Casos de Prueba y Pruebas
Casos de Prueba:

test_fft1
```
1 1 1
1 1 1 1 1
1 2 3
```

test_fft1.ref
```
(3, 0) (0, -1) (1, 0) (0, 1)
(5, 0) (0, -2.414213562) (1, 0) (0, -0.4142135624) (1, 0) (0, 0.4142135624) (1, 0)
(0, 2.414213562)
(6, 0) (-2, -2) (2, 0) (-2, 2)
```

test_ft1
```
(5,0)
5,0)
(0,8)
0,8)
(1,2)
0,8
(3,4)
(1)
(1,0) (1,0)
(1,)
(0,1) (0,1)
(,2)
(1,2) (3,4)
(,)
(-5,1) (1,1) (4,-6) (-2, -7)
```

test_ft1.ref
```
(5, 0)

(0, 8)

(1, 2)

(3 ,4)

(2, 0) (0, 0)

(0, 2) (0, 0)

(4, 6) (-2, -2)

(-2, -11) (-1, 4) (0, 1) (-17, 10)
```

test_ft2
```
1
1 1
1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

test_ft2.ref

```
(1, 0)
(2, 0) (0, 0)
(4, 0) (0, 0) (0, 0) (0, 0)
(8, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0)
(16, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0)
(32, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0)
```

test_ft3

```
(1, 0) (0, 0) 0 (0, 0) 0 0 (0, 0) (0, 0)
```

test_ft3.ref

```
(1, 0) (1, 0) (1, 0) (1, 0) (1, 0) (1, 0) (1, 0) (1, 0)
```

test_ft4

```
1 1 1 1
(1, 0) (-1, 0) (1, 0) (-1, 0)
```

test_ft4.ref

```
(4, 0) (0, 0) (0, 0) (0, 0)
(0, 0) (0, 0) (4, 0) (0, 0)
```

test_ifft1

```
1 1 1
1 1 1 1 1
```

test_ifft1.ref

```
(0.75, 0) (0, 0.25) (0.25, 0) (0, -0.25)
(0.625, 0) (0, 0.301777) (0.125, 0) (0, 0.0517767) (0.125, 0) (-9.71445e-17, -0.0517767) (0.125, 0) (0, -0.301777)
```

test_ift1

```
(5,0)
5,0)
(0,8)
0,8)
(1,2)
0,8
(3,4)
(1)
(2,0) (0,0)
(1,)
(0,2) (0,0)
(,2)
(4,6) (-2,-2)
(,)
(-2,-11) (-1,4) (0,1) (-17,10)
```

test_ift1.ref

```
(5, 0)

(0, 8)
```

```
(1, 2)

(3, 4)

(1, 0) (1, 0)

(0, 1) (0, 1)

(1, 2) (3, 4)

(-5, 1) (1, 1) (4, -6) (-2, -7)
```

test_ift2
```
(1,0)
(2,0) (0,0)
(4,0) (0,0) (0,0) (0,0)
```

test_ift2.ref
```
(1, 0)
(1, 0) (1, 0)
(1, 0) (1, 0) (1, 0) (1, 0)
```

test_ift3
```
(1,0) (1,0) (1,0) (1,0) (1,0) (1,0) (1,0) (1,0)
```

test_ift3.ref
```
(1, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0)
```

test_ift4
```
(4, 0) (0, 0) (0, 0) (0, 0)
(0, 0) (0, 0) (4, 0) (0, 0)
```

test_ift4.ref
```
1 1 1 1
(1, 0) (-1, 0) (1, 0) (-1, 0)
```

test_vector
```
#STREAM
1 = (1,0)
1 1 = (1,0) (1,0)
1 0 0  = (1,0) (0,0) (0,0)
(1,1,1) = ERROR
(1,0) (2,1) (3,4) = (1,0) (2,1) (3,4)
(1,) = ERROR

#LARGO
1 = 1
1 1 1 = 3
(1,0) (1,0) = 2
0 0 = 2
= 0
(1,0) 0 0 0 0 4 5 = 7
(111,11,1) = ERROR

#==
(1,0) == 1 = TRUE
```

```
0 == 2 = FALSE
1 1 == (1,0) (1,0) = TRUE
(1,2) == (1,2) = TRUE
```

Código de Prueba:

test_vector.cpp

```
#include <iostream>
#include <sstream>
#include <stdlib.h>
#include <fstream>
#include <string.h>

#include "complejo.h"
#include "vector_t.h"

using namespace std;

//------------------------------------------------------------
void OperacionSuma(ofstream &, string );
void OperacionIO(ofstream &, string);
void OperacionLargo(ofstream &, string);
void OperacionIgualdad(ofstream &, string);

#define TEST_LARGO_OPCIONES_METODOS 4
char * const Opciones_metodos[] =
{
    "#+", //0-Prueba el operador suma
    "#STREAM", //1-Prueba el input y el output
    "#LARGO", //2-Prueba el largo del vector
    "#=="  //3-Prueba la igualdad
};

void (*Operaciones[])(ofstream &, string) =
{
    OperacionSuma,
    OperacionIO,
    OperacionLargo,
    OperacionIgualdad
};

//-----------------------------------------------------

int main(int argc, char ** argv)
{
    ifstream file_in;
    ofstream file_out;
    string nombre_archivo(argv[1]);
    string line_readed;
    bool operacion_elegida = false;

    void (*Operacion_a_probar)(ofstream &, string);

    file_in.open(nombre_archivo);
    file_out.open("out_" + nombre_archivo, ofstream::trunc);

    if(!file_in || !file_out)
    {
        cout << "No se pudo abrir el archivo de prueba\n";
        return 1;
```

```
    }

    while(!file_in.eof())
    {
        getline(file_in, line_readed);

        //En caso de que la linea este vacia, saltear
        if(line_readed.empty())
        {
            file_out << endl;
            continue;
        }

        else  //En caso de que sea una de las opciones
        {
            for(int i = 0; i < TEST_LARGO_OPCIONES_METODOS; i++)
            {
                if(strcmp(line_readed.c_str(),Opciones_metodos[i]) == 0)
                {
                    file_out << Opciones_metodos[i] << endl;
                    Operacion_a_probar = Operaciones[i];
                    operacion_elegida = true;
                }
            }
        }

        if(operacion_elegida) //Salteo esta iteracion si hubo un metodo elegido
        {
            operacion_elegida = false;
            continue;
        }

        Operacion_a_probar(file_out, line_readed); //Pruebo la operacion
    }

    file_in.close();
    file_out.close();
}

void OperacionSuma(ofstream & out, string line){return;}

void OperacionIO(ofstream & out, string line){

    vector_t vector_prueba;
    string print_line = line.substr(0, line.find_last_of("="));
    stringstream string_buffer(print_line);

    out << print_line << " = ";
    string_buffer >>  vector_prueba;

    if(string_buffer.bad())
    {
        out << "ERROR" << endl;
        return;
    }

    out << vector_prueba << endl;
    return;
}
```

```
void OperacionLargo(ofstream & out, string line){

    vector_t vector_prueba;
    string print_line = line.substr(0, line.find_last_of("="));
    stringstream string_buffer(print_line);

    out << print_line << " = ";
    string_buffer >>  vector_prueba;

    if(string_buffer.bad())
    {
        out << "ERROR" << endl;
        return;
    }

    out << vector_prueba.leng() << endl;
    return;
}

void OperacionIgualdad(ofstream & out, string line){

    vector_t vector_izquierdo, vector_derecho;
    string print_line = line.substr(0, line.find_last_of("="));
    string line_after_operator = print_line.substr(print_line.find_first_of("=") +
1);

    out << print_line << " = ";

    stringstream
buffer_izquierdo(print_line.substr(0,print_line.find_first_of("=")));
    stringstream
buffer_derecho(line_after_operator.substr(1,line_after_operator.size()-1)); //Borro
basura

    buffer_izquierdo >> vector_izquierdo;
    buffer_derecho >> vector_derecho;

    if(vector_izquierdo == vector_derecho)
    {
        out << "TRUE" << endl;
        return;
    }

    out << "FALSE" << endl;
    return;
}
```

test_diff.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
    using namespace std;
#include <string>

#include "vector_t.h"
#include "complejo.h"

#define MAX_DIF 1e-5
```

```
//compara dos archivos de igual nombre pero diferente extención
//se debe ingresar solo el nombre del archivo como argumento
//Este programa da por sentado que los archvios de referencia están bien escritos.
si esto no es así podria fallar

int main(int argc, char const *argv[])
{
    string name_file = argv[1];

    ifstream file_out(name_file + ".out");
    ifstream file_ref(name_file + ".ref");

    string line_out;
    string line_ref;

    complejo c_out, c_ref;
    int i = 1;
    bool good = true;
    bool eof = false;
    bool falla_lectura = false;
    bool linea_corrupta;

    if(!file_out.is_open() || !file_ref.is_open()){
        cout << "error al abrir los archivos " << endl;
        return 1;
    }

    while(!eof && !falla_lectura){

        linea_corrupta = false;

        getline(file_out, line_out);
        getline(file_ref, line_ref);

        if(file_out.eof() || file_ref.eof()){

            eof = true;
            continue;
        }

        if(file_out.bad() || file_ref.bad()){

            cout << "Error al leer el los archivos" << endl;
            falla_lectura = true;
            good = false;
            continue;
        }

        if(line_out.compare(line_ref)){

            if(line_out.empty() || line_ref.empty()){
                cout << "ERROR en la linea nº " << i << "\n" << line_out << "\nEs
diferente a:\n" << line_ref << "\n" << endl;
                    good = false;
                    i++;
                    continue;
            }
```

```
            istringstream str_line_out(line_out);
            istringstream str_line_ref(line_ref);

            while(str_line_out >> c_out && str_line_ref >> c_ref && !
linea_corrupta){

                if(abs(c_out.re() - c_ref.re()) > MAX_DIF || abs(c_out.im() -
c_ref.im()) > MAX_DIF){
                //compara los complejos con cierta tolerancia

                    cout << "ERROR en la linea nº " << i << "\n" << line_out << "\
nEs diferente a:\n" << line_ref << "\n" << endl;
                    linea_corrupta = true;
                    good = false;
                }
            }
        }
        i++;
    }

    if(!falla_lectura && file_out.eof() != file_ref.eof()){
        cout << "ERROR: faltan lineas en el archivo" << endl;
        good = false;
    }

    file_out.close();
    file_ref.close();

    if(good)
        return 0;

    else
        return 1;
}
```