

## Código fuente del programa:

### Makefile

```
objects_dft = complejo.o vector_t.o dft.o test_dft.o
objects_prog = complejo.o vector_t.o dft.o fft.o cmdline.o main.o
objects_diff = complejo.o vector_t.o test_diff.o
```

```
programa: $(objects_prog)
    g++ -Wall -g -o programa $(objects_prog)
```

```
cmdline.o: cmdline.cpp cmdline.h
    g++ -Wall -g -c cmdline.cpp
```

```
complejo.o : complejo.cpp complejo.h
    g++ -Wall -g -c complejo.cpp
```

```
vector_t.o : vector_t.cpp vector_t.h complejo.h
    g++ -Wall -g -c vector_t.cpp
```

```
dft.o : dft.cpp dft.h vector_t.h complejo.h
    g++ -Wall -g -c dft.cpp
```

```
fft.o : fft.cpp fft.h vector_t.h complejo.h
    g++ -Wall -g -c fft.cpp
```

```
main.o : main.cpp cmdline.h complejo.h vector_t.h dft.h
    g++ -Wall -g -c main.cpp
```

```
test_dft.o: test_dft.cpp
    g++ -Wall -g -c test_dft.cpp
```

```
test_vector.o : test_vector.cpp
    g++ -Wall -g -c test_vector.cpp
```

```
test_diff.o: test_diff.cpp complejo.h vector_t.h
    g++ -Wall -g -c test_diff.cpp
```

#los archivos de prueba se deben llamar test y un numero

```
test_programa_dft: programa test_diff.o
    @g++ -Wall -g -o test_diff $(objects_diff)

    @echo "\n-----INICIA PRUEBA DE PROGRAMA-----\n"

    @echo "Probando fft sin argumento '-m:'\n"
    @set -e; for t in test_ft?; do
        echo Aplicando FFT a $$t;
        ./programa -i $$t -o $$t.out;
    done

    @set -e; for t in test_fft?; do
        echo Aplicando FFT a $$t;
        ./programa -i $$t -o $$t.out;
    done

    @echo "\n"
```

```

@set -e; for t in test_ft?; do
    echo Testing: $$t;
    ./test_diff $$t;
    echo Test ok;
done

@set -e; for t in test_fft?; do
    echo Testing: $$t;
    ./test_diff $$t;
    echo Test ok;
done

@rm *.out

@echo "\nProbando fft con argumento -m:\n"
@set -e; for t in test_ft?; do
    echo Aplicando FFT a $$t;
    ./programa -m "fft" -i $$t -o $$t.out;
done

@set -e; for t in test_fft?; do
    echo Aplicando FFT a $$t;
    ./programa -m "fft" -i $$t -o $$t.out;
done

@echo "\n"

@set -e; for t in test_ft?; do
    echo Testing: $$t;
    ./test_diff $$t;
    echo Test ok;
done

@set -e; for t in test_fft?; do
    echo Testing: $$t;
    ./test_diff $$t;
    echo Test ok;
done

@echo "\nTEST_FFT OK.\n"

@rm *.out

@echo "Probando ifft:\n"
@set -e; for t in test_ifft?; do
    echo Aplicando IFFT a $$t;
    ./programa -m "ifft" -i $$t -o $$t.out;
done

@set -e; for t in test_ifft?; do
    echo Aplicando IFFT a $$t;
    ./programa -m "ifft" -i $$t -o $$t.out;
done

@echo "\n"

@set -e; for t in test_ifft?; do
    echo Testing: $$t;
    ./test_diff $$t;

```

```

        echo Test ok; \
done

@set -e; for t in test_ifft?; do \
    echo Testing: $$t; \
    ./test_diff $$t; \
    echo Test ok; \
done

@echo "\nTEST_IFFT OK.\n"

@rm *.out

@echo "Probando dft:\n"
@set -e; for t in test_ft?; do \
    echo Aplicando DFT a $$t; \
    ./programa -m "dft" -i $$t -o $$t.out; \
done

@echo "\n"

@set -e; for t in test_ft?; do \
    echo Testing: $$t; \
    ./test_diff $$t; \
    echo Test ok; \
done

@echo "\nTEST_DFT OK.\n"

@echo "Probando idft:\n"
@set -e; for t in test_ifft?; do \
    echo Aplicando IDFT a $$t; \
    ./programa -m "idft" -i $$t -o $$t.out; \
done

@echo "\n"

@set -e; for t in test_ifft?; do \
    echo Testing: $$t; \
    ./test_diff $$t; \
    echo Test ok; \
done

@echo "\nTEST_IDFT OK.\n"

@echo "\n-----LA PRUEBA SE HA EJECUTADO SIN DETECTAR ERRORES-----\n"

@rm *.out test_diff

#comprueba que no haya fugas de memoria en el programa
test_programa_valgrind: programa
@echo "\n-----INICIA PRUEBA DE MEMORIA-----\n"

@echo "PROBANDO FFT:"

@set -e; for t in test_ft?; do \
    echo "\n" testing: $$t "\n"; \
    valgrind --leak-check=full ./programa -m "fft" -i $$t -o $$t.out; \
done

@set -e; for t in test_fft?; do \

```

```

        echo "\n" testing: $$t "\n";
        valgrind --leak-check=full ./programa -m "fft" -i $$t -o $$t.out;
done
@echo "\nPROBANDO IFFT:"

@set -e; for t in test_ifft?; do
    echo "\n" testing: $$t "\n";
    valgrind --leak-check=full ./programa -m "ifft" -i $$t -o $$t.out;
done

@set -e; for t in test_ifft?; do
    echo "\n" testing: $$t "\n";
    valgrind --leak-check=full ./programa -m "ifft" -i $$t -o $$t.out;
done

@echo "\nPROBANDO DFT:"

@set -e; for t in test_ft?; do
    echo "\n" testing: $$t "\n";
    valgrind --leak-check=full ./programa -m "dft" -i $$t -o $$t.out;
done

@echo "PROBANDO IDFT:"

@set -e; for t in test_ifft?; do
    echo "\n" testing: $$t "\n";
    valgrind --leak-check=full ./programa -m "idft" -i $$t -o $$t.out;
done

@echo "\n PROBANDO ARGUMENTO: '--help'\n"
@set -e; valgrind --leak-check=full ./programa -h;

@rm *.out

@echo "\n-----PRUEBA DE MEMORIA FINALIZADA-----\n"

#Prueba de la clase vector_t
test-vector_t: complejo.o vector_t.o test_vector.o
    g++ -Wall -o test_vector_t complejo.o vector_t.o test_vector.o

#Empieza la ejecución de la prueba y compara los archivos
    ./test_vector_t test_vector_t.txt
    @printf "\n-----Prueba de Vector_t-----\n\n"
    -@diff -T -s -b -w test_vector_t.txt out_test_vector_t.txt
    @rm complejo.o vector_t.o test_vector.o test_vector_t

test-vector_t-memory: complejo.o vector_t.o test_vector.o
    g++ -Wall -o test-vector_t-memory complejo.o vector_t.o test_vector.o
    @printf "\n-----Prueba de memoria del Vector_t-----\n\n"
    @valgrind --leak-check=yes ./test-vector_t-memory test_vector_t.txt
    @rm complejo.o vector_t.o test_vector.o test-vector_t-memory

clean:
    @rm -f *o *.out programa test_diff

```

## cmdline.h

```
#ifndef _CMDLINE_H_INCLUDED_
#define _CMDLINE_H_INCLUDED_
```

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

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

```
#define DFT "dft"
#define IDFT "idft"
#define FFT "fft"
#define IFFT "ifft"
```

```
using namespace std;
```

```
typedef enum type_method {
    METHOD_DFT,
    METHOD_IDFT,
    METHOD_FFT,
    METHOD_IFFT
} type_method;
```

```
struct option_t {
    int has_arg;
    const char *short_name;
    const char *long_name;
    const char *def_value;
    void (*parse)(std::string const &; // Puntero a función de opciones
    int flags;
};
```

```
class cmdline {
    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 []);
};
```

```
void opt_input(string const &);
void opt_output(string const &);
void opt_method(string const &);
void opt_help(string const &);
/////////////////////////
extern option_t options[];
```

```
extern type_method method;
```

```
extern std::istream *iss;    // Input Stream (clase para manejo de los flujos de
entrada)
extern std::ostream *oss;    // Output Stream (clase para manejo de los flujos de
salida)
extern std::fstream ifs;      // Input File Stream (derivada de la clase ifstream
que deriva de istream para el manejo de archivos)
extern std::fstream ofs;      // Output File Stream (derivada de la clase
ofstream que deriva de ostream para el manejo de archivos)
////////////////////////////////////
#endif
```

### cmdline.cpp

```
#include "cmdline.h"
#include "commands_MSG.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 << ARGUMENTO_INVALIDO
                 << 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 << "Opcion obligatoria "
                 << "-"
                 << OPTION_NAME(op)
```

```

        << " no ingresada."
        << "\n";
        exit(1);
    }
    if (op->def_value == 0)
        continue;
    op->parse(string(op->def_value));
}

int cmdline::do_long_opt(const char *opt, const char *arg){
    for (option_t *op = option_table; op->long_name != 0; ++op) {
        if (string(opt) == string(op->long_name)) {

            op->flags |= OPT_SEEN;

            if (op->has_arg) {
                if (arg == 0) {
                    cerr << FALTA_ARGUMENTO
                        << "--"
                        << opt
                        << "\n";
                    exit(1);
                }
                op->parse(string(arg));
                return 1;
            } else {
                op->parse(string(""));
                return 0;
            }
        }
    }

    cerr << OPCION_DESCONOCIDA << "--" << opt << "." << endl;
    exit(1);

    return -1;
}

int cmdline::do_short_opt(const char *opt, const char *arg){
    option_t *op;

    for (op = option_table; op->short_name != 0; ++op) {
        if (string(opt) == string(op->short_name)) {

            op->flags |= OPT_SEEN;

            if (op->has_arg) {
                if (arg == 0) {
                    cerr << FALTA_ARGUMENTO
                        << "-"
                        << opt
                        << "\n";
                    exit(1);
                }
            }
        }
    }
}

```

```

        }
        op->parse(string(arg));
        return 1;
    } else {

        op->parse(string(""));
        return 0;
    }
}

cerr << OPCION_DESCONOCIDA
    << "-"
    << opt
    << "."
    << endl;
exit(1);

return -1;
}

void opt_input(string const &arg){

    // Si el nombre del archivos es "-", usaremos la entrada
    // estándar. De lo contrario, abrimos un archivo en modo
    // de lectura.
    //
    if (arg == "-") {
        iss = &cin;        // Establezco la entrada estandar cin como flujo de entrada
    }
    else {
        ifs.open(arg.c_str(), ios::in); // c_str(): Returns a pointer to an array
that contains a null-terminated          // sequence of characters (i.e., a C-
string) representing                    // the current value of the string object.

        iss = &ifs;
    }

    // Verificamos que el stream este OK.
    //
    if (!iss->good()) {
        cerr << FALLO_ABRIR
            << arg
            << "."
            << endl;
        exit(1);
    }
}

void opt_output(string const &arg){

    // Si el nombre del archivos es "-", usaremos la salida
    // estándar. De lo contrario, abrimos un archivo en modo
    // de escritura.
    //
    if (arg == "-") {
        oss = &cout;        // Establezco la salida estandar cout como flujo de salida
    } else {

```



```

        ofs.open(arg.c_str(), ios::out);
        //printf("Prueba\n");

        oss = &ofs;
    }

    // Verificamos que el stream este OK.
    //
    if (!oss->good()) {
        cerr << "cannot open "
            << arg
            << "."
            << endl;
        exit(1);        // EXIT: Terminación del programa en su totalidad
    }
}

void opt_method(string const &arg){

    if(arg.compare(IDFT)==0)
        method=METHOD_IDFT;
    else if (arg.compare(FFT)==0)
        method=METHOD_FFT;
    else if (arg.compare(IFFT)==0)
        method=METHOD_IFFT;
    else if (arg.compare(DFT)!=0 && arg.compare("-")!=0) {
        cout<<METODO_INVALIDO<<endl;
        exit(1);
    }
}

void opt_help(string const &arg){

    cout << MSJ_AYUDA << endl;
    exit(0);
}

```

#### commands MSG.h

```

#ifndef COMMANDS_MSG_H
#define COMMANDS_MSG_H

#define ARGUMENTO_INVALIDO "Argumento no válido: "
#define FALTA_ARGUMENTO "La opción requiere argumento: "
#define OPCION_DESCONOCIDA "Opción desconocida: "
#define FALLO_ABRIR "No se pudo abrir: "
#define METODO_INVALIDO "Argumento no válido para -m/--method"
#define MSJ_AYUDA "Los argumentos del programa son:\n\t-i/--infile [ARCHIVO] En caso de no especificar uno, CIN está por defecto.\n\t-o/--output [ARCHIVO] En caso de no especificar uno, COUT está por defecto.\n\t-m/--method Los posibles son \"dft\" y \"idft\". En caso de no especificar uno o de ingresar \"-\" se tomará por defecto a dft.\n\t-h/--help Muestra la ayuda por pantalla."

#endif

```

#### complejo.h

```

#ifndef _COMPLEJO_H_INCLUDED_
#define _COMPLEJO_H_INCLUDED_

```

```

#include <iostream>

class complejo {
    double re_, im_;
public:
    complejo();
    complejo(double);
    complejo(double, double);
    complejo(const complejo &);
    complejo const &operator=(complejo const &);
    complejo const &operator*=(complejo const &);
    complejo const &operator+=(complejo const &);
    complejo const &operator-=(complejo const &);
    ~complejo();

    void set_re(double x);
    void set_im(double x);

    double re() const;
    double im() const;
    double abs() const;
    double abs2() const;
    complejo const &conjugar();
    complejo const conjugado() const;
    bool zero() const;

    friend complejo const operator+(complejo const &, complejo const &);
    friend complejo const operator-(complejo const &, complejo const &);
    friend complejo const operator*(complejo const &, complejo const &);
    friend complejo const operator*(complejo const &, double);
    friend complejo const operator/(complejo const &, complejo const &);
    friend complejo const operator/(complejo const &, double);

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

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

#endif

```

#### complejo.cpp

```

#include "complejo.h"
#include <iostream>
#include <cmath>

using namespace std;

complejo::complejo() : re_(0), im_(0)
{
}

complejo::complejo(double r) : re_(r), im_(0)
{
}

complejo::complejo(double r, double i) : re_(r), im_(i)

```

```

{
}

complejo::complejo(complejo const &c) : re_(c.re_), im_(c.im_)
{
}

complejo const &
complejo::operator=(complejo const &c)
{
    re_ = c.re_;
    im_ = c.im_;
    return *this;
}

complejo const &
complejo::operator*=(complejo const &c)
{
    double re = re_ * c.re_
               - im_ * c.im_;
    double im = re_ * c.im_
               + im_ * c.re_;
    re_ = re, im_ = im;
    return *this;
}

complejo const &
complejo::operator+=(complejo const &c)
{
    double re = re_ + c.re_;
    double im = im_ + c.im_;
    re_ = re, im_ = im;
    return *this;
}

complejo const &
complejo::operator-=(complejo const &c)
{
    double re = re_ - c.re_;
    double im = im_ - c.im_;
    re_ = re, im_ = im;
    return *this;
}

complejo::~~complejo()
{
}

void complejo::set_re(double x)
{
    re_=x;
}

void complejo::set_im(double x)
{
    im_=x;
}

double

```

```

complejo::re() const
{
    return re_;
}

double complejo::im() const
{
    return im_;
}

double
complejo::abs() const
{
    return std::sqrt(re_ * re_ + im_ * im_);
}

double
complejo::abs2() const
{
    return re_ * re_ + im_ * im_;
}

complejo const &
complejo::conjugar()
{
    im_*= -1;
    return *this;
}

complejo const
complejo::conjugado() const
{
    return complejo(re_, -im_);
}

bool
complejo::zero() const
{
#define ZERO(x) ((x) == +0.0 && (x) == -0.0)
    return ZERO(re_) && ZERO(im_) ? true : false;
}

complejo const
operator+(complejo const &x, complejo const &y)
{
    complejo z(x.re_ + y.re_, x.im_ + y.im_);
    return z;
}

complejo const
operator-(complejo const &x, complejo const &y)
{
    complejo r(x.re_ - y.re_, x.im_ - y.im_);
    return r;
}

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

```

```

    complejo r(x.re_ * y.re_ - x.im_ * y.im_,
               x.re_ * y.im_ + x.im_ * y.re_);
    return r;
}

complejo const
operator*(complejo const &x, double k)
{
    complejo r(x.re() * k, x.im() * k);
    return r;
}

complejo const
operator/(complejo const &x, complejo const &y)
{
    return x * y.conjugado() / y.abs2();
}

complejo const
operator/(complejo const &c, double f)
{
    return complejo(c.re_ / f, c.im_ / f);
}

bool
operator==(complejo const &c, double f)
{
    bool b = (c.im_ != 0 || c.re_ != f) ? false : true;
    return b;
}

bool
operator==(complejo const &x, complejo const &y)
{
    bool b = (x.re_ != y.re_ || x.im_ != y.im_) ? false : true;
    return b;
}

ostream &
operator<<(ostream &os, const complejo &c)
{
    return os << "("
               << c.re_
               << ", "
               << c.im_
               << ")";
}

istream &
operator>>(istream &is, complejo &c)
{
    int good = false;
    int bad  = false;
    double re = 0;
    double im = 0;
    char ch = 0;

    if (is >> ch
        && ch == '(') {

```

```

        if (is >> re
            && is >> ch
            && ch == ', '
            && is >> im
            && is >> ch
            && ch == ')'){
            good = true;
        }
        else
            bad = true;
    }
    else if (is.good()) {
        is.putback(ch);
        if (is >> re)
            good = true;
        else
            bad = true;
    }

    if (good)
        c.re_ = re, c.im_ = im;
    if (bad){
        is.clear(ios::badbit);
    }

    return is;
}

```

#### dft.h

```

#ifndef DFT_H
#define DFT_H

#include <iostream>
#include <cmath>
#include "complejo.h"
#include "vector_t.h"

vector_t dft(vector_t & vector_in);
vector_t idft(vector_t & vector_in);

#endif

```

#### dft.cpp

```

#include "dft.h"

vector_t dft(vector_t & vector_in){

    int k, n, largo_in = vector_in.leng();
    vector_t vector_out(largo_in);
    complejo aux{};

    for (k = 0; k < largo_in; k++)
    {
        for (n = 0, aux.set_re(0), aux.set_im(0); n < largo_in; n++)
        {
            aux.set_re(aux.re() + (vector_in[n].re())*cos(2*M_PI*n*k/largo_in) +
            (vector_in[n].im())*sin(2*M_PI*n*k/largo_in));
        }
    }
}

```

```

        aux.set_im(aux.im() - (vector_in[n].re())*sin(2*M_PI*n*k/largo_in) +
(vector_in[n].im())*cos(2*M_PI*n*k/largo_in));
    }
    vector_out.append(aux);
}

return vector_out;
}

vector_t idft(vector_t & vector_in){

    int n, k, largo_in = vector_in.leng();
    vector_t vector_out(largo_in);
    complejo aux{};

    for (n = 0; n < largo_in; n++)
    {
        for (k=0, aux.set_re(0), aux.set_im(0); k < largo_in; k++)
        {
            aux.set_re(aux.re() + (vector_in[k].re())*cos(2*M_PI*k*n/largo_in) -
(vector_in[k].im())*sin(2*M_PI*k*n/largo_in));
            aux.set_im(aux.im() + (vector_in[k].re())*sin(2*M_PI*k*n/largo_in) +
(vector_in[k].im())*cos(2*M_PI*k*n/largo_in));
        }
        aux.set_re(aux.re() / largo_in);
        aux.set_im(aux.im() / largo_in);
        vector_out.append(aux);
    }

    return vector_out;
}

```

#### fft.h

```

#ifndef FFT_H
#define FFT_H

```

```

#include <iostream>
#include <cmath>
#include "complejo.h"
#include "vector_t.h"
#include "dft.h"

vector_t fft (vector_t & vector_in);
vector_t _fft (vector_t & vector_in);
vector_t ifft (vector_t & vector_in);
vector_t _ifft (vector_t & vector_in);
void fill0till_exp2 (vector_t & vector_in);

#endif

```

#### fft.cpp

```

#include "fft.h"

```

```

vector_t fft (vector_t & vector_in) {

    vector_t vector_out = vector_in;

    fill0till_exp2 (vector_out); //Redimensiona el vector hasta un longitud 2^n
}

```

```

    return _fft (vector_out);
}

vector_t _fft (vector_t & vector_in) {

    int i, largo = vector_in.leng();
    complejo aux, wn;

    vector_t vector_out(largo);
    vector_t vector_in_par(largo / 2), vector_in_impar(largo / 2),
    vector_out_par(largo / 2), vector_out_impar(largo / 2); //crear con tamaño
    determinado

    largo = vector_in.leng();

    if (vector_in.leng() >= 2) {

        for (i = 0; i < largo; i += 2) { //Crea el vector con los indices par

            aux = vector_in[i];
            vector_in_par.append(aux);
        }

        for (i = 1; i < largo; i += 2) { //Crea el vector con los indices impar

            aux = vector_in[i];
            vector_in_impar.append(aux);
        }

        vector_out_par = _fft (vector_in_par);
        vector_out_impar = _fft (vector_in_impar);

        for (i = 0; i < largo/2; i++) {

            wn.set_re (cos (2 * M_PI * i / largo));
            wn.set_im (-sin (2 * M_PI * i / largo));

            aux = vector_out_par[i] + vector_out_impar[i] * wn;

            vector_out.append(aux);
        }

        for(; i<largo; i++) {

            wn.set_re (cos (2 * M_PI * i / largo));
            wn.set_im (-sin (2 * M_PI * i / largo));

            aux = vector_out_par[i - largo/2] + vector_out_impar[i - largo/2] * wn;

            vector_out.append(aux);
        }
    } else {

        vector_out = vector_in;
    }
}

```



```

    return vector_out;
}

vector_t ifft (vector_t & vector_in) {

    vector_t vector_out = vector_in, vector_aux;
    int i, largo;
    complejo aux;

    fill0till_exp2 (vector_out);
    vector_out = _ifft (vector_out);

    largo = vector_out.leng();

    for(i=0; i < largo; i++) {
        aux = vector_out[i] / largo;
        vector_aux.append(aux);
    }

    return vector_aux;
}

vector_t _ifft (vector_t & vector_in) {

    vector_t vector_out, vector_in_par, vector_in_impar, vector_out_par,
    vector_out_impar;
    int i, largo;
    complejo aux, wn;

    largo = vector_in.leng();

    if (vector_in.leng() >= 2) {

        for (i = 0; i < largo; i += 2) {

            aux = vector_in[i];
            vector_in_par.append(aux);
        }

        for (i = 1; i < largo; i += 2) {

            aux = vector_in[i];
            vector_in_impar.append(aux);
        }

        vector_out_par = _ifft (vector_in_par);
        vector_out_impar = _ifft (vector_in_impar);

        for (i = 0; i < largo/2; i++) {

            wn.set_re (cos (2 * M_PI * i / largo));
            wn.set_im (sin (2 * M_PI * i / largo));

            aux = (vector_out_par[i] + vector_out_impar[i] * wn);

            vector_out.append(aux);
        }

        for(; i<largo; i++) {

```

```

        wn.set_re (cos (2 * M_PI * i / largo));
        wn.set_im (sin (2 * M_PI * i / largo));

        aux = (vector_out_par[i - largo/2] + vector_out_impar[i - largo/2] *
wn);

        vector_out.append(aux);
    }

} else {

    vector_out = vector_in;
}

return vector_out;
}

void fill0till_exp2 (vector_t & vector_in) {

    int largo = vector_in.leng();
    double exp2 = log2(largo);
    int nappends, int_exp2 = exp2;
    complejo cero;

    if(exp2 - int_exp2){

        nappends = pow(2, int_exp2 + 1) - largo;

        while (nappends > 0){
            vector_in.append(cero);
            nappends--;
        }
    }
}

```

main.cpp

```

#include <string>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <cstdlib>
#include <cmath>
#include <iomanip>

#include "cmdline.h"
#include "complejo.h"
#include "vector_t.h"
#include "dft.h"
#include "fft.h"

#define MSJ_ERR_VEC_CORRPUTO_1 "El vector de la línea "
#define MSJ_ERR_VEC_CORRPUTO_2 " se encuentra corrupto, es imposible transformar"

using namespace std;

option_t options[] = {
    {1, "i", "input", "-", opt_input, OPT_DEFAULT},

```

```

    {1, "o", "output", "-", opt_output, OPT_DEFAULT},
    {1, "m", "method", "-", opt_method, OPT_DEFAULT},
    {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
    {0, },
};

std::istream *iss=0;    // Input Stream (clase para manejo de los flujos de
entrada)
std::ostream *oss=0;    // Output Stream (clase para manejo de los flujos de
salida)
std::fstream ifs;       // Input File Stream (derivada de la clase ifstream que
deriva de istream para el manejo de archivos)
std::fstream ofs;       // Output File Stream (derivada de la clase ofstream que
deriva de ostream para el manejo de archivos)
type_method method = METHOD_FFT;

int main (int argc, char * const argv[])
{
    cmdline cmdl(options);
    cmdl.parse(argc, argv); // Metodo de parseo de la clase cmdline

    vector_t vec_in;
    vector_t vec_out;
    vector_t (*funcionFourier)(vector_t &);
    size_t num_linea = 1;

    //Selecciona cual es el metodo a usar

    if (method == METHOD_DFT)
        funcionFourier = dft;
    else if (method == METHOD_IFFT)
        funcionFourier = ifft;
    else if (method == METHOD_IDFT)
        funcionFourier = idft;
    else
        funcionFourier = fft;

    while(!iss->eof()){
        if(!(*iss >> vec_in)){
            if(iss -> eof()){
                break;
            }

            if(oss != &cout){
                *oss << endl;
            }

            cout << MSJ_ERR_VEC_CORRPUTO_1 << num_linea << MSJ_ERR_VEC_CORRPUTO_2
<< endl;

            iss->clear(ios::goodbit);
            vec_in.clean();
            vec_out.clean();
            num_linea++;

            continue;
        }
    }
}

```

```

        vec_out = funcionFourier(vec_in);

        *oss << vec_out << endl;

        vec_in.clean();
        vec_out.clean();
        num_linea++;
    }

    if(ifs.is_open())
        ifs.close();

    if(ofs.is_open())
        ofs.close();

    return 0;
}

```

#### vector t.h

```

#ifndef VECTOR_T_H
#define VECTOR_T_H

#define VECTOR_DEFAULT_CAPACIDAD_INICIAL 10;
#define VECTOR_DEFAULT_STEP 5

#include <iostream>
#include "complejo.h"

using namespace std;

class vector_t
{
private:
    complejo* p;
    size_t tam = 0;
    size_t capacidad = 0;

    void aumentar_cap(size_t cant);

public:
    vector_t();
    vector_t(size_t largo);
    vector_t(const vector_t &copia);
    ~vector_t();

    size_t leng() const; //Devuelve el tamaño del vector
    void append(complejo &valor); //Agrega el complejo al final del vector
    bool swap(complejo &val, int pos); //Cambia el valor del complejo en la
posición seleccionada
    void clean(); //Limpia el vector y lo devuelve a la capacidad default;

    vector_t operator+(const vector_t &a);
    complejo operator[](int pos);
    vector_t& operator = (const vector_t &vec);
    bool operator==(const vector_t &vec);

```

```

        friend istream & operator>>(istream &is, vector_t &v);
        friend ostream & operator<<(ostream &os, vector_t &v);

};

#endif

vector_t.cpp
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

#include "vector_t.h"

vector_t::vector_t(){
    capacidad = VECTOR_DEFAULT_CAPACIDAD_INICIAL;

    p = new complejo[capacidad];
}

vector_t::vector_t(size_t largo){
    capacidad = largo;
    p = new complejo[capacidad];
}

vector_t::vector_t(const vector_t &copia){
    tam = copia.tam;
    capacidad = copia.capacidad;

    p = new complejo[capacidad];

    for(size_t i = 0; i < capacidad; i++){
        p[i] = copia.p[i];
    }
}

vector_t::~~vector_t(){
    delete[] p;
}

size_t vector_t::leng() const{
    return tam;
}

void vector_t::aumentar_cap(size_t cant){
    complejo* aux = new complejo[capacidad + cant];

    for (size_t i = 0; i < capacidad; i++){
        aux[i] = p[i];
    }

    capacidad = capacidad + cant;
}

```

```

        delete[] p;
        p = aux;
    }

void vector_t::append(complejo &valor)
{
    if(tam == capacidad){
        aumentar_cap(VECTOR_DEFAULT_STEP);
    }

    p[tam] = valor;
    tam++;
}

bool vector_t::swap(complejo &val, int pos){
    if(pos >= 0 && (size_t)pos < tam){
        p[pos - 1] = val;
        return true;
    }
    else
        return false;
}

void vector_t::clean()
{
    tam = 0;
    capacidad = VECTOR_DEFAULT_CAPACIDAD_INICIAL;
    delete [] p;
    p = new complejo[capacidad];
}

vector_t vector_t::operator +(const vector_t &a){
    size_t longitud = tam + a.leng();

    if(capacidad < longitud){
        this->aumentar_cap(longitud);
    }

    for (size_t i = 0; i < (size_t)a.leng(); ++i)
    {
        p[this->leng() + i] = a.p[i];
    }

    return *this;
}

vector_t& vector_t::operator = (const vector_t &vec){
    tam = vec.tam;
    capacidad = vec.capacidad;

    delete[] p;
    p = new complejo[capacidad];
}

```

```

        for(size_t i = 0; i < capacidad; i++){
            p[i] = vec.p[i];
        }

        return *this;
    }

    complejo vector_t::operator [](int pos){
        return p[pos];
    }

    bool vector_t::operator ==(const vector_t &vec){

        if(tam != vec.tam)
        {
            return false;
        }

        for(size_t i = 0; i < tam; i++)
        {
            if(!(p[i] == vec.p[i]))
            {
                return false;
            }
        }

        return true;
    }

    istream & operator >>(istream &is, vector_t &v){ //lee un vector_t de complejos
    separados por espacios del archivo is

        complejo c;
        string line;

        if(!getline(is, line)){
            return is;
        }

        istringstream stream_line(line);

        while(stream_line >> c){
            v.append(c);
        }

        //En caso de que haya error, cambia estado de is
        if(stream_line.bad()){
            is.clear(ios::badbit);
        }

        return is;
    }

    ostream & operator <<(ostream &os, vector_t &v){

        for (size_t i = 0; i < v.tam; i++)
        {
            os << v[i] << " ";
        }
    }

```

```
    }  
    return os;  
}
```