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 14 de Junio 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 (IFFT).

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, eivtando la duplicación de código y 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 implementación de la clase Complex como plantilla permite utilizar el tipo de dato que se desee para armar el complejo, según la precisión que se requiera en la aplicación en la que se lo va a utilizar. Por otro lado, también para 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 que luego se ordenan progresivamente (según el orden de entrada) dentro de la clase Vector.

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 y jerarquía de clases

Se utilizó para este proyecto el patrón de diseño Strategy. Esto permite reutilizar código y proporciona al usuario una misma interfaz que, gracias a la implementación realizada del método, 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.

Resulta importante detallar la jerarquía de clases de la implementación realizada a partir de este patrón de diseño. Se implementaron las clases abstractas FourierTransform y FourierAlgorithm, las cuales indican que se las distintas implementaciones de la trasnformación de Fourier pueden utilizar el método Compute pero no definen su implementación. Dichas clases abstractas, que no pueden ser instanceadas, se usan como clases bases para Discrete y Fast, que son subclases de la clase FourierAlgorithm, y definen su propia implementación de la función Compute que se utiliza para realizar la transformación. A su vez, tanto Discrete como Fast definen que sus clases hijas van a utilizar el método Coefficient, pero la implementación de dicha función la realiza la clase que hereda.

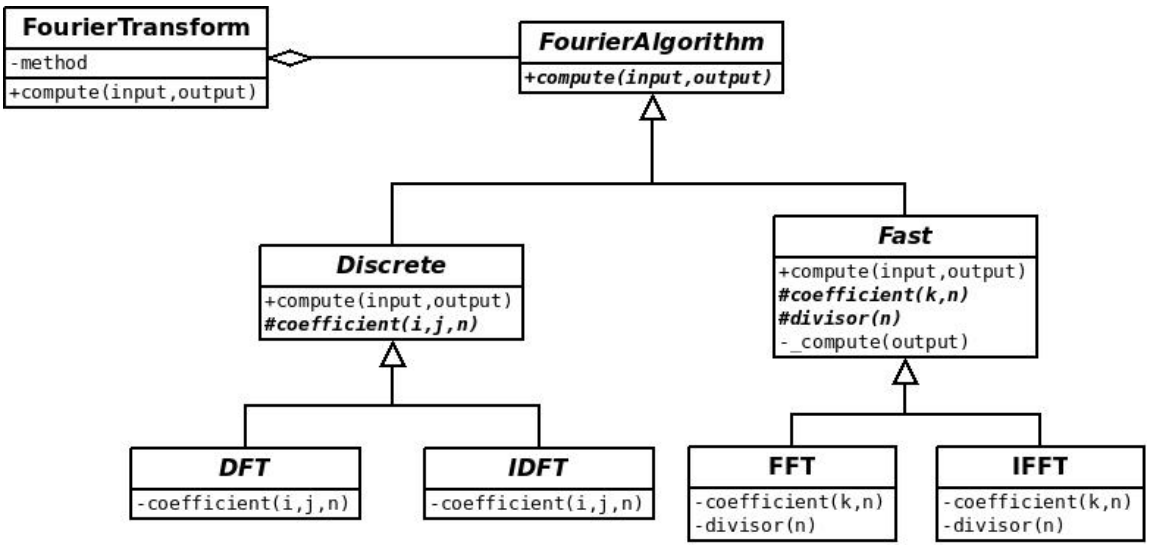

Figura 1: Diagrama de clases según el lenguaje unificado de modelado UML

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 $O(n.lg(n))$, en lugar de $O(n^2)$, siendo n el número de muestras de la señal de entrada a transformar. 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, a partir de comparaciones de los tiempos de ejecución de cada prueba, 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()*.

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 presentaron diferentes 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, se utilizó como herramienta Valgrind para poder verificar que no hubiera fugas 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

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
	$(CXX) $(CXXFLAGS) $(CXXARGS) -c $(TESTS)/fourier_test.cpp -o fourier_test.o

fourier_test.exe: fourier_test.o gtest-all.o cmdline.o
	$(CXX) $(CXXFLAGS) $(CXXARGS) gtest-all.o fourier_test.o fourier.o cmdline.o io.o -o fourier_test.exe

clean:
	$(RM) -vf *.o *.exe *.t *.out *.err
```

Código fuente

main.h

```
#ifndef _MAIN_H_INCLUDED_
#define _MAIN_H_INCLUDED_

#include "cmdline.h"
#include "Complex.h"
#include "Vector.h"
#include "io.h"
#include "fourier.h"

static void opt_input(std::string const &);
static void opt_output(std::string const &);
static void opt_method(std::string const &);
static void opt_help(std::string const & = "");
FourierAlgorithm* choose_method(std::string read_method);

#endif   // _MAIN_H_INCLUDED_
```

main.cpp

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

#include "main.h"

```

```
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", nullptr, opt_help, OPT_DEFAULT},
15    {0, },
16 };
17
18 static char *program_name;
19 static FourierTransform *transform;
20 static istream *iss = nullptr;
21 static ostream *oss = nullptr;
22 static fstream ifs;
23 static fstream ofs;
24
25 static void
26 opt_input(string const &arg)
27 {
28    if (arg == "-") {
29       iss = &cin;
30    }
31    else {
32       ifs.open(arg.c_str(), ios::in);
33       iss = &ifs;
34    }
35
36    if (!iss->good()) {
37       cerr << "Cannot open "
38            << arg
39            << "."
40            << endl;
41       exit(1);
42    }
43 }
44
45 static void
46 opt_output(string const &arg)
47 {
48    if (arg == "-") {
49       oss = &cout;
50    } else {
51       ofs.open(arg.c_str(), ios::out);
52       oss = &ofs;
53    }
54
55    if (!oss->good()) {
56       cerr << "Cannot open "
57            << arg
58            << "."
59            << endl;
```

```
60          exit(1);
61      }
62  }
63
64  static void
65  opt_method(string const &arg)
66  {
67      istringstream iss(arg);
68      string read_method;
69
70      if (!(iss >> read_method) || iss.bad()) {
71          cerr << "Cannot read method."
72               << endl;
73          exit(1);
74      }
75
76      FourierAlgorithm *chosen_method = choose_method(read_method);
77
78      if (chosen_method == nullptr) {
79          cerr << "Not a posible method: "
80               << arg
81               << "."
82               << endl;
83          opt_help();
84          exit(1);
85      }
86      ::transform = new FourierTransform(chosen_method);
87      if (!::transform)
88          exit(1);
89  }
90
91  static void
92  opt_help(string const & arg)
93  {
94      cout << "Usage: "
95           << program_name
96           << " [-m FFT | IFFT | DFT | IDFT] [-i file] [-o file]"
97           << endl;
98      exit(0);
99  }
100
101 FourierAlgorithm *
102 choose_method (string read_method)
103 {
104     if (read_method == "FFT")
105         return new FFT;
106     if (read_method == "IFFT")
107         return new IFFT;
108     if (read_method == "DFT")
109         return new DFT;
110     if (read_method == "IDFT")
111         return new IDFT;
```

```
112     return nullptr;
113 }
114 
115 int
116 main(int argc, char * const argv[])
117 {
118     program_name = argv[0];
119     cmdline cmdl(options);
120     cmdl.parse(argc, argv);
121 
122     // Cuestiones de formato para la impresión:
123     oss->setf(ios::fixed, ios::floatfield);
124     oss->precision(6);
125 
126     bool status = process(*::transform, *iss, *oss);
127 
128     delete ::transform;
129     ::transform = nullptr;
130 
131     return status;
132 }
```

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 
```

```
#endif
```

cmdline.cpp

```
#include <string>
#include <cstdlib>
#include <iostream>

#include "cmdline.h"

using namespace std;

cmdline::cmdline()
{
}

cmdline::cmdline(option_t *table) : option_table(table)
{
}

void
cmdline::parse(int argc, char * const argv[])
{
#define END_OF_OPTIONS(p)       \
   ((p)->short_name == 0    \
    && (p)->long_name == 0 \
    && (p)->parse == 0)
   for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op)
      op->flags &= ~OPT_SEEN;
   for (int i = 1; i < argc; ++i) {
      if (argv[i][0] != '-') {
         cerr << "Invalid non-option argument: "
              << argv[i]
              << endl;
         exit(1);
      }
      if (argv[i][1] == '-'
          && argv[i][2] == 0)
         break;
      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]);
   }
   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)
```

```
50                << " is mandatory."
51                << "\n";
52            exit(1);
53        }
54        if (op->def_value == 0)
55            continue;
56        op->parse(string(op->def_value));
57    }
58}
59
60int
61cmdline::do_long_opt(const char *opt, const char *arg)
62{
63    for (option_t *op = option_table; op->long_name != 0; ++op) {
64        if (string(opt) == string(op->long_name)) {
65            op->flags |= OPT_SEEN;
66
67            if (op->has_arg) {
68                if (arg == 0) {
69                    cerr << "Option requires argument: "
70                         << "--"
71                         << opt
72                         << "\n";
73                    exit(1);
74                }
75                op->parse(string(arg));
76                return 1;
77            } else {
78                op->parse(string(""));
79                return 0;
80            }
81        }
82    }
83    cerr << "Unknown option: "
84         << "--"
85         << opt
86         << "."
87         << endl;
88    exit(1);
89    return -1;
90}
91
92int
93cmdline::do_short_opt(const char *opt, const char *arg)
94{
95    option_t *op;
96    for (op = option_table; op->short_name != 0; ++op) {
97        if (string(opt) == string(op->short_name)) {
98            op->flags |= OPT_SEEN;
99            if (op->has_arg) {
100                if (arg == 0) {
101                    cerr << "Option requires argument: "
```

```
                    << "-"
                    << opt
                    << "\n";
                exit(1);
            }
            op->parse(string(arg));
            return 1;
        } else {
            op->parse(string(""));
            return 0;
        }
    }
  }
  cerr << "Unknown option: "
       << "-"
       << opt
       << "."
       << endl;
  exit(1);
  return -1;
}
```

io.h

```
#ifndef _IO_H_INCLUDED_
#define _IO_H_INCLUDED_

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

bool load_signal(std::istream &, Vector<Complex<> > &);
bool print_signal(std::ostream &, Vector<Complex<> > const &);
void print_msg_and_exit(std::string const & msg);
bool process(FourierTransform& transform, std::istream& is, std::ostream&
    os);

#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)
```

```
14          input.push_back(c);
15      if (is.bad())
16          return false;
17      return true;
18  }
19
20  bool
21  print_signal(ostream & os, Vector<Complex<> > const & output)
22  {
23      for (size_t i = 0; i < output.size(); ++i)
24          os << output[i] << ' ';
25      os << endl;
26      if (os.bad())
27          return false;
28      return true;
29  }
30
31  void
32  print_msg(string const & msg)
33  {
34      cerr << msg
35           << endl;
36      exit(1);
37  }
38
39  bool
40  process(FourierTransform& transform, istream& is, ostream& os)
41  {
42      ComplexVector inSignal;
43      ComplexVector outSignal;
44      istringstream line;
45      string s;
46      bool status;
47
48      for (int lineNo = 1; getline(is, s); ++lineNo) {
49          if (is.bad()) {
50              print_msg("An error occured while processing line " + to_string(
                    lineNo) + ".");
51              return false;
52          }
53
54          line.clear(); // vacía los flags del istringstream.
55          line.str(s);
56
57          status = load_signal(line, inSignal);
58          if (!status) {
59              print_msg("Error processing \"" + line.str() + "\" (line " +
                    to_string(lineNo) + ").");
60              return false;
61          }
62
63          status = transform.compute(inSignal, outSignal);
```

```
64        if (!status) {
65            print_msg("An error occured while performing the requested
                  operation.");
66            return false;
67        }
68
69        status = print_signal(os, outSignal);
70        if (!status) {
71            print_msg("Cannot write to output stream.");
72            return false;
73        }
74
75        // vacía los vectores para reutilizarlos en el siguiente ciclo.
76
77        inSignal.clear();
78        outSignal.clear();
79    }
80    return true;
81 }
```

fourier.h

```
1  #ifndef _FOURIER_H_INCLUDED_
2  #define  _FOURIER_H_INCLUDED_
3
4  #include "Complex.h"
5  #include "Vector.h"
6
7  using ComplexVector = Vector <Complex <long double> >;
8
9  class FourierAlgorithm {
10 public:
11     virtual bool compute(ComplexVector const & input, ComplexVector &
           output) = 0;
12     virtual ~FourierAlgorithm() {}
13 };
14
15 class FourierTransform {
16 public:
17     FourierTransform(FourierAlgorithm *method) : _method(method) {}
18     virtual ~FourierTransform() {
19         delete _method;
20     }
21     bool compute(ComplexVector const & input, ComplexVector & output) {
22         return _method? _method->compute(input, output) : false;
23     }
24 private:
25     FourierAlgorithm *_method;
26 };
27
28 class Discrete : public FourierAlgorithm {
29 public:
30     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 {
    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 {
    const Complex <> _coefficient(int const i, int const j, int const n)
        override {
        return exp(I * 2.0 * M_PI * i * j / n) / n;
    }
};

class Fast : public FourierAlgorithm {
public:
    bool compute(ComplexVector const & input, ComplexVector & output);
protected:
    virtual const Complex <> _coefficient(int const k, int const n) = 0;
    virtual const size_t _divisor(size_t const n) = 0;
private:
    ComplexVector _compute(ComplexVector const & input);
};

class FFT : public Fast {
    const Complex <> _coefficient(int const k, int const n) override {
        return exp(-I * 2.0 * M_PI * k / n);
    }
    const size_t _divisor(size_t const n) {
        return 1;
    }
};

class IFFT : public Fast {
    const Complex <> _coefficient(int const k, int const n) override {
        return exp(I * 2.0 * M_PI * k / n);
    }
    const size_t _divisor(size_t const n) {
        return n;
    }
};


#endif    // _FOURIER_H_INCLUDED_
```

fourier.cpp

```
#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    output.clear();
26    size_t n = input.size();
27
28    if (n & (n - 1)) { // si el ñtamao no es una potencia de dos...
29       ComplexVector auxInput(input);
30       while (n & (n - 1)) {
31          auxInput.push_back(0); // ...rellenar con ceros hasta que lo sea
               .
32          ++n;
33       }
34       output = _compute(auxInput);
35    }
36    else
37       output = _compute(input);
38
39    for (size_t i = 0; i < output.size(); ++i)
40       output[i] /= _divisor(output.size());
41
42    return true;
43 }
44
45 ComplexVector
46 Fast::_compute(ComplexVector const & input)
47 {
48    size_t n = input.size();
49    if (n <= 1) {
50       return ComplexVector(input);
51    }
52
```

```
    ComplexVector inputEven(n/2);
    ComplexVector inputOdd(n/2);

    for (size_t i = 0; i < n / 2; ++i) {
        inputEven[i] = input[2 * i];
        inputOdd[i] = input[2 * i + 1];
    }

    ComplexVector outputEven(_compute(inputEven));
    ComplexVector outputOdd(_compute(inputOdd));

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

    return output;
}
```

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 y les asigna el valor value:
    //
    Vector(size_t count, T const & value = 0) : data(new T[count]{value}),
         allocated(count), used(count) {
    }
    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;
        data = nullptr;
    }
    Vector& operator=(Vector const & v) {
```

```
29          // Check for self-assignment:
30          //
31          if (this == &v)
32             return *this;
33
34          // Same size optimization:
35          //
36          if (used == v.used) {
37             for (size_t i = 0; i < used; ++i)
38                data[i] = (v.data)[i];
39             return *this;
40          }
41          delete[] data;
42          used = v.used;
43          data = new T[used];
44          allocated = used;
45          for (size_t i = 0; i < used; ++i)
46             data[i] = (v.data)[i];
47          return *this;
48       }
49       // versión const:
50       //
51       const T& operator[](size_t position) const {
52          if (position >= used)
53             assert("Illegal position.");
54          return data[position];
55       }
56       // versión no const:
57       //
58       T& operator[](size_t position) {
59          if (position >= used)
60             assert("Illegal position.");
61          return data[position];
62       }
63       size_t size() const {
64          return used;
65       }
66       size_t capacity() const {
67          return allocated;
68       }
69       bool empty() const {
70          return (bool) used;
71       }
72       // agrega un elemento al final:
73       //
74       void push_back(T const & value) {
75          if (used == allocated)
76             reserve(allocated + chop_size);
77          data[used] = value;
78          ++used;
79       }
80       // llena el vector con count copias de valor value
```

```
    //
    void assign(size_t count, T const & value) {
       if (count > allocated)
          reserve(count);
       used = count;
       for (size_t i = 0; i < used; ++i)
          data[i] = value;
    }
    // reserva espacio para new_capacity elementos
    //
    void reserve(size_t new_capacity) {
       if (new_capacity > allocated) {
          T* new_data = new T[new_capacity];
          allocated = new_capacity;
          for (size_t i = 0; i < used; ++i)
             new_data[i] = data[i];
          delete[] data;
          data = new_data;
       }
    }
    void clear() {
       for (size_t i = 0; i < allocated; ++i)
          data[i] = 0;
       used = 0;
    }
};

#endif    //_VECTOR_H_INCLUDED_
```

Complex.h

```
#ifndef _COMPLEX_H_INCLUDED_
#define _COMPLEX_H_INCLUDED_

#include <iostream>
#include <limits>
#include <algorithm>
#include <cmath>

// Función para la comparación con margen de error:
template<typename T>
inline const bool almostEqual(T a, T b);

template <typename T = long double>
class Complex {
    T x;
    T y;
public:
    Complex(T real = 0, T imag = 0) : x(real), y(imag) {}
    Complex(const Complex& C) : x(C.x), y(C.y) {}
    ~Complex() {}

    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 #define DEFAULT_METHOD "fast"
12
13 static void opt_number(std::string const & arg);
14 static void opt_method(std::string const & arg);
15 static void opt_help(std::string const & arg = "");
```

```
16
17 #endif // _FOURIER_TEST_H_INCLUDED_
```

fourier_test.cpp

```
1  #include <iostream>
2  #include <fstream>
3  #include <cstdlib>
4  #include <ctime>
5  #include <string>
6  #include <limits>
7  #include <gtest/gtest.h>
8
9  #include "fourier_test.h"
10
11 using ComplexVector = Vector <Complex <long double> >;
12
13 using namespace std;
14
15 static char *program_name;
16 static size_t vectorSize;
17 static const size_t file_amount = 28;
18 static FourierTransform* ft;
19 static FourierTransform* ift;
20 static FourierAlgorithm* chosen_method;
21 static FourierAlgorithm* chosen_inverse_method;
22 static const string test_files[file_amount] = {
23    "testfiles/Frecuencia1.txt",
24    "testfiles/TFrecuencia1.txt",
25    "testfiles/Frecuencia1B.txt",
26    "testfiles/TFrecuencia1B.txt",
27    "testfiles/Frecuencia2.txt",
28    "testfiles/TFrecuencia2.txt",
29    "testfiles/Frecuencia2B.txt",
30    "testfiles/TFrecuencia2B.txt",
31    "testfiles/Frecuencia3.txt",
32    "testfiles/TFrecuencia3.txt",
33    "testfiles/Frecuencia3B.txt",
34    "testfiles/TFrecuencia3B.txt",
35    "testfiles/Frecuencia4.txt",
36    "testfiles/TFrecuencia4.txt",
37    "testfiles/Frecuencia4B.txt",
38    "testfiles/TFrecuencia4B.txt",
39    "testfiles/Frecuencia5.txt",
40    "testfiles/TFrecuencia5.txt",
41    "testfiles/Frecuencia5B.txt",
42    "testfiles/TFrecuencia5B.txt",
43    "testfilees/Pulso.txt",
44    "testfilees/TPulso.txt",
45    "testfilees/PulsoB.txt",
46    "testfilees/TPulsoB.txt",
47    "testfilees/dwavfs11025.txt",
48    "testfilees/Tdwavfs11025.txt",
```

```
49     "testfilees/gwavfs11025.txt",
50     "testfilees/Tgwavfs11025.txt"
51 };
52 
53 static option_t options[] = {
54     {1, "n", "number", DEFAULT_AMOUNT, opt_number, OPT_DEFAULT},
55     {1, "m", "method", DEFAULT_METHOD, opt_method, 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_method(std::string const & arg)
81 {
82     if (arg == "fast") {
83         chosen_method = new FFT;
84         chosen_inverse_method = new IFFT;
85     }
86     else if (arg == "discrete") {
87         chosen_method = new DFT;
88         chosen_inverse_method = new IDFT;
89     }
90     else {
91         std::cerr << "Not a possible method: "
92                   << arg
93                   << "."
94                   << std::endl;
95         opt_help();
96         exit(1);
97     }
98 }
99 
100 static void
```

```
opt_help(std::string const & arg)
{
   std::cout << "Usage: "
             << program_name
             << " [-n amount]"
             << " [-m fast | discrete]"
             << std::endl;
   exit(0);
}

// Google Test exige que las pruebas estén en un namespace sin nombre
//
namespace {
   class RandomVectors : public ::testing::Test {
   protected:
      RandomVectors() : OrigVector(vectorSize),
                        FTVector(vectorSize),
                        FinalVector(vectorSize)
      {
         srand(time(NULL));
         for (size_t i = 0; i < vectorSize; ++i) {
            long double randA = rand() * rand() * 10000;
            long double randB = rand() * rand() * 10000;
            OrigVector.push_back(Complex <long double>(randA, randB));
         }
      }
      ~RandomVectors() {
      }
      ComplexVector OrigVector;
      ComplexVector FTVector;
      ComplexVector FinalVector;
   };

   class VectorsFromFiles : public ::testing::Test {
   protected:
      VectorsFromFiles() : i(0) {
      }
      void read_vectors_from_files() {
         ifs.open(test_files[i], ios::in);
         if (!load_signal(ifs, *originalVector))
            exit(1);
         ifs.close();
         ++i;
         ifs.open(test_files[i], ios::in);
         if (!load_signal(ifs, *transformedVector))
            exit(1);
         ifs.close();
         ++i;
      }
      ~VectorsFromFiles() {
      }
      size_t i;
```

```
153         ifstream ifs;
154         ComplexVector *originalVector;
155         ComplexVector *transformedVector;
156         ComplexVector *FTOutput;
157         ComplexVector *IFTOutput;
158     };
159 
160     TEST_F(RandomVectors, FTandIFT) {
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, FTandIFT) {
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     }
188 
189 }  // namespace
190 
191 int main(int argc, char **argv) {
192     program_name = argv[0];
193     cmdline cmdl(options);
194     cmdl.parse(argc, argv);
195 
196     int test_result;
197 
198     ::testing::InitGoogleTest(&argc, argv);
199 
200     cout << "En la primera prueba se crea un vector de "
201          << vectorSize
202          << " números complejos pseudo-aleatorios "
203          << "(la cantidad de elementos puede ser cambiada llamando "
204          << program_name
```

```
205         << " -n <cantidad>). "
206         << endl
207         << "Luego se le aplica la transformada elegida "
208         << "(por defecto se usan la FFT e IFFT; "
209         << "la transformada a utilizar puede  ser cambiada llamando "
210         << program_name
211         << " -m <fast|discrete>), y al vector resultante se le aplica "
212         << "la antitransformada."
213         << endl
214         << "Por último, se comprueba que el vector original y "
215         << "la antitransformada de su transformada sean iguales."
216         << endl;
217
218    cout << "En la segunda prueba se lee vectores y sus transformadas "
219         << "de archivos y luego se los compara a los valores obtenidos "
220         << "al aplicarles la transformada elegida."
221         << endl
222         << "Se considera que dos números son iguales si "
223         << "el módulo de su diferencia es menor o igual a "
224         << Complex_acceptableDelta
225         << "."
226         << endl;
227
228    ft = new FourierTransform(chosen_method);
229    ift = new FourierTransform(chosen_inverse_method);
230
231    test_result = RUN_ALL_TESTS();
232
233    delete ft;
234    delete ift;
235    return test_result;
236 }
```

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         std::vector <std::complex <long double> > stdComplex;
81         std::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             if(stdComplex[j] != 0.0l && myComplex[j] != 0.0l)
112                 EXPECT_EQ(stdComplex[i] / stdComplex[j],
113                           myComplex[i] / myComplex[j]);
114     }
115
116     // Probar que Complex::conj() funcione
117     TEST_F(ComplexTest, Conj) {
118         for (size_t i = 0; i < vectorSize; ++i)
119             EXPECT_EQ(std::conj(stdComplex[i]), myComplex[i].conj());
120     }
121
122     // Probar que Complex::norm() funcione
123     TEST_F(ComplexTest, Norm) {
124         for (size_t i = 0; i < vectorSize; ++i)
125             EXPECT_DOUBLE_EQ(std::abs(stdComplex[i]), myComplex[i].norm());
126     }
127
128     // Probar que Complex::arg() funcione
129     TEST_F(ComplexTest, Arg) {
130         for (size_t i = 0; i < vectorSize; ++i)
131             EXPECT_DOUBLE_EQ(std::arg(stdComplex[i]), myComplex[i].arg());
132     }
133
134     // Probar que exp() funcione
135     TEST_F(ComplexTest, Exp) {
136         for (size_t i = 0; i < vectorSize; ++i)
137             EXPECT_EQ(std::exp(stdComplex[i]), exp(myComplex[i]));
138     }
139
140     // Probar que la representación polar es el mismo número
141     TEST_F(ComplexTest, Polar) {
142         for (size_t i = 0; i < vectorSize; ++i)
143             EXPECT_EQ(myComplex[i], Complex<long double>(myComplex[i].norm()
                    )
144                     * exp(I * Complex<long double>(myComplex[i].arg())));
145     }
146 }   // namespace
147
148 int main(int argc, char **argv) {
149     program_name = argv[0];
```

```
150     cmdline cmdl(options);
151     cmdl.parse(argc, argv);
152 
153     std::cerr << "Esta prueba crea dos vectores de "
154               << vectorSize
155               << " números complejos pseudo-aleatorios"
156               << "(la cantidad de elementos puede ser cambiada llamando "
157               << program_name
158               << " -n <cantidad>): uno de ellos usando la "
159               << "clase std::complex y el otro usando la clase Complex "
160               << "ñdiseada para el TP."
161               << std::endl
162               << "El programa aplica las diversas funciones asociadas "
163               << "a la clase a cada número y compara lo obtenido "
164               << "con lo obtenido del llamado de las funciones análogas"
165               << " asociadas a la clase std::complex."
166               << std::endl;
167 
168     ::testing::InitGoogleTest(&argc, argv);
169     return RUN_ALL_TESTS();
170 }
```