FACULTAD DE INGENIERÍA
DE LA
UNIVERSIDAD DE BUENOS AIRES

Algoritmos y Programación II [95.12]

Trabajo Práctico n.° 0:

Programación C++

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 10 de Mayo de 2018

Introducción

En este trabajo se busca obtener conocimientos de programación en el lenguaje C++. Para ello se realiza un programa en dicho lenguaje implementando la transformada discreta de Fourier (DFT) 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 la DFT y la IDFT. 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.

Se definió un tipo de dato enumerativo para indicar al programa si debe realizar la DFT o la IDFT variando en la función de la transformada el coeficiente en el exponente, y en el caso de la IDFT multiplicando por la inversa de N. Con esto se evita repetir el código en cada transformada.

Por último, además de las dos clases creadas, 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.

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 o IDFT). En caso de que esta opción no sea indicada, el programa realiza la DFT 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

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

A partir de las pruebas realizadas, se detectaron mínimas fallas en la función exponencial de la clase Complex (2 en un millón de números) al cargar el vector con una extrema cantidad de complejos. Al corregir dicho error, el tiempo de ejecución de las funciones afectadas se duplicó. A pesar de que la falla podría considerarse despreciable, se corrigió procurando una mayor robustez y resignando eficiencia.

Posibles mejoras

El aspecto principal a mejorar es la eficiencia del algoritmo diseñado para calcular la transformada discreta de Fourier y su inversa, dado que este se implementó de la forma más simple posible, utilizando una iteración muy ineficiente. Se consideraron distintas alternativas para mejorarlo, como utilizar una matriz que contenga las exponenciales complejas utilizadas en la transformación y de esta forma evitar realizar dichas operaciones de forma repetida. Además, investigando y estudiando dicha matriz algebraicamente, se encontró que esta resulta una matriz hermética, lo cual simplificaría su construcción disminuyendo más aún la cantidad de operaciones necesarias para la transformación.

Por otro lado, la DFT se podría realizar de manera más eficiente utilizando el algoritmo de

la FFT, Fast Fourier Transform. Este algoritmo fue extensamente estudiado y perfeccionado, y es comunmente utilizado para realizar esta transformada.

Conclusiones

EL programa cumple su función de manera eficaz, aunque se podría realizar ciertas mejoras para que sea más eficiente. 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.

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 DFT_test.exe

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

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

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

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

```

```
gtest-all.o:
	$(CXX) $(CXXFTESTLAGS) -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

DFT_test.o: $(TESTS)/DFT_test.cpp $(INCLUDE)/DFT_test.h $(INCLUDE)/
	Complex.h $(INCLUDE)/Vector.h $(INCLUDE)/io.h $(INCLUDE)/cmdline.h $(
	INCLUDE)/DFT.h
	$(CXX) $(CXXFLAGS) $(CXXARGS) -c $(TESTS)/DFT_test.cpp -o DFT_test.o

DFT_test.exe: DFT_test.o gtest-all.o cmdline.o
	$(CXX) $(CXXFLAGS) $(CXXARGS) gtest-all.o DFT_test.o DFT.o cmdline.o
	   io.o -o DFT_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 "DFT.h"
#include "io.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 &);
static void print_msg_and_exit(std::string const &);

#endif   // _MAIN_H_INCLUDED_
```

main.cpp

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

```
5  #include "main.h"
6
7  using namespace std;
8  using namespace DFT;
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", "DFT", opt_method, OPT_DEFAULT},
14    {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
15    {0, },
16 };
17
18 static char *program_name;
19 static TransformType chosen_method;
20 static istream *iss = NULL;
21 static ostream *oss = NULL;
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)
71          || !iss.eof()) {
72          cerr << "Not a posible method: "
73              << arg
74              << "."
75              << endl;
76          exit(1);
77      }
78      if (iss.bad()) {
79          cerr << "Cannot read method."
80              << endl;
81          exit(1);
82      }
83      if (read_method == "-" || read_method == "DFT") {
84          chosen_method = TransformType::DFT;
85          return;
86      }
87      if (read_method == "IDFT") {
88          chosen_method = TransformType::IDFT;
89          return;
90      }
91      cerr << "Cannot read method."
92           << endl;
93      exit(1);
94  }
95
96  static void
97  opt_help(string const &arg)
98  {
99      cout << program_name << " [-m DFT | IDFT] [-i file] [-o file]"
100          << endl;
101     exit(0);
102 }
103
104 static void
105 print_msg_and_exit(string const & msg)
106 {
107     cerr << msg
108          << endl;
```

```
109     exit(1);
110 }
111 
112 int
113 main(int argc, char * const argv[])
114 {
115     program_name = argv[0];
116     cmdline cmdl(options);
117     cmdl.parse(argc, argv);
118 
119     // Cuestiones de formato para la impresión:
120     oss->setf(ios::fixed, ios::floatfield);
121     oss->precision(6);
122 
123     bool status;
124     ComplexVector inSignal;
125     ComplexVector outSignal;
126     istringstream line;
127     string s;
128 
129     for (int lineNo = 1; getline(*iss, s); ++lineNo)
130     {
131         if (iss->bad())
132             print_msg_and_exit("An error occured while processing line " +
                    to_string(lineNo) + ".");
133 
134         line.str(s); //
135         line.clear();
136 
137         status = load_signal(line, inSignal);
138         if (!status)
139             print_msg_and_exit("Error processing \"" + line.str() + "\" (
                    line " + to_string(lineNo) + ").");
140 
141         status = transform(inSignal, outSignal, chosen_method);
142         if (!status)
143             print_msg_and_exit("An error occured while performing the
                    requested operation.");
144 
145         status = print_signal(*oss, outSignal);
146         if (!status)
147             print_msg_and_exit("Cannot write to output stream.");
148 
149         inSignal.clear();
150         outSignal.clear();
151     }
152 }
```

cmdline.h

```
1 #ifndef _CMDLINE_H_INCLUDED_
2 #define _CMDLINE_H_INCLUDED_
3 
```

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

#define OPT_DEFAULT   0
#define OPT_SEEN      1
#define OPT_MANDATORY 2

struct option_t {
   int has_arg;
   const char *short_name;
   const char *long_name;
   const char *def_value;
   void (*parse)(std::string const &);
   int flags;
};

class cmdline {
   option_t *option_table;
   cmdline();
   int do_long_opt(const char *, const char *);
   int do_short_opt(const char *, const char *);
public:
   cmdline(option_t *);
   void parse(int, char * const []);
};

#endif
```

cmdline.cpp

```
// cmdline - procesamiento de opciones en la [U+FFFD]lnea de comando.
//
// $Date: 2012/09/14 13:08:33 $
//
#include <string>
#include <cstdlib>
#include <iostream>

#include "cmdline.h"

using namespace std;

cmdline::cmdline()
{
}

cmdline::cmdline(option_t *table) : option_table(table)
{
   /*
   - Lo mismo que hacer:

   option_table = table;

```

```
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            }
```

```
            op->parse(string(arg));
            return 1;
        } else {
            op->parse(string(""));
            return 0;
        }
    }
  }
  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"
#include "DFT.h"

bool load_signal(std::istream &, DFT::ComplexVector &);
bool print_signal(std::ostream &, DFT::ComplexVector const &);

#endif   //_IO_H_INCLUDED_
```

io.cpp

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

#include "io.h"

using namespace std;
using namespace DFT;

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

```
18    return true;
19 }
20
21 bool
22 print_signal(ostream & os, ComplexVector const & output)
23 {
24    for (size_t i = 0; i < output.size(); ++i)
25       os << output[i];
26    os << endl;
27    if (os.bad())
28       return false;
29    return true;
30 }
```

DFT.h

```
1  #ifndef _DFT_H_INCLUDED_
2  #define  _DFT_H_INCLUDED_
3
4  #include "Complex.h"
5  #include "Vector.h"
6
7  namespace DFT {
8     typedef Vector <Complex <> > ComplexVector;
9     enum class TransformType { DFT, IDFT };
10    bool transform(ComplexVector const &, ComplexVector &, TransformType
         const & = TransformType::DFT);
11    const Complex <> coefficient(int const i, int const j, int const n,
         TransformType const & method = TransformType::DFT);
12 }
13
14 #endif   // _DFT_H_INCLUDED_
```

DFT.cpp

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

```
            sum = 0;
        }
        return true;
    }

    inline const Complex <>
    coefficient(int const i, int const j, int const n, TransformType const
         & method)
    {
        switch (method) {
            case TransformType::DFT:
                return exp(I * -2.0 * M_PI * i * j / n);
            case TransformType::IDFT:
                return exp(I * 2.0 * M_PI * i * j / n) / n;
            default:
                return 0;
        }
    }

}
```

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)
```

```
29          data[i] = (v.data)[i];
30      }
31      ~Vector() {
32          delete[] data;
33      }
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;
```

```
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; }
    T im() const { return y; }

    const Complex conj() const {
        return Complex(x, -y);
    }
    const T norm() const {
        return sqrt(x*x + y*y);
    }
    const T arg() const {
        return std::atan2(y,x);
    }
    const Complex operator+() const {
        return Complex(+x, +y);
    }
    const Complex operator-() const {
        return Complex(-x, -y);
    }
    const Complex operator+(const Complex& c) const {
        return Complex(x + c.x, y + c.y);
    }
    const Complex operator-(const Complex& c) const {
        return Complex(x - c.x, y - c.y);
    }
    const Complex operator*(const Complex& c) const {
        return Complex(x*c.x - y*c.y, y*c.x + x*c.y);
    }
    const Complex operator/(const Complex& c) const {
        return Complex((x*c.x + y*c.y) / (c.x*c.x + c.y*c.y),
                       (y*c.x - x*c.y) / (c.x*c.x + c.y*c.y));
    }
    Complex& operator=(const Complex& c) {
        x = c.x;
        y = c.y;
        return *this;
    }
    Complex& operator+=(const Complex& c) {
        x += c.x;
        y += c.y;
        return *this;
    }
    Complex& operator-=(const Complex& c) {
        x -= c.x;
        y -= c.y;
        return *this;
    }
    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_
```

DFT_test.h

```
1 #ifndef _DFT_TEST_H_INCLUDED_
2 #define _DFT_TEST_H_INCLUDED_
3 
4 #include "cmdline.h"
5 #include "io.h"
6 #include "DFT.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
15 #endif // _DFT_TEST_H_INCLUDED_
```

DFT_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 "DFT_test.h"
10
11 typedef Vector <Complex <long double> > ComplexVector;
12
13 using namespace std;
14 using namespace DFT;
15
16 #define PRINT(X) \
17    std::cerr << #X << ": " << X << std::endl
18
19 static char *program_name;
20 static size_t vectorSize;
21 static const size_t file_amount = 28;
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      {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
56      {0, },
57  };
58
59  static void
60  opt_number(std::string const &arg)
61  {
62      std::istringstream iss(arg);
63
64      if (!(iss >> vectorSize) || !iss.eof()) {
65          std::cerr << "Not a possible amount: "
66                    << arg
67                    << "."
68                    << std::endl;
69          exit(1);
70      }
71      if (iss.bad()) {
72          std::cerr << "Cannot read amount."
73                    << std::endl;
74          exit(1);
75      }
76  }
77
78  static void
79  opt_help(std::string const &arg)
80  {
81      std::cerr << program_name << " [-n amount]"
82                << std::endl;
83      exit(0);
84  }
85  // Google Test exige que las pruebas estén en un namespace sin nombre
86  //
87  namespace {
88      class RandomVectors : public ::testing::Test {
89      protected:
90          RandomVectors() : OrigVector(vectorSize),
91                            DFTVector(vectorSize),
92                            FinalVector(vectorSize)
93          {
94              cerr << "Esta prueba crea un vector de "
95                   << vectorSize
96                   << " números complejos pseudo-aleatorios "
```

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

```
145         }
146         ~VectorsFromFiles() {
147             cerr << endl;  // for formatting reasons
148         }
149         size_t i;
150         ifstream ifs;
151         ComplexVector *originalVector;
152         ComplexVector *transformedVector;
153         ComplexVector *DFTOutput;
154         ComplexVector *IDFTOutput;
155     };
156 
157     // Calculates the DFT of a random vector and then compares the IDFT of
        the result with the same vector
158     TEST_F(RandomVectors, DFTandIDFT) {
159         transform(OrigVector, DFTVector, TransformType::DFT);
160         transform(DFTVector, FinalVector, TransformType::IDFT);
161         for (size_t i = 0; i < vectorSize; ++i)
162             EXPECT_EQ(OrigVector[i], FinalVector[i]);
163     }
164 
165     TEST_F(VectorsFromFiles, DFTandIDFT) {
166         while (i < file_amount) {
167             originalVector = new ComplexVector;
168             transformedVector = new ComplexVector;
169             DFTOutput = new  ComplexVector;
170             IDFTOutput = new ComplexVector;
171             read_vectors_from_files();
172             if (!transform(*originalVector, *DFTOutput, TransformType::DFT))
173                 exit(1);
174             for (size_t j = 0 ; j < originalVector->size(); ++j)
175                 EXPECT_EQ((*transformedVector)[j], (*DFTOutput)[j]);
176             if (!transform(*transformedVector, *IDFTOutput, TransformType::
                IDFT))
177                 exit(1);
178             for (size_t j = 0 ; j < originalVector->size(); ++j)
179                 EXPECT_EQ((*originalVector)[j], (*IDFTOutput)[j]);
180             delete transformedVector;
181             delete DFTOutput;
182             delete originalVector;
183             delete IDFTOutput;
184             cerr << test_files[i]
185                  << " fue procesado."
186                  << endl;
187         }
188     }
189 
190 }  // namespace
191 
192 int main(int argc, char **argv) {
193     program_name = argv[0];
194     cmdline cmdl(options);
```

```
195     cmdl.parse(argc, argv);
196
197     ::testing::InitGoogleTest(&argc, argv);
198     return RUN_ALL_TESTS();
199 }
```

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 }
```