FACULTAD DE INGENIERÍA
DE LA
UNIVERSIDAD DE BUENOS AIRES

Algoritmos y Programación II [95.12]

Trabajo Práctico n.° 1:

Objetos y algoritmos

Integrantes: Grassi, Tomás Miguel (99551) - tomas96@gmail.com

Martinez Mikulic, Mateo (99602) - mmartinezmikulic@gmail.com

Wagner, Marcos (98607) - marcoswagneer.18@gmail.com

Profesor: Ing. Calvo, Patricia

Ing. Santi, Leandro

Lic. Santi, Lucio

Curso 1
Jueves 17 de Mayo de 2018

Introducción

En este trabajo se busca obtener conocimientos de programación orientada a objetos y de diseño de algoritmos. Para ello se realiza un programa utilizando el patrón de diseño Strategy y se mejora el diseño original previo de la transformada discreta de Fourier implementando la transformación rápida de Fourier (FFT) y su transformada inversa (IDFT).

Diseño e implementación del programa

Diseño del programa

Se implementó para este programa la clase Complex y la clase Vector, ambas implementadas como templates. De esta forma se logra una mayor simplicidad, ya que se codifica una única función sin importar el tipo de dato que se le pase como parámetro, logrando un código más mantenible. Además, a partir del uso del uso de templates se logra una generalización de cada clase, ya que la misma se puede utilizar para distintos tipos de datos. Por ejemplo, la clase vector, gracias a la implementación en forma de plantilla, es posible que cada instancia del vector contenga cualquier objeto, como un complejo, o tipo de dato que se necesite.

La clase Complex es utilizada para cargar cada uno de los datos leídos y dentro de la clase Vector se ordenan en orden leído cada uno de los objetos de la clase Complex.

Para la clase Complex se sobrecargaron los operadores correspondientes a la suma, resta, multiplicación, división, los cuales son utilizados en la implementación de todas las transformadas. Además se sobrecargaron los operadores de asignación, negación y comparación. De esta forma se logra un código más mantenible y legible. Por otro lado para la clase vector se sobrecargaron los operadores de asignación e indexación.

Por último, se utilizó el código provisto para el manejo de argumentos, realizando los cambios necesarios para que parsee el archivo de señales, de forma que lea una señal por línea.

Patrón de diseño

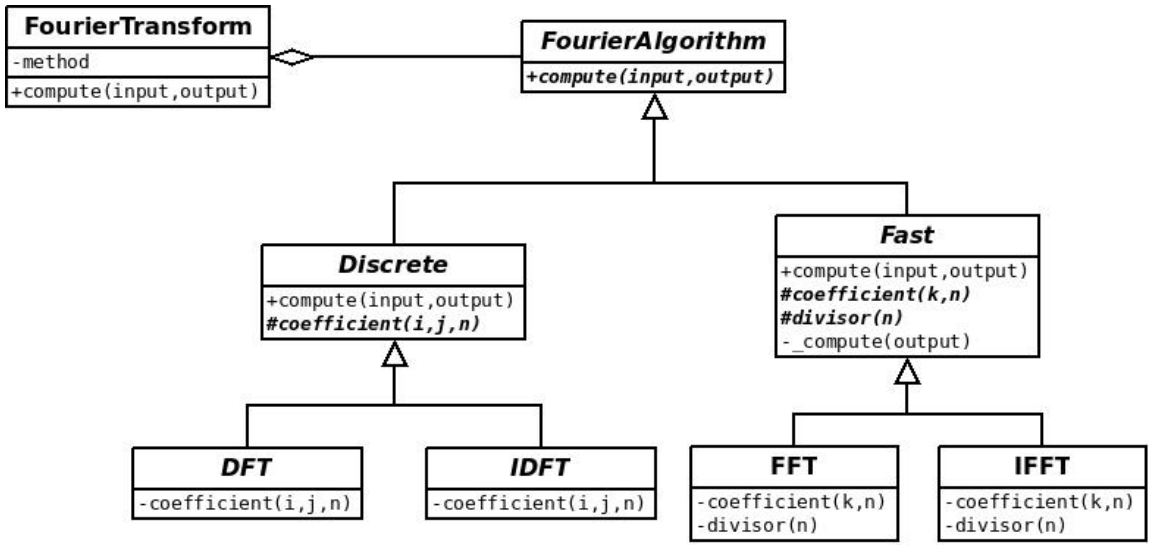

Figura 1: Diagrama de clases según el lenguaje unificado de modelado UML

Se utilizó para este proyecto el patrón de diseño Strategy. Esto permite reutilizar código y proporciona al usuario una misma interfaz que asegura un correcto funcionamiento sin importar la transformada utilizada. Según el método elegido el constructor de FourierTransform recibirá un puntero a una subclase de FourierAlgorithm. Esta subclase de FourierAlgorithm ejecutará la función *_compute()* con el coeficiente correspondiente.

Diseño del algoritmo de la transformada rápida de Fourier

La transformada rápida de Fourier se realizó de manera recursiva con el método dividir y conquistar, aprovechando la propiedad de las raíces complejas de la unidad y su periodicidad, que permite computar la DFT en tiempo $O(nlg(n))$, en lugar de $O(n^2)$. El requisito para esto es que el número de entradas sea una potencia de 2. Al realizar dividir y conquistar se divide en cada pasada recursiva a los elementos en posiciones par y a los posicionados en numeros impares, y se los procesa por separado, logrando reducir la cantidad de operaciones totales.

Interfaz

La interacción con el programa es a través de comandos en línea de ordenes. Tanto la entrada como la salida, puede direccionarse desde o hacia un archivo utilizando el flag correspondiente (*-i, -o* respectivamente) seguido del nombre del archivo. En caso de no indicar ningún archivo, el programa utiliza los flujos standar de entrada (por teclado) y salida (por pantalla). Por otro lado, también es posible indicar el método de transformación que se desea utilizar a partir del flag de método *-m* e indicando luego el método por su abreviatura (DFT, IDFT, FFT o IFFT). En caso de que esta opción no sea indicada, el programa realiza la FFT por defecto.

Formato de entrada y salida

El archivo de entrada será un archivo de texto con pares ordenados de complejos (re, im), separados por espacios. Cada línea en el archivo de entrada será una señal diferente. La salida tendrá el mismo formato, siendo cada línea la transformada o antitransformada de la señal correspondiente.

Corridas de prueba

Las pruebas realizadas son similares a las realizadas en el proyecto anterior. Solo se agregaron las pruebas correspondientes al procesamiento de datos con FFT e IFFT. Estas pruebas permitieron determinar que el tiempo de procesamiento de la transformada rápida de Fourier es mil veces mas rápida que la transformada discreta de Fourier.

En las pruebas realizadas con un número de entradas diferente a una potencia de dos el resultado varía al desarrollado por la DFT al rellenar con ceros los complejos faltantes.

Para realizar las pruebas de funcionamiento decidimos usar Google C++ Testing Framework, una herramienta que permite que las pruebas sean independientes y repetibles. Permite separar las pruebas en módulos separados, reflejando la estructura del código evaluado y logrando de esta manera que sea mas entendible y mantenible. Al fallar una prueba, según la naturaleza de la falla, la prueba puede seguir desarrollándose o simplemente detener ese módulo para continuar con el siguiente.

En este proyecto se realizaron pruebas sobre las clases desarrolladas y luego sobre el procesamiento de datos. Al realizar pruebas sobre las clases se verificó que las funciones correspondientes a la clase Complex funcionaran correctamente, tomando como patrón de comparación la clase *std::complex* definida en el archivo de cabecera *<complex.h>* provisto por la biblioteca estándar de C++. Al desarrollar las pruebas de procesamiento de datos el enfoque fue primero la verificación del correcto

funcionamiento tanto de la función *DFT()* como de la función *IDFT()* y luego la robustez del proceso, realizando pruebas que variaban en el volumen de datos pseudo-aleatorios obtenidos con *rand()*.

Probando la clase Complex de esta manera, se descubrió que la forma en que se estaba implementando la función *exp()*, que calcula la exponencial compleja, difería de los valores obtenidos con *std::exp()* en algunos casos: en general, dentro de los 10 000 000 de números que se probaba, fallaba en alrededor de 20. Esto se debía a que la función de comparación implementada con el fin de que las comparaciones tengan en cuenta una cierta tolerancia debida a los errores propios de la aritmética de punto flotante no manejaban adecuadamente el valor inf.

Asimismo, se realizaron pruebas para determinar cuál debía ser el valor máximo en que pueden diferir dos números para ser considerados iguales. Para esto se fueron ejecutando las pruebas creadas para números complejos y para la DFT variando, de a potencias de diez, el factor por el que se multiplicaba a *std::numeric_limits<long double>::epsilon()*, el cual es la diferencia mínima que debe haber entre 1 y un número para que éste sea considerado el siguiente valor respresentable. De esta forma se llegó a una cota en el valor mínimo de comparación: 10^{-6}.

Para ejecutar las pruebas sólo es necesario usar el comando make y luego ejecutar el programa de pruebas; éstas informan si alguno de los resultados no es el esperado.

Problemas durante el desarrollo y soluciones

Al realizar el algoritmo de un proceso tan complejo como la transformada rápida de Fourier se desarrollaron varios problemas y contratiempos hasta que se pudo lograr una versión del algoritmo funcional. Estos errores fueron causados por el uso de la recursividad en un algoritmo del cual desconocíamos su funcionamiento. Los valores en el exponente en *coefficient()* y el desarrollo de la función inversa fue lo que más confusión causó.

Por otro lado, utilizando Valgrind se verificó que no hubiera fuga de memoria. Inicialmente había un uso incorrecto de memoria dinámica lo cual provocó que el programa pise memoria que no le correspondia. Gracias a este diagnóstico se pudo diseñar el programa de forma correcta.

Conclusiones

El programa cumple su función de manera eficaz cualquiera sea el método de la transformada que se utilice. La eficiencia del mismo depende del método utilizado, siendo la FFT mil veces más eficiente que la DFT en cuanto al tiempo de ejecución. Sin embargo, es posible mejorar ligeramente la eficiencia del algoritmo utilizando propiedades de los numeros complejos para reducir parcialmente la cantidad de operaciones que se realizan aunque la complejidad final seguiría siendo *O(nlg(n))*. Por otro lado utilizando algoritmos más complejos, sería posible trabajar con una cantidad de muestras de señales que no sean necesariamente potencia de dos, y de esta manera evitar aproximar los resultados de nuestra transformada al completar la muestra con ceros al final del vector hasta llegar a la potencia entera de 2 más cercana.

Por otro lado, utilizar herramientas como Google C++ Testing Framework facilita la tarea de probar el funcionamiento y permite de manera sencilla verificar el funcionamiento a medida que diferentes cambios son realizados al código.

Bibliografía

- Ghezzi, Carlo & Jazayeri, Mehdi & Mandrioli, Dino (1991). *Fundamentals of Software Engineering* (1.^{era} ed.) Upper Saddle River, NJ 07458: Prentice Hall, Inc.

- Stroustrup, Bjarne (1988). *The C++ Programming Language* (4.^{ta} ed.) Upper Saddle River, NJ 07458: Addison-Wesley.
- Cormen, Thomas & Leiserson, Charles & Rivest, Ronald & Stein, Clifford (1989). *Introduction to Algorithms* (1.^{era} ed.) Upper Saddle River, NJ 07458: MIT Press.

Script de compilación

makefile

```
CXXFLAGS = -g -Wall -Wpedantic -Wdeprecated -std=c++11 -O3
SRC = source
INCLUDE = include
TESTS = source/tests
GOOGLETEST =  source/tests/googletest
CXXARGS = -I. -iquote $(INCLUDE) -isystem $(GOOGLETEST)/include -pthread

all: fourier.exe Complex_test.exe fourier_test.exe

cmdline.o: $(SRC)/cmdline.cpp $(INCLUDE)/cmdline.h
   $(CXX) $(CXXFLAGS) $(CXXARGS) -c $(SRC)/cmdline.cpp -o cmdline.o

fourier.o: $(SRC)/fourier.cpp $(INCLUDE)/fourier.h $(INCLUDE)/Complex.h $
   (INCLUDE)/Vector.h
   $(CXX) $(CXXFLAGS) $(CXXARGS) -c $(SRC)/fourier.cpp -o fourier.o

io.o: $(SRC)/io.cpp $(INCLUDE)/io.h $(INCLUDE)/Complex.h $(INCLUDE)/
   Vector.h
   $(CXX) $(CXXFLAGS) $(CXXARGS) -c $(SRC)/io.cpp -o io.o

main.o: $(SRC)/main.cpp $(INCLUDE)/main.h $(INCLUDE)/io.h $(INCLUDE)/
   Complex.h $(INCLUDE)/Vector.h $(INCLUDE)/cmdline.h $(INCLUDE)/fourier.h
   $(CXX) $(CXXFLAGS) $(CXXARGS) -c $(SRC)/main.cpp -o main.o

fourier.exe: cmdline.o fourier.o io.o main.o
   $(CXX) $(CXXFLAGS) $(CXXARGS) cmdline.o fourier.o  io.o main.o -o
      fourier.exe

gtest-all.o:
   $(CXX) $(CXXFLAGS) -isystem $(GOOGLETEST)/include -I$(GOOGLETEST) -
      pthread -c $(GOOGLETEST)/src/gtest-all.cc -o gtest-all.o

Complex_test.o: $(TESTS)/Complex_test.cpp $(INCLUDE)/Complex_test.h $(
   INCLUDE)/Complex.h $(INCLUDE)/Vector.h
   $(CXX) $(CXXFLAGS) $(CXXARGS) -c $(TESTS)/Complex_test.cpp -o
      Complex_test.o

Complex_test.exe: gtest-all.o Complex_test.o cmdline.o
   $(CXX) $(CXXFLAGS) $(CXXARGS) gtest-all.o Complex_test.o cmdline.o -o
      Complex_test.exe

```

```
34 fourier_test.o: $(TESTS)/fourier_test.cpp $(INCLUDE)/fourier_test.h $(
       INCLUDE)/Complex.h $(INCLUDE)/Vector.h $(INCLUDE)/io.h $(INCLUDE)/
       cmdline.h $(INCLUDE)/fourier.h
35     $(CXX) $(CXXFLAGS) $(CXXARGS) -c $(TESTS)/fourier_test.cpp -o
           fourier_test.o
36
37 fourier_test.exe: fourier_test.o gtest-all.o cmdline.o
38     $(CXX) $(CXXFLAGS) $(CXXARGS) gtest-all.o fourier_test.o fourier.o
           cmdline.o io.o -o fourier_test.exe
39
40 clean:
41     $(RM) -vf *.o *.exe *.t *.out *.err
```

Código fuente

main.h

```
1  #ifndef _MAIN_H_INCLUDED_
2  #define _MAIN_H_INCLUDED_
3
4  #include "cmdline.h"
5  #include "Complex.h"
6  #include "Vector.h"
7  #include "io.h"
8  #include "fourier.h"
9
10 static void opt_input(std::string const &);
11 static void opt_output(std::string const &);
12 static void opt_method(std::string const &);
13 static void opt_help(std::string const &);
14 static void print_msg_and_exit(std::string const &);
15
16 #endif   // _MAIN_H_INCLUDED_
```

main.cpp

```
1  #include <fstream>
2  #include <iostream>
3  #include <sstream>
4  #include <cstdlib>
5
6  #include "main.h"
7
8  using namespace std;
9
10 static option_t options[] = {
11     {1, "i", "input", "-", opt_input, OPT_DEFAULT},
12     {1, "o", "output", "-", opt_output, OPT_DEFAULT},
13     {1, "m", "method", "FFT", opt_method, OPT_DEFAULT},
14     {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
15     {0, },
16 };
```

```
17
18 static char *program_name;
19 static FourierAlgorithm *chosen_method;
20 static FourierTransform *transform;
21 static istream *iss = NULL;
22 static ostream *oss = NULL;
23 static fstream ifs;
24 static fstream ofs;
25
26 static void
27 opt_input(string const &arg)
28 {
29     if (arg == "-") {
30         iss = &cin;
31     }
32     else {
33         ifs.open(arg.c_str(), ios::in);
34         iss = &ifs;
35     }
36
37     if (!iss->good()) {
38         cerr << "Cannot open "
39              << arg
40              << "."
41              << endl;
42         exit(1);
43     }
44 }
45
46 static void
47 opt_output(string const &arg)
48 {
49     if (arg == "-") {
50         oss = &cout;
51     } else {
52         ofs.open(arg.c_str(), ios::out);
53         oss = &ofs;
54     }
55
56     if (!oss->good()) {
57         cerr << "Cannot open "
58              << arg
59              << "."
60              << endl;
61         exit(1);
62     }
63 }
64
65 static void
66 opt_method(string const &arg)
67 {
68     istringstream iss(arg);
```

```
69      string read_method;
70
71      if (!(iss >> read_method)
72          || !iss.eof()) {
73          cerr << "Not a posible method: "
74               << arg
75               << "."
76               << endl;
77          exit(1);
78      }
79      if (iss.bad()) {
80          cerr << "Cannot read method."
81               << endl;
82          exit(1);
83      }
84      if (read_method == "FFT")
85          chosen_method = new FFT;
86      else if (read_method == "IFFT")
87          chosen_method = new IFFT;
88      else if (read_method == "DFT")
89          chosen_method = new DFT;
90      else if (read_method == "IDFT")
91          chosen_method = new IDFT;
92      else {
93          cerr << "Cannot read method."
94                  << endl;
95          exit(1);
96      }
97      ::transform = new FourierTransform(chosen_method);
98  }
99
100 static void
101 opt_help(string const &arg)
102 {
103     cout << program_name << " [-m DFT | IDFT] [-i file] [-o file]"
104          << endl;
105     exit(0);
106 }
107
108 static void
109 print_msg_and_exit(string const & msg)
110 {
111     cerr << msg
112          << endl;
113     exit(1);
114 }
115
116 int
117 main(int argc, char * const argv[])
118 {
119     program_name = argv[0];
120     cmdline cmdl(options);
```

```
121     cmdl.parse(argc, argv);
122
123     // Cuestiones de formato para la impresión:
124     oss->setf(ios::fixed, ios::floatfield);
125     oss->precision(6);
126
127     bool status;
128     ComplexVector inSignal;
129     ComplexVector outSignal;
130     istringstream line;
131     string s;
132
133     for (int lineNo = 1; getline(*iss, s); ++lineNo)
134     {
135         if (iss->bad())
136             print_msg_and_exit("An error occured while processing line " +
                    to_string(lineNo) + ".");
137
138         line.str(s); //
139         line.clear();
140
141         status = load_signal(line, inSignal);
142         if (!status)
143             print_msg_and_exit("Error processing \"" + line.str() + "\" (
                    line " + to_string(lineNo) + ").");
144
145         status = ::transform->compute(inSignal, outSignal);
146         if (!status)
147             print_msg_and_exit("An error occured while performing the
                    requested operation.");
148
149         status = print_signal(*oss, outSignal);
150         if (!status)
151             print_msg_and_exit("Cannot write to output stream.");
152         inSignal.clear();
153         outSignal.clear();
154     }
155     delete chosen_method;
156     delete ::transform;
157 }
```

cmdline.h

```
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    option_t *option_table;
22    cmdline();
23    int do_long_opt(const char *, const char *);
24    int do_short_opt(const char *, const char *);
25 public:
26    cmdline(option_t *);
27    void parse(int, char * const []);
28 };
29
30 #endif
```

cmdline.cpp

```
1  // cmdline - procesamiento de opciones en la [U+FFFD]lnea de comando.
2  //
3  // $Date: 2012/09/14 13:08:33 $
4  //
5  #include <string>
6  #include <cstdlib>
7  #include <iostream>
8
9  #include "cmdline.h"
10
11 using namespace std;
12
13 cmdline::cmdline()
14 {
15 }
16
17 cmdline::cmdline(option_t *table) : option_table(table)
18 {
19    /*
20    - Lo mismo que hacer:
21
22    option_table = table;
23
24    Siendo "option_table" un atributo de la clase cmdline
25    y table un puntero a objeto o struct de "option_t".
26
27    Se [U+FFFD]estara contruyendo una instancia de la clase cmdline
28    cargandole los datos que se hayan en table (la table con
29    las opciones, ver el [U+FFFD]cdigo en main.cc)
30
```

```
31     */
32 }
33
34 void
35 cmdline::parse(int argc, char * const argv[])
36 {
37 #define END_OF_OPTIONS(p)       \
38     ((p)->short_name == 0    \
39      && (p)->long_name == 0 \
40      && (p)->parse == 0)
41     for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op)
42         op->flags &= ~OPT_SEEN;
43     for (int i = 1; i < argc; ++i) {
44         if (argv[i][0] != '-') {
45             cerr << "Invalid non-option argument: "
46                  << argv[i]
47                  << endl;
48             exit(1);
49         }
50         if (argv[i][1] == '-'
51              && argv[i][2] == 0)
52             break;
53         if (argv[i][1] == '-')
54             i += do_long_opt(&argv[i][2], argv[i + 1]);
55         else
56             i += do_short_opt(&argv[i][1], argv[i + 1]);
57     }
58     for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op) {
59 #define OPTION_NAME(op) \
60     (op->short_name ? op->short_name : op->long_name)
61         if (op->flags & OPT_SEEN)
62             continue;
63         if (op->flags & OPT_MANDATORY) {
64             cerr << "Option "
65                  << "-"
66                  << OPTION_NAME(op)
67                  << " is mandatory."
68                  << "\n";
69             exit(1);
70         }
71         if (op->def_value == 0)
72             continue;
73         op->parse(string(op->def_value));
74     }
75 }
76
77 int
78 cmdline::do_long_opt(const char *opt, const char *arg)
79 {
80     for (option_t *op = option_table; op->long_name != 0; ++op) {
81         if (string(opt) == string(op->long_name)) {
82             op->flags |= OPT_SEEN;
```

```
83
84          if (op->has_arg) {
85              if (arg == 0) {
86                  cerr << "Option requires argument: "
87                       << "--"
88                       << opt
89                       << "\n";
90                  exit(1);
91              }
92              op->parse(string(arg));
93              return 1;
94          } else {
95              op->parse(string(""));
96              return 0;
97          }
98      }
99  }
100 cerr << "Unknown option: "
101      << "--"
102      << opt
103      << "."
104      << endl;
105 exit(1);
106 // Algunos compiladores se quejan con funciones que
107 // [U+FFFD]lgicamente no pueden terminar, y que no devuelven
108 // un valor en esta [U+FFFD]ltima parte.
109 //
110 return -1;
111 }
112
113 int
114 cmdline::do_short_opt(const char *opt, const char *arg)
115 {
116     option_t *op;
117     for (op = option_table; op->short_name != 0; ++op) {
118         if (string(opt) == string(op->short_name)) {
119             op->flags |= OPT_SEEN;
120             if (op->has_arg) {
121                 if (arg == 0) {
122                     cerr << "Option requires argument: "
123                          << "-"
124                          << opt
125                          << "\n";
126                     exit(1);
127                 }
128                 op->parse(string(arg));
129                 return 1;
130             } else {
131                 op->parse(string(""));
132                 return 0;
133             }
134         }
```

```
    }
    cerr << "Unknown option: "
         << "-"
         << opt
         << "."
         << endl;
    exit(1);
    // Algunos compiladores se quejan con funciones que
    // [U+FFFD]lgicamente no pueden terminar, y que no devuelven
    // un valor en esta [U+FFFD]ltima parte.
    //
    return -1;
}
```

io.h

```
#ifndef _IO_H_INCLUDED_
#define _IO_H_INCLUDED_

#include "Complex.h"
#include "Vector.h"

bool load_signal(std::istream &, Vector<Complex<> > &);
bool print_signal(std::ostream &, Vector<Complex<> > const &);

#endif   //_IO_H_INCLUDED_
```

io.cpp

```
#include <iostream>
#include <cstdlib>
#include <sstream>

#include "io.h"

using namespace std;

bool
load_signal(istream & is, Vector<Complex<> > & input)
{
    Complex <long double> c;
    while (is >> c)
        input.push_back(c);
    if (is.bad())
        return false;
    return true;
}

bool
print_signal(ostream & os, Vector<Complex<> > const & output)
{
    for (size_t i = 0; i < output.size(); ++i)
        os << output[i];
    os << endl;
```

```
    if (os.bad())
        return false;
    return true;
}
```

fourier.h

```
#ifndef _FOURIER_H_INCLUDED_
#define  _FOURIER_H_INCLUDED_

#include "Complex.h"
#include "Vector.h"

using ComplexVector = Vector <Complex <long double> >;

class FourierAlgorithm {
public:
    virtual bool compute(ComplexVector const & input, ComplexVector &
        output) = 0;
    virtual ~FourierAlgorithm() {}
};

class FourierTransform {
public:
    FourierTransform(FourierAlgorithm *method) : _method(method) {}
    virtual ~FourierTransform() {}
    inline bool compute(ComplexVector const & input, ComplexVector &
        output) {
        return _method? _method->compute(input, output) : false;
    }
private:
    FourierAlgorithm *_method;
};

class Discrete : public FourierAlgorithm {
public:
    bool compute(ComplexVector const & input, ComplexVector & output);
protected:
    virtual const Complex<> coefficient(int const i, int const j, int
        const n) = 0;
};

class DFT : public Discrete {
private:
    inline const Complex <> coefficient(int const i, int const j, int
        const n) override {
        return exp(I * -2.0 * M_PI * i * j / n);
    }
};

class IDFT : public Discrete {
private:
    inline const Complex <> coefficient(int const i, int const j, int
```

```
        const n) override {
43          return exp(I * 2.0 * M_PI * i * j / n) / n;
44      }
45  };
46
47  class Fast : public FourierAlgorithm {
48  public:
49      bool compute(ComplexVector const & input, ComplexVector & output);
50      bool _compute(Complex <> *, Complex <> *, int n);
51  protected:
52      virtual const Complex <> FFTcoefficient(int const k, int const n)=0;
53  };
54
55  class FFT : public Fast {
56  public:
57      inline const Complex <> FFTcoefficient(int const k, int const n)
            override {
58          Complex<> W( cos(2*M_PI*k / n) , sin(2*M_PI*k / n) );
59          return W;
60      }
61  };
62
63  class IFFT : public Fast {
64  public:
65      inline const Complex <> FFTcoefficient(int const k, int const n)
            override {
66          Complex <> W( cos(-2*M_PI*k / n) , sin(-2*M_PI*k / n) );
67          return W;
68      }
69  };
70
71  #endif    // _FOURIER_H_INCLUDED_
```

fourier.cpp

```
1   #include <iostream>
2   #include <cmath>
3
4   #include "fourier.h"
5
6   bool
7   Discrete::compute(ComplexVector const & input, ComplexVector & output)
8   {
9       size_t n = input.size();
10      output.reserve(n);
11      Complex <> sum = 0;
12      for (size_t i = 0; i < n; ++i) {
13          for (size_t j = 0; j < n; ++j) {
14              sum += input[j] * coefficient(i, j, n);
15          }
16          output.push_back(sum);
17          sum = 0;
18      }
```

```
19      return true;
20  }
21
22  bool
23  Fast::compute(ComplexVector const & input, ComplexVector & output)
24  {
25      bool status;
26      Complex<>* AuxComplexArray = new Complex<>[input.size()];
27      Complex<>* InputAux = new Complex<>[input.size()];
28
29      for(size_t i=0; i<input.size(); i++)    // Para la función recursiva se
            utiliza arreglos
30          InputAux[i] = input[i];           // de complejos, aquí se cargan
               los datos de
31                                      // entrada.
32      status = _compute(InputAux , AuxComplexArray , input.size());
33
34      for(size_t i=0; i<input.size(); ++i)       // Se carga el arreglo de
           complejos
35          output.push_back(AuxComplexArray[i]);  // resultante al vector de
               salida.
36
37      delete[] AuxComplexArray;
38      delete[] InputAux;
39
40      return status;
41  }
42
43  bool
44  Fast::_compute(Complex <> * input, Complex <> * outputEven, int n)
45  {
46
47      if(n==1) {
48          output[0]=input[0];
49          return true;
50      }
51
52      Complex<>* inputEven= new Complex<>[n/2]; //Se crean arreglos de
           complejos para
53      Complex<>* inputOdd= new Complex<>[n/2];  //realizar dividir y
           conquistar
54      Complex<>* outputEven= new Complex<>[n/2];   //Se divide
55      Complex<>* outputOdd= new Complex<>[n/2];
56
57      Complex<> w(1,0);
58      Complex<> wn(cos( -2*M_PI / n ), sin( -2*M_PI / n));
59
60      for (int i = 0; i < (n / 2); i++) {
61          inputEven[i] = input[2 * i];
62          inputOdd[i] = input[2 * i + 1];
63      }
64      _compute(inputEven, outputEven, n/2);
```

```
    _compute(inputOdd, outputOdd, n/2);

    for(int k = 0; k < n/2; ++k){
        output[k] = outputEven[k] + outputOdd[k]* FFTcoefficient(k, n);
        output[k+n/2] = outputEven[k] - (outputOdd[k])* FFTcoefficient(k, n
            );
    }

    delete[] inputEven;
    delete[] inputOdd;
    delete[] outputEven;
    delete[] outputOdd;

    return true;
}
```

Vector.h

```
#ifndef _VECTOR_H_INCLUDED_
#define _VECTOR_H_INCLUDED_

#include <cassert>

template <typename T>
class Vector {
    T* data;
    size_t allocated;
    size_t used;
    const static size_t init_size = 15;
    const static size_t chop_size = 20;
public:
    Vector() : data(new T[init_size]), allocated(init_size), used(0) {
    }
    // Reserva espacio para count elementos
    //
    Vector(size_t count) : data(new T[count]), allocated(count), used(0) {
    }
    // Aloja espacio para count elementos y les asigna el valor value
    //
    Vector(size_t count, T const & value) : data(new T[count]), allocated(
        count), used(count) {
        for (size_t i = 0; i < count; ++i) {
            data[i](value);
        }
    }
    Vector(const Vector& v) : data(new T[v.used]), allocated(v.used), used
        (v.used) {
        for (size_t i = 0; i < used; ++i)
            data[i] = (v.data)[i];
    }
    ~Vector() {
        delete[] data;
    }
```

```
34    Vector& operator=(const Vector& v) {
35       // Check for self-assignment:
36       //
37       if (this == &v)
38          return *this;
39
40       // Same size optimization:
41       //
42       if (used == v.used) {
43          for (size_t i = 0; i < used; ++i)
44             data[i] = (v.data)[i];
45          return *this;
46       }
47       delete[] data;
48       used = v.used;
49       data = new T[used];
50       allocated = used;
51       for (size_t i = 0; i < used; ++i)
52          data[i] = (v.data)[i];
53       return *this;
54    }
55    // versión const:
56    //
57    const T& operator[](size_t position) const {
58       if (position >= used)
59          assert("Illegal position.");
60       return data[position];
61    }
62    // versión no const:
63    //
64    T& operator[](size_t position) {
65       if (position >= used)
66          assert("Illegal position.");
67       return data[position];
68    }
69    size_t size() const {
70       return used;
71    }
72    size_t capacity() const {
73       return allocated;
74    }
75    bool empty() const {
76       return (bool) used;
77    }
78    // agrega un elemento al final:
79    //
80    void push_back(const T& value) {
81       if (used == allocated)
82          reserve(allocated + chop_size);
83       data[used] = value;
84       ++used;
85    }
```

```
86     // llena el vector con count copias de valor value
87     //
88     void assign(size_t count, const T& value) {
89        if (count > allocated)
90           reserve(count);
91        used = count;
92        for (size_t i = 0; i < used; ++i)
93           data[i] = value;
94     }
95     // reserva espacio para new_capacity elementos
96     //
97     void reserve(size_t new_capacity) {
98        if (new_capacity <= allocated)
99           return ;
100       T* new_data = new T[new_capacity];
101       allocated = new_capacity;
102       for (size_t i = 0; i < used; ++i)
103          new_data[i] = data[i];
104       delete[] data;
105       data = new_data;
106    }
107    void clear() {
108       for (size_t i = 0; i < allocated; ++i)
109          data[i] = 0;
110       used = 0;
111    }
112 };
113
114 #endif    //_VECTOR_H_INCLUDED_
```

Complex.h

```
1  #ifndef _COMPLEX_H_INCLUDED_
2  #define _COMPLEX_H_INCLUDED_
3
4  #include <iostream>
5  #include <limits>
6  #include <algorithm>
7  #include <cmath>
8
9  // Función para la comparación con margen de error:
10 template<typename T>
11 inline const bool almostEqual(T a, T b);
12
13 template <typename T = long double>
14 class Complex {
15    T x;
16    T y;
17 public:
18    Complex(T real = 0, T imag = 0) : x(real), y(imag) {}
19    Complex(const Complex& C) : x(C.x), y(C.y) {}
20    ~Complex() {}
21
```

```
22      T re() const { return x; }
23      T im() const { return y; }
24
25      const Complex conj() const {
26          return Complex(x, -y);
27      }
28      const T norm() const {
29          return sqrt(x*x + y*y);
30      }
31      const T arg() const {
32          return std::atan2(y,x);
33      }
34      const Complex operator+() const {
35          return Complex(+x, +y);
36      }
37      const Complex operator-() const {
38          return Complex(-x, -y);
39      }
40      const Complex operator+(const Complex& c) const {
41          return Complex(x + c.x, y + c.y);
42      }
43      const Complex operator-(const Complex& c) const {
44          return Complex(x - c.x, y - c.y);
45      }
46      const Complex operator*(const Complex& c) const {
47          return Complex(x*c.x - y*c.y, y*c.x + x*c.y);
48      }
49      const Complex operator/(const Complex& c) const {
50          return Complex((x*c.x + y*c.y) / (c.x*c.x + c.y*c.y),
51                         (y*c.x - x*c.y) / (c.x*c.x + c.y*c.y));
52      }
53      Complex& operator=(const Complex& c) {
54          x = c.x;
55          y = c.y;
56          return *this;
57      }
58      Complex& operator+=(const Complex& c) {
59          x += c.x;
60          y += c.y;
61          return *this;
62      }
63      Complex& operator-=(const Complex& c) {
64          x -= c.x;
65          y -= c.y;
66          return *this;
67      }
68      Complex& operator*=(const Complex& c) {
69          x = x*c.x - y*c.y;
70          y = y*c.x + x*c.y;
71          return *this;
72      }
73      Complex& operator/=(const Complex& c) {
```

```
74          x = (x*c.x + y*c.y) / (c.x*c.x + c.y*c.y);
75          y = (y*c.x - x*c.y) / (c.x*c.x + c.y*c.y);
76          return *this;
77      }
78      bool operator==(const Complex & c) const {
79          return almostEqual(x, c.x) && almostEqual(y, c.y);
80      }
81      bool operator!=(const Complex& c) const {
82          return !almostEqual(x, c.x) || !almostEqual(y, c.y);
83      }
84      friend std::ostream& operator<<(std::ostream& os, const Complex& c) {
85          return os << '('
86                    << c.x
87                    << ','
88                    << ' '
89                    << c.y
90                    << ')';
91      }
92      friend std::istream& operator>>(std::istream& is, Complex& c) {
93          bool good = false;
94          bool bad = false;
95          T re = 0;
96          T im = 0;
97          char ch;
98          if (is >> ch && ch == '(') {
99              if (is >> re
100             && is >> ch
101             && ch == ','
102             && is >> im
103             && is >> ch
104             && ch == ')')
105                 good = true;
106             else
107                 bad = true;
108         }
109         else if (is.good()) {
110             is.putback(ch);
111             if (is >> re)
112                 good = true;
113             else
114                 bad = true;
115         }
116         if (good) {
117             c.x = re;
118             c.y = im;
119         }
120         else if (bad)
121             is.setstate(std::ios::badbit);
122         return is;
123     }
124 };
125
```

```
126 const Complex <long double> I(0, 1);
127 const long double Complex_acceptableDelta = 10e-6;
128
129 template <typename T> Complex <T>
130 exp(const Complex <T> & c)
131 {
132    return typename Complex<T>::Complex( std::exp (c.re()) * std::cos(c.im
       ()),
133            std::exp(c.re()) * std::sin(c.im()));
134 }
135
136 // Función para la comparación que comprueba si dos números difieren en
    lo
137 // suficientemente poco.
138 //
139 template <typename T> inline const bool
140 almostEqual(T a, T b)
141 {
142    const T absA = std::abs(a);
143    const T absB = std::abs(b);
144    const T absDelta = std::abs(a - b);
145    const bool deltaIsAcceptable = absDelta <= Complex_acceptableDelta;
146
147    if (a == b) // si son iguales, incluso inf
148       return true;
149    // si a o b son cero, o están lo suficientemente cerca
150    //
151    if (a == 0 || b == 0 || deltaIsAcceptable)
152       return true;
153    // sino, usar el error relativo
154    return Complex_acceptableDelta >
155           absDelta / std::min<T>(absA + absB, std::numeric_limits<T>::max
              ());
156 }
157
158 #endif   //_COMPLEX_H_INCLUDED_
```

fourier_test.h

```
1  #ifndef _FOURIER_TEST_H_INCLUDED_
2  #define _FOURIER_TEST_H_INCLUDED_
3
4  #include "cmdline.h"
5  #include "io.h"
6  #include "fourier.h"
7  #include "Complex.h"
8  #include "Vector.h"
9
10 #define DEFAULT_AMOUNT "5000"
11
12 static void opt_number(std::string const &arg);
13 static void opt_help(std::string const &arg);
14
```

```
#endif // _FOURIER_TEST_H_INCLUDED_
```

fourier_test.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <limits>
#include <gtest/gtest.h>

#include "fourier_test.h"

using ComplexVector = Vector <Complex <long double> >;

using namespace std;

#define PRINT(X) \
   std::cerr << #X << ": " << X << std::endl

static char *program_name;
static size_t vectorSize;
static const size_t file_amount = 28;
static FourierTransform* ft;
static FourierTransform* ift;
static const string test_files[file_amount] = {
   "testfiles/Frecuencia1.txt",
   "testfiles/TFrecuencia1.txt",
   "testfiles/Frecuencia1B.txt",
   "testfiles/TFrecuencia1B.txt",
   "testfiles/Frecuencia2.txt",
   "testfiles/TFrecuencia2.txt",
   "testfiles/Frecuencia2B.txt",
   "testfiles/TFrecuencia2B.txt",
   "testfiles/Frecuencia3.txt",
   "testfiles/TFrecuencia3.txt",
   "testfiles/Frecuencia3B.txt",
   "testfiles/TFrecuencia3B.txt",
   "testfiles/Frecuencia4.txt",
   "testfiles/TFrecuencia4.txt",
   "testfiles/Frecuencia4B.txt",
   "testfiles/TFrecuencia4B.txt",
   "testfiles/Frecuencia5.txt",
   "testfiles/TFrecuencia5.txt",
   "testfiles/Frecuencia5B.txt",
   "testfiles/TFrecuencia5B.txt",
   "testfilees/Pulso.txt",
   "testfilees/TPulso.txt",
   "testfilees/PulsoB.txt",
   "testfilees/TPulsoB.txt",
   "testfilees/dwavfs11025.txt",
   "testfilees/Tdwavfs11025.txt",
```

```
50     "testfilees/gwavfs11025.txt",
51     "testfilees/Tgwavfs11025.txt"
52 };
53 
54 static option_t options[] = {
55     {1, "n", "number", DEFAULT_AMOUNT, opt_number, OPT_DEFAULT},
56     {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
57     {0, },
58 };
59 
60 static void
61 opt_number(std::string const &arg)
62 {
63     std::istringstream iss(arg);
64 
65     if (!(iss >> vectorSize) || !iss.eof()) {
66         std::cerr << "Not a possible amount: "
67                   << arg
68                   << "."
69                   << std::endl;
70         exit(1);
71     }
72     if (iss.bad()) {
73         std::cerr << "Cannot read amount."
74                   << std::endl;
75         exit(1);
76     }
77 }
78 
79 static void
80 opt_help(std::string const &arg)
81 {
82     std::cerr << program_name << " [-n amount]"
83               << std::endl;
84     exit(0);
85 }
86 
87 // Google Test exige que las pruebas estén en un namespace sin nombre
88 //
89 namespace {
90     class RandomVectors : public ::testing::Test {
91     protected:
92         RandomVectors() : OrigVector(vectorSize),
93                           FTVector(vectorSize),
94                           FinalVector(vectorSize)
95         {/*
96             cerr << "Esta prueba crea un vector de "
97                  << vectorSize
98                  << " números complejos pseudo-aleatorios "
99                  << "(la cantidad de elementos puede ser cambiada llamando "
100                 << program_name
101                 << " -n <cantidad>). "
```

```
102                 << endl
103                 << "Luego le aplica la DFT, y al vector "
104                 << "resultante le aplica la IDFT, "
105                 << "mediante las funciones utiizadas en el TP. "
106                 << endl
107                 << "Por último, comprueba que el vector y la
                       antitransformada de "
108                 << "su transformada sean iguales."
109                 << endl;
110       */
111           srand(time(NULL));
112           for (size_t i = 0; i < vectorSize; ++i) {
113              long double randA = rand() * rand() * 10000;
114              long double randB = rand() * rand() * 10000;
115              OrigVector.push_back(Complex <long double>(randA, randB));
116           }
117       }
118       ~RandomVectors() {
119           cerr << endl;  // por razones de formato de la impresión
120       }
121       ComplexVector OrigVector;
122       ComplexVector FTVector;
123       ComplexVector FinalVector;
124   };
125
126   class VectorsFromFiles : public ::testing::Test {
127   protected:
128       VectorsFromFiles() : i(0) { /*
129           cerr << "Esta prueba lee vectores y sus transformadas de
                      archivos de prueba "
130                << "y luego las compara a los valores obtenidos al
                       aplicarle "
131                << "la función DFT() e IDFT() usadas en el código del TP."
132                << endl
133                << "Se considera que dos números son iguales si su
                       diferencia es menor o igual a "
134                << Complex_acceptableDelta
135                << endl;
136       */}
137       void read_vectors_from_files() {
138           ifs.open(test_files[i], ios::in);
139           if (!load_signal(ifs, *originalVector))
140              exit(1);
141           ifs.close();
142           ++i;
143           ifs.open(test_files[i], ios::in);
144           if (!load_signal(ifs, *transformedVector))
145              exit(1);
146           ifs.close();
147           ++i;
148       }
149       ~VectorsFromFiles() {
```

```
150             cerr << endl;  // por razones de formato
151         }
152         size_t i;
153         ifstream ifs;
154         ComplexVector *originalVector;
155         ComplexVector *transformedVector;
156         ComplexVector *FTOutput;
157         ComplexVector *IFTOutput;
158     };
159 
160     TEST_F(RandomVectors, DFTandIDFT) {
161         ft->compute(OrigVector, FTVector);
162         ift->compute(FTVector, FinalVector);
163         for (size_t i = 0; i < vectorSize; ++i)
164             EXPECT_EQ(OrigVector[i], FinalVector[i]);
165     }
166 
167     TEST_F(VectorsFromFiles, DFTandIDFT) {
168         while (i < file_amount) {
169             originalVector = new ComplexVector;
170             transformedVector = new ComplexVector;
171             FTOutput = new  ComplexVector;
172             IFTOutput = new ComplexVector;
173             read_vectors_from_files();
174             if (!ft->compute(*originalVector, *FTOutput))
175                 exit(1);
176             for (size_t j = 0 ; j < originalVector->size(); ++j)
177                 EXPECT_EQ((*transformedVector)[j], (*FTOutput)[j]);
178             if (!ift->compute(*transformedVector, *IFTOutput))
179                 exit(1);
180             for (size_t j = 0 ; j < originalVector->size(); ++j)
181                 EXPECT_EQ((*originalVector)[j], (*IFTOutput)[j]);
182             delete transformedVector;
183             delete FTOutput;
184             delete originalVector;
185             delete IFTOutput;
186             /*
187             cerr << test_files[i]
188                  << " fue procesado."
189                  << endl;
190             */
191         }
192     }
193 
194 }  // namespace
195 
196 int main(int argc, char **argv) {
197     program_name = argv[0];
198     cmdline cmdl(options);
199     cmdl.parse(argc, argv);
200 
201     DFT dft;
```

```
202     IDFT idft;
203     FFT fft;
204     IFFT ifft;
205 
206     ::testing::InitGoogleTest(&argc, argv);
207 
208     cerr << "Pruebas para la DFT e IDFT: " << endl;
209     ft = new FourierTransform(&dft);
210     ift = new FourierTransform(&idft);
211     RUN_ALL_TESTS();
212     delete ft;
213     delete ift;
214 /*
215     cerr << "Pruebas para la FFT e IFFT: " << endl;
216     ft = new FourierTransform(&fft);
217     ift = new FourierTransform(&ifft);
218     RUN_ALL_TESTS()
219     delete ft;
220     delete ift;
221 */
222     return 0;
223 }
```

Complex_test.h

```
1  #ifndef _COMPLEX_TEST_H_INCLUDED_
2  #define _COMPLEX_TEST_H_INCLUDED_
3  
4  #include "cmdline.h"
5  #include "Complex.h"
6  #include "Vector.h"
7  
8  #define DEFAULT_AMOUNT "1000000"
9  
10 static void opt_number(std::string const &arg);
11 static void opt_help(std::string const &arg);
12 
13 #endif // _COMPLEX_TEST_H_INCLUDED_
```

Complex_test.cpp

```
1  #include <iostream>
2  #include <sstream>
3  #include <cstdlib>
4  #include <ctime>
5  #include <complex>
6  #include <gtest/gtest.h>
7  
8  #include "Complex_test.h"
9  
10 static char *program_name;
11 static size_t vectorSize;
12 
13 static option_t options[] = {
```

```
14     {1, "n", "number", DEFAULT_AMOUNT, opt_number, OPT_DEFAULT},
15     {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
16     {0, },
17 };
18 
19 static void
20 opt_number(std::string const &arg)
21 {
22     std::istringstream iss(arg);
23 
24     if (!(iss >> vectorSize) || !iss.eof()) {
25         std::cerr << "Not a possible amount: "
26                   << arg
27                   << "."
28                   << std::endl;
29         exit(1);
30     }
31     if (iss.bad()) {
32         std::cerr << "Cannot read amount."
33                   << std::endl;
34         exit(1);
35     }
36 }
37 
38 static void
39 opt_help(std::string const &arg)
40 {
41     std::cout << program_name
42               << " [-n amount]"
43               << std::endl;
44     exit(0);
45 }
46 
47 // Definidos ad hoc para estas pruebas:
48 //
49 template <typename T> inline bool
50 operator==(std::complex <T> const & std, Complex <T> const & own)
51 {
52     // la conversión está pues necesita usar la comparación con tolerancia
        ,
53     // definida como Complex::operator==()
54     Complex <T> own_from_std(std.real(), std.imag());
55     return own == own_from_std;
56 }
57 
58 template <typename T> inline bool
59 operator==(Complex <T> const & own, std::complex <T> const & std)
60 {
61     Complex <T> own_from_std(std.real(), std.imag());
62     return own_from_std == own;
63 }
64 
```

```
65 // Google Test exige que las pruebas estén en un namespace sin nombre
66 //
67 namespace {
68    // Clase a reutilizar en múltiples pruebas con llamados a TEST_F()
69    class ComplexTest : public ::testing::Test {
70    protected:
71       ComplexTest() : stdComplex(vectorSize), myComplex(vectorSize) {
72          srand(time(NULL));
73          for (size_t i = 0; i < vectorSize; ++i) {
74             long double randA = rand() * rand() * 10000;
75             long double randB = rand() * rand() * 10000;
76             stdComplex.push_back(std::complex <long double>(randA, randB)
                  );
77             myComplex.push_back(Complex <long double>(randA, randB));
78          }
79       }
80       Vector <std::complex <long double> > stdComplex;
81       Vector <Complex <long double> > myComplex;
82    };
83
84    // Probar que Complex::Complex() funcione correctamente
85    TEST_F(ComplexTest, Constructor) {
86       for (size_t i = 0; i < vectorSize; ++i)
87          EXPECT_EQ(stdComplex[i], myComplex[i]);
88    }
89
90    // Probar que operator+() funcione
91    TEST_F(ComplexTest, Addition) {
92       for (size_t i = 0, j = vectorSize -1; i < vectorSize && j >= 0; ++i
          , --j)
93          EXPECT_EQ(stdComplex[i] + stdComplex[j], myComplex[i] +
             myComplex[j]);
94    }
95
96    // Probar que operator-() funcione
97    TEST_F(ComplexTest, Substraction) {
98       for (size_t i = 0, j = vectorSize -1; i < vectorSize && j >= 0; ++i
          , --j)
99          EXPECT_EQ(stdComplex[i] - stdComplex[j], myComplex[i] -
             myComplex[j]);
100   }
101
102   // Probar que operator*() funcione
103   TEST_F(ComplexTest, Multiplication) {
104      for (size_t i = 0, j = vectorSize -1; i < vectorSize && j >= 0; ++i
          , --j)
105         EXPECT_EQ(stdComplex[i] * stdComplex[j], myComplex[i] *
             myComplex[j]);
106   }
107
108   // Probar que operator/() funcione
109   TEST_F(ComplexTest, Division) {
```

```
110         for (size_t i = 0, j = vectorSize -1; i < vectorSize && j >= 0; ++i
                , --j)
111             EXPECT_EQ(stdComplex[i] / stdComplex[j], myComplex[i] /
                    myComplex[j]);
112     }
113
114     // Probar que Complex::conj() funcione
115     TEST_F(ComplexTest, Conj) {
116         for (size_t i = 0; i < vectorSize; ++i)
117             EXPECT_EQ(std::conj(stdComplex[i]), myComplex[i].conj());
118     }
119
120     // Probar que Complex::norm() funcione
121     TEST_F(ComplexTest, Norm) {
122         for (size_t i = 0; i < vectorSize; ++i)
123             EXPECT_DOUBLE_EQ(std::abs(stdComplex[i]), myComplex[i].norm());
124     }
125
126     // Probar que Complex::arg() funcione
127     TEST_F(ComplexTest, Arg) {
128         for (size_t i = 0; i < vectorSize; ++i)
129             EXPECT_DOUBLE_EQ(std::arg(stdComplex[i]), myComplex[i].arg());
130     }
131
132     // Probar que exp() funcione
133     TEST_F(ComplexTest, Exp) {
134         for (size_t i = 0; i < vectorSize; ++i)
135             EXPECT_EQ(std::exp(stdComplex[i]), exp(myComplex[i]));
136     }
137
138     // Probar que la representación polar es el mismo número
139     TEST_F(ComplexTest, Polar) {
140         for (size_t i = 0; i < vectorSize; ++i)
141             EXPECT_EQ(myComplex[i], Complex<long double>(myComplex[i].norm()
                    ) \
142                     * exp(I * Complex<long double>(myComplex[i].arg())));
143     }
144 }  // namespace
145
146 int main(int argc, char **argv) {
147     program_name = argv[0];
148     cmdline cmdl(options);
149     cmdl.parse(argc, argv);
150
151     std::cerr << "Esta prueba crea dos vectores de "
152                     << vectorSize
153                     << " números complejos pseudo-aleatorios"
154                     << "(la cantidad de elementos puede ser cambiada
                            llamando "
155                     << program_name
156                     << " -n <cantidad>): uno de ellos usando la "
157                     << "clase std::complex y el otro usando la clase
```

```
                        Complex"
158                     << " ñ diseada para el TP."
159                     << std::endl
160                     << "El programa aplica las diversas funciones
                        asociadas "
161                     << "a la clase a cada número y compara lo obtenido "
162                     << "con lo obtenido del llamado de las funciones aná
                        logas"
163                     <<  "asociadas a la clase std::complex."
164                     << std::endl;
165
166     ::testing::InitGoogleTest(&argc, argv);
167     return RUN_ALL_TESTS();
168 }
```