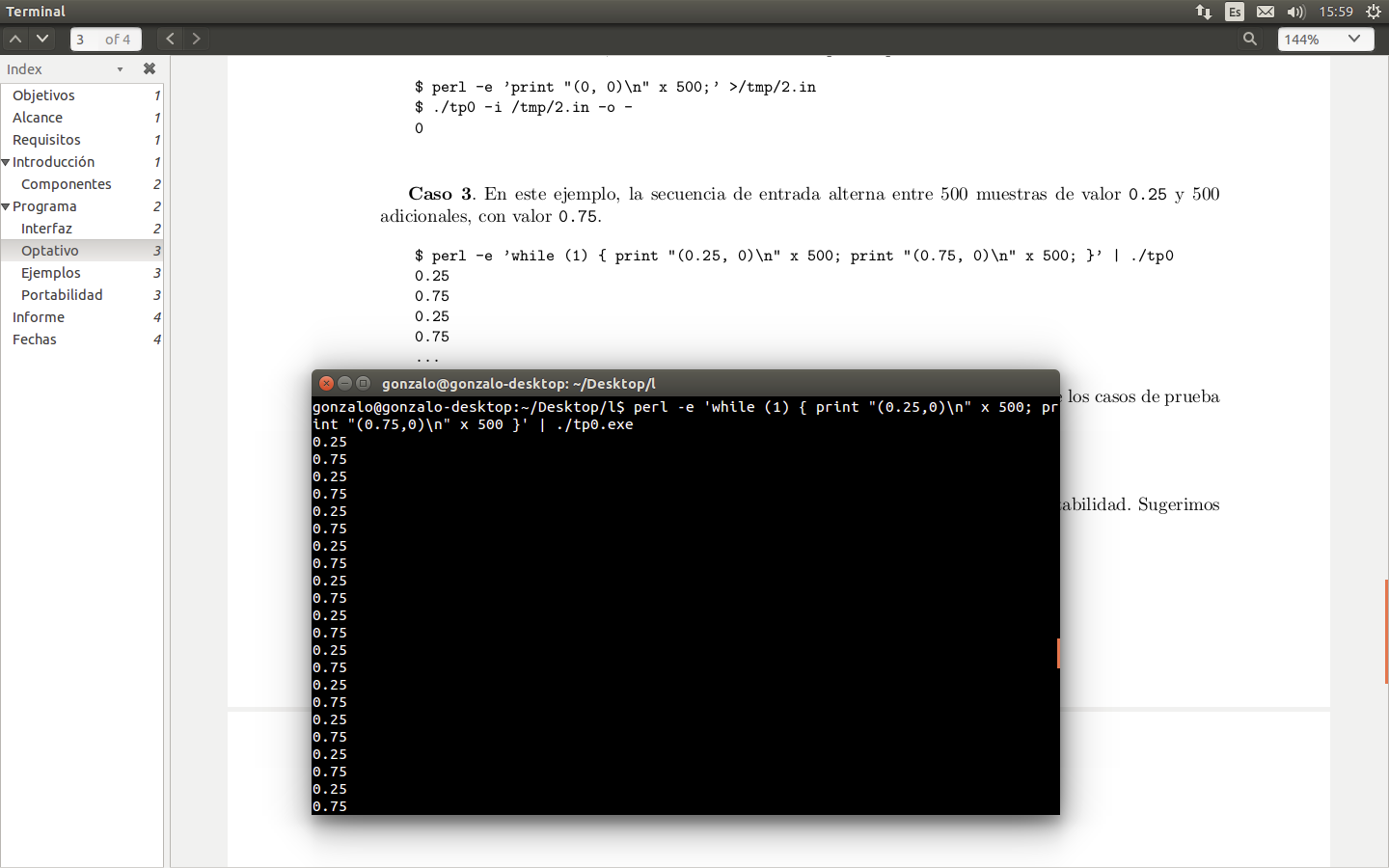
En el primer caso creamos un stream de muestras vació donde el resultado obtenido por el programa es el esperado, ya que el el archivo de salida *1.out* no tiene escrito ningún carácter.

* Caso 2:



En el segundo caso generamos una secuencia de 500 complejos con valor nulo. El valor que se espera obtener es el 0. En la anterior entrega, el programa devolvía a continuación del 0, el acrónimo “*Not a Number”.* Luego de revisar el código se encontró que la razón por la cual se presentaba ese acrónimo, era debido a una división por cero que se hacía mediante la variable i = 0 cuando se calculaba el promedio de las muestras. Cuando se completaban las 500 muestras el cursor no avanzaba sino hasta volver a entrar en el ciclo *for*, donde se leía en ese momento, el carácter *EOF* y la variable *i* seguía siendo igual a 0. Por lo tanto se agregó un *if* que verifica, para esos casos, que la variable i sea distinta de 0 para proseguir con el procesamiento.

* Caso 3:



En este ejemplo, la secuencia de entrada alterna entre 500 muestras de valor 0.25 y 500

adicionales, con valor 0.75 obteniéndose el mismo resultado que se especifica en el enunciado.

* **Resultados de corridas con VALGRIND:**

Se puede observar que en la corrida del programa con **“*valgrind --leak-check=yes ./tp0 -<opciones correspondientes> <argumento correspondiente> “***, la salida del *memcheck* no mostró ninguna pérdida de memoria.

==4667== HEAP SUMMARY:

==4667== in use at exit: 72,704 bytes in 1 blocks

==4667== total heap usage: 23 allocs, 22 frees, 99,848 bytes allocated

==4667==

==4667== LEAK SUMMARY:

==4667== definitely lost: 0 bytes in 0 blocks

==4667== indirectly lost: 0 bytes in 0 blocks

==4667== possibly lost: 0 bytes in 0 blocks

==4667== still reachable: 72,704 bytes in 1 blocks

==4667== suppressed: 0 bytes in 0 blocks

**Conclusiones**

Como resultado de la elaboración de este trabajo, se tuvo una primera experiencia con el lenguaje C++. La utilización del paradigma orientado a objetos, permite con mayor facilidad volcar en código concreto conceptos del mundo real, en este caso matemáticos, utilizados en el procesamiento de señales. Por ejemplo, el hecho de utilizar a los complejos como un objeto resulta más natural a la hora de aplicar las operaciones características que los mismos realizan, ya que uno se puede abstraer de la forma en la que los complejos trabajan y de esa forma pensar el problema a resolver desde otro punto de vista..

Más allá de que en C puede usarse el paradigma orientado a objetos utilizando estructuras, no permite la utilización del mismo con tanta naturalidad como C++.

**Bibliografía**

-H. M. DEITEL y P. J. DEITEL, “Cómo programar en C++”, Sexta edición, Prentice Hall, México, 2008.