ANÁLISIS Y DISEÑO DE MÉTODOS NUMÉRICOS EN VARIABLE COMPLEJA CON MODELOS PARA INGENIERÍA

ALFONSO PIO AGUDELO S.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES
2001

INTRODUCCION

La mayoría de los textos sobre métodos numéricos, trabajan básicamente sobre variable real, pero muchos de los problemas en ingeniería deben resolverse en variable compleja, este trabajo aborda dicho tema. Para cada problema se hace análisis, diseño de los algoritmos y su implementación en lenguaje C.

El primer capítulo hace una introducción al lenguaje C, como elemento fundamental para el desarrollo de los programas.

El segundo capítulo versa sobre los sistemas de numeración de complejos, haciendo incapié en las operaciones básicas tanto de números como de matrices complejas.

El tercer capítulo trata sobre raíces de funciones complejas, en particular polinomios con raíces complejas y en segundo lugar polinomios de números complejos.

El último capítulo aborda los sistemas de ecuaciones simultáneas lineales complejas.

TABLA DE CONTENIDO

1. LENGUAJE DE PROGRAMACION C

1.1. Los tipos de datos básicos

Un dato es todo aquella forma de valor con la que puede trabajar el computador. Un tipo de dato define el conjunto de valores que puede tomar una variable. Existen cinco tipos de datos básicos en C: char, int, float, double y void. El número de bytes que ocupa y el rango de datos en que varía cada tipo depende del computador y de la versión del compilador de C.

1.1.1 Tipo char

El valor de un dato de tipo char, corresponde al valor numérico en binario de ese carácter dentro del conjunto determinado por el estándar (ASCII), dicho valor es almacenado en la memoria de un byte, esto es ocho bits (dígito binario). Ese valor numérico representa a uno y sólo uno de las 256 posibles representaciones en dicho estándar de codificación.

A los caracteres se les expresa o representa formalmente en C utilizando las comillas simples.

Ejemplo:

'a', '1', ',', 'A', 'I', '&'.

Donde el valor numérico en binario 01000001, que es equivalente en valor numérico decimal al 65, le corresponde el carácter 'A'.

Existe un grupo de caracteres especiales que cumple una función específica cuando se utiliza en un programa C, estos son:

'\'' Comilla simple.
'\"' Comilla doble.
'\\' Backslash.
'\0' Final de un arreglo o de un string, es el carácter nulo.
'\a' Pitido.
'\b' Backspace (retroceso).
'\f' Salta una nueva hoja.
'\n' Salta a la próxima línea en el dispositivo de salida.
'\r' Return (retorno de carro).
'\t' Tabula la salida de datos.

1.1.2. Tipo int

Un dato de tipo int corresponde a un valor numérico de dominio entero. Puede contener números enteros con signo, esto es, negativos y positivos.

Ejemplos:

4555, 5, -286.

Los datos que se encuentran definidos sólo como int, para 16 bits se determinan en el rango de valores: -32768 a 32767, que corresponde a un dominio de 65536 números enteros, es por esto que un valor numérico que se salga de este rango será representado en la memoria del computador con un valor diferente.

Ejemplo: el número 32769, almacenado como valor de una variable tipo int, en una máquina que maneje tipo int de 2 bytes (16 bits), estará representado por el valor –32767.

Los datos de tipo int en C pueden tener tres modificaciones:

1.1.2.1 short int (precisión corta). Este tipo de dato ocupa 1, 2, 4 ó 8 bytes, este lo determina la arquitectura de la máquina. En 8 bits (1 byte) se define el rango de valores: -128 a 127.

1.1.2.2 long int (precisión larga). Ocupa 32 ó 64 bits de acuerdo a la arquitectura de la máquina.
En 32 bits (4 bytes), se define el rango de valores: -4294967296 a 4294967295.

1.1.2.3 unsigned int (entero sin signo). Ocupa 16, 32 ó 64 bits. En 16 bits (2 bytes) se define el rango de valores: 0 a 65535.

1.1.3. Tipo float

Corresponde al tipo que puede almacenar valores reales o también conocidos como valores en punto flotante, son números que pueden poseer fracciones decimales. Un tipo float utiliza 4 bytes de memoria y su dominio se encuentra en el rango de valores:

$-3.4x10^{38}$ hasta $-3.4x10^{-38}$, 0, $3.4x10^{-38}$ hasta $3.4x10^{38}$.

Un tipo float domina al tipo int, quiere decir que cualquier valor almacenado en un tipo int podrá almacenarse en un tipo float, la diferencia está en que en éste tipo el valor almacenado gastará más memoria y su formato de almacenamiento será diferente.

Ejemplos:

623571.3254, -565.00, 32223.50, 6754.6, -767.69.

1.1.4. Tipo double

Es un número real con mayor precisión que un dato de tipo float, ocupa 8 bytes de memoria, que permite almacenar valores en el rango:

$-1.7x10^{308}$ hasta $-1.7x10^{-308}$, 0, $1.7x10^{-308}$ hasta $1.7x10^{308}$.

1.1.4.1 Tipo long double

Consiste en una modificación del tipo double, el cual ocupa 10 bytes de memoria, permitiendo manejar valores en memoria del rango:
$-3.4x10^{4932}$ hasta $-3.4x10^{-4932}$, 0, $3.4x10^{-4932}$ a $3.4x10^{4932}$.

1.1.5 Tipo void

Este tipo se usa principalmente para declarar funciones en forma explícita que no devuelven valor alguno o que no sea significativo. También se utiliza para crear punteros void (punteros a void) los cuales son punteros genéricos que poseen la capacidad de apuntar a cualquier tipo de objeto o dato.

Ejemplo:

void main()

En este caso, se indica que la función principal main no entrega valor alguno.

1.2. Estructuras de datos

1.2.1 Variables

Son posiciones de memoria identificadas con un nombre único. Su función es la de almacenar datos en forma temporal ya que puede cambiar su valor en la ejecución del programa.

Para dar el nombre a una variable se deben tener en cuenta las siguientes reglas:

1-Debe comenzar por una letra

2-Puede contener solamente letras y números.
3-Puede contener más de ocho caracteres pero solamente para su identificación se tienen en cuenta los primeros ocho.
4-El carácter '_' puede tratarse como una letra al definir el nombre de una variable, excepto en el primer carácter del nombre de la variable.

Ejemplos:

p25, Cas_ser

Los nombres de las variables no pueden ser palabras reservadas. Siendo una palabra reservada una palabra clave dentro del lenguaje C, por lo tanto no se pueden utilizar como nombre de variable ni de función, las 32 palabras reservadas definidas en el estándar ANSI son:

auto	**continue**	**void**	**if**	**int**
extern	**else**	**const**	**register**	**for**
signed	**float**	**typedef**	**do**	**volatile**
double	**static**	**while**	**struct**	**goto**
switch	**union**	**return**	**case**	**long**
sizeof	**void**	**short**	**break**	**enum**
unsigned	**char**			

Declarar una variable es asociarle el respectivo tipo y darle un nombre, lo cual hace que el compilador le reserve una localización apropiada en memoria.

Primero se declara el tipo de dato y posteriormente el nombre de la variable correspondiente. Todas las variables deben haber sido declaradas antes de usarlas.

Ejemplos:

int y; short int x; float w; char a;

El compilador C, asignará 2 bytes de memoria para la variable y, un byte para la variable x, 4 bytes para la variable w y un byte para la variable a.

La sentencia de asignación se utiliza para almacenar un valor determinado en una variable. La forma general de la sentencia de asignación es: nombre_variable = valor;

Ejemplo:

a=20;

En la posición de memoria de la variable a, almacena el valor 20.

1.2.2. Constantes

Una constante es una localización de memoria cuya función es la de almacenar datos bajo un nombre único con la característica que su valor no puede variar durante la ejecución del programa.

Para definir una constante se hace de la siguiente manera:

 # define nombre-constante valor

Ejemplos:

Las constantes carácter se deben definir en su valor entre comillas simples.

 # define CARACTER 'A'

 # define ESCRIBA printf("%f",a)

La definición obliga al compilador a hacer en el programa la sustitución cada vez que encuentre la cadena ESCRIBA por la cadena printf("%f",a).

 # define KA 342

Se ha definido la constante KA como constante entera equivalente a 342.

 # define PI 3.141592

La constante PI se ha definido como un real 3.141592, se podrá utilizar esta constante real como un valor en el programa, para realizar las operaciones necesarias.

 # define NUMERO –5945.4E-10

Esta constante real corresponde al valor –0.00000059454 que se ha expresado en notación científica, nótese que la expresión E-10 significa $x10^{-10}$.

1.2.3. Arreglos

Un arreglo es un conjunto de datos que se encuentran organizados de manera secuencial en memoria, se debe declarar el arreglo asociándole: un tipo de datos, un nombre de arreglo y un tamaño.

La forma general de declarar un arreglo de una dimensión es:

Tipo nombredevariable [tamaño];

Ejemplo:

float vector[18];

El compilador reserva en memoria un espacio equivalente 18 float, esto es, si tenemos 4 bytes por float, equivale a reservar 72 bytes para la variable llamada vector.

Para hacer referencia a un dato del arreglo, se hace referencia al nombre de la variable y seguido entre corchetes rectangulares al subíndice que indica la posición del valor dentro del arreglo, empezando dicha posición en cero: vector[0] se refiere al primer valor del arreglo, vector[1] al segundo valor, vector[17] al último valor del arreglo declarado. El subíndice puede referirse a otra variable, por ejemplo, vector[k] en donde k debe ser una variable tipo entero con un valor; si k=5, se está haciendo referencia a vector[5].

La forma general de declarar un arreglo de dos dimensiones es:

Tipo nombredevariable [tamañof][tamañoc];

Ejemplo:

char cliente[10][30];

En este caso, la variable cliente, es un conjunto de valores tipo carácter que consiste de 10 cadenas de caracteres (string) cada una de ellas con hasta 29 caracteres.

1.2.4. Punteros

Un puntero es una variable que indica una posición de un dato, esto quiere decir que un puntero almacena una dirección de memoria en donde se encuentra una dato o un conjunto de datos. Los punteros son de gran utilidad y son usados con cierta frecuencia en la programación de computadores. Son usados, por ejemplo para pasar datos desde una función a otra, o desde una función a algún lugar de llamada. Los punteros se relacionan con los arreglos, en el sentido que aquellos se utilizan en formaciones logrando una forma práctica de acceso a las direcciones de memoria en donde se encuentran los datos.

En lenguaje C, los punteros se declaran al igual que otras variables en la cabecera de la función, el tipo puntero se refiere al tipo de dato donde direcciona el puntero.

Ejemplo:

```
int *p;
```

La anterior declaración le indica al compilador, que la variable p es un puntero, que almacenará la dirección de memoria en donde se almacena un dato de tipo entero.

1.2.5. Estructuras

Una estructura es una organización de datos cuyos elementos individuales pueden ser de distinto tipo, una estructura puede tener simultáneamente datos de tipo entero, flotante, arreglos o matrices, caracteres, punteros y otras estructuras. Al elemento de una estructura se le conoce con el nombre de miembro o campo.

Ejemplo:

```
struct cliente
      { int codigo;
        char nombre[30];
        float sueldo;
       }
```

La declaración de la estructura lleva la palabra reservada struct y después el nombre de la estructura (cliente). Esta estructura maneja tres campos o miembros: el código, el nombre y el sueldo del cliente

1.2.6. Archivos de datos

Muchos programas requieren que los datos permanezcan en dispositivos de memoria auxiliar, dado que cuando ocurre un fallo en el suministro de energía de una computadora, a esta se le borran los datos de la memoria RAM principal. Por este motivo, los archivos almacenan datos de forma permanente y permiten el acceso y modificación de los mismo cuando sea necesario.

En lenguaje C se distinguen dos tipos de archivos: los archivos secuenciales de datos o estándar y los archivos orientados al sistema o de bajo nivel. Usualmente es más fácil trabajar con archivos de datos secuenciales que con los otros, por esta razón son esos archivos los más utilizados.

Los archivos secuenciales de datos se pueden clasificar en dos categorías: los archivos de texto, que contienen caracteres en forma consecutiva y que se pueden referenciar carácter a carácter como componentes de una cadena; la otra categoría constituye los archivos de datos secuenciales sin formato, pues se organizan los datos en bloques de bytes contiguos.

La declaración de un archivo en un programa en C, debe contener la palabra reservada FILE, y la variable del archivo como un puntero.

Ejemplo:

```
FILE *archivo;
```

1.3. Entradas y salidas

1.3.1. Función printf()

La función printf() en C permite escribir y efectuar salidas por pantalla. La forma general de esta función es la siguiente:

printf (cadena de control, lista de argumentos);

La cadena de control consiste en dos tipos de elementos. El primer elemento está construido por caracteres que serán impresos en la pantalla. El segundo contiene órdenes de formato que definen la forma de visualizar los argumentos. Una orden de formato empieza con el signo de porcentaje "%". Debe haber el mismo número de argumentos que de ordenes de formato. Dichos argumentos deben corresponderse en el orden de los formatos definidos.

Ejemplo:

```
printf("el carácter %c el entero %d la cadena %s", 'A', 89, "Mani");
```

En la pantalla aparece:
el carácter A el entero 89 la cadena Mani

Los formatos pueden ser:

%c char (un carácter).
%hd short int (entero corto).
%u unsigned (entero sin signo).
%lu unsigned long int (entero largo sin signo).
%d int (entero).
%i int (entero).
%ld long int (entero largo).
%e float o double (real o real doble) en notación exponencial.
%f float o double (real o real doble).
%lf double (real doble).
%Lf long double (real doble largo).
%o octal.

%x hexadecimal.
%s string (cadena de caracteres).
%p puntero.

Las órdenes de formato pueden tener modificadores, para determinar la longitud del campo o número de decimales que se debe presentar al igual que se puede determinar la justificación de éste.

Para definir la longitud mínima de un campo, se coloca el valor entero entre el signo porcentaje y el formato. Si se desea que un campo mida cinco dígitos por lo menos, pero el valor del campo es menor de cinco dígitos, éste se rellenará con espacios en blanco, si se desea que se rellene de ceros entonces se debe anteponer el valor 0 al número de longitud mínima del campo.

Ejemplo:

```
#include "stdio.h"
int a,b; char c;
void main()
{
 a = 5;
 b = 4;
 printf("Los números son....: %3d %02d",a,b);
}
```

El resultado de este programa al correr es:

```
Los números son....:   5 04
```

Si se escribe el signo negativo como modificador; alineará los números a la izquierda.

Ejemplo:

```
#include "stdio.h"
int a,b; char c;
main()
{
  a = 5;
  b = 4;
  printf("Los números son....: %-3d %02d",a,b);
}
```

Los resultados son los mismos del ejemplo anterior, a diferencia que los números los deja alineados a la izquierda. El printf también se utiliza para sacar mensajes

por pantalla y pueden ir acompañado de los formatos mencionados anteriormente.

Ejemplo:

```
#include "stdio.h"
main()
{
  printf("%15s","UNIVERSIDAD NACIONAL");
}
```

El resultado será:

UNIVERSIDAD NACIONAL

1.3.2. Función scanf()

Esta función se utiliza para leer datos desde un medio de almacenamiento externo. La función lee los datos y los almacena en las variables indicadas de acuerdo a un formato.

Ejemplo:

```
#include "stdio.h"
int a;
main()
{
 printf("Entre el número..---> ");
 scanf("%d",&a);
 printf("El valor entrado es : %d",a);
}
```

El resultado del programa es:

```
Entre el número  ---> 7
El valor entrado es :  7
```

El valor del número queda almacenado en la variable a. Hay que tener en cuenta que antes de la variable hay que anteponerle el signo ampersand (&). Al anteponer el signo ampersand se hace referencia al contenido de la variable.
Se pueden determinar modificadores a los formatos de órdenes de tal manera que se puedan obviar ciertas entradas por teclado. Para realizar esta tarea se recurre al uso del símbolo * entre c y signo de tanto por ciento y el formato a omitir.

Ejemplo:

```
#include "stdio.h"
int d,m,a;
main()
{
        printf("Entre la fecha de liquidación (dd-mm-aaaa)..");
        scanf("%d%*c%d%*c%d",&d,&m,&a);
        printf("\n La fecha de liquidación es.....");
        printf("%2d-%2d-%4d",d,m,a);
}
```

El resultado del programa:

Entre la fecha de liquidación (dd-mm-aaaa).. 01-02-2001
La fecha de liquidación es.....1 -2 -2001

En el ejemplo anterior se dispone la entrada de múltiples variables, en este caso tres, la entrada por consola determina los valores para las variables tecleándolas por consola y separando cada uno de ellos por un espacio en blanco, pero en este caso la persona que se encuentra al mando del teclado puede entrar los valores de la siguiente manera: 01-02-2001 y el almacenamiento se distribuirá en las variables descritas de tal manera que los valores encontrados en las variables serán: d = 01, m = 02 y a = 2001. En este caso el carácter guión - no es tomado en la captura y almacenamiento de las variables, esto sucede debido a que el formato %c de la función scanf() se encontraba de la manera %*c permitiendo así la omisión de un carácter. El símbolo * después del carácter % ignora la entrada y avanza al siguiente formato.

Los formatos para la función scanf() son las siguientes:

%c un carácter.
%i Decimal.
%e Notación exponencial.
%f Real o de punto flotante.
%h Entero corto.
%o Octal.
%s Cadena de carácter.
%p Puntero.
%x Hexadecimal.

1.3.3. Función getchar()

La función getchar permite la captura de caracteres por teclado. El uso de esta función es de menor nivel que la de scanf(). Los valores que almacena son los correspondientes en la tabla del código ASCII

Ejemplo:

```
#include "stdio.h"
int a,b;
char c; main()
{
  a= getchar();
  b= getchar();
  printf("El primer carácter es %c el otro es %c",a,b);
}
```

El resultado del programa:

```
kf
El primer carácter es k el otro es f
```

El uso de la instrucción #include determina una llamada a alguna de las librerías que posee el compilador C o alguna creada por el usuario, la función getchar() necesita el uso de la librería stdio.h para que sea reconocida mediante la compilación del programa.

1.3.4. Función putchar( )

La función putchar() retorna el valor de almacenamiento en memoria.

Ejemplo:

```
#include "stdio.h"
int a,b;
main()
 {
    a = 'a';
    b = 'z';
    putchar(a);
    putchar(b);
}
```

El resultado del programa:

```
az
```

En este programa se asignan valores a las variables a y b, después, estos valores son enviados al dispositivo estándar a través de la función putchar(); obsérvese que entre paréntesis se debe colocar la localización de memoria que se desea visualizar.

1.3.5. Función getch( )

La función getch() permite la captura de un carácter. La diferencia con el getchar( ) es que el carácter capturado no es presentado por pantalla; para poderlo ver hay que imprimirlo con printf o con putchar( ).

1.3.6. gotoxy()

Esta función se utiliza para localizar los mensajes en la columna y fila deseada en la salida. Para poder hacer uso de ella, se recurre de la librería stdio.h.

Su sintaxis es:

gotoxy(columna,fila);

1.3.7. clrscr()

Esta opción permite limpiar la pantalla, para lo cual se debe incluir la librería stdio.h.

Sintaxis:

clrscr();

1.4. Operadores

1.4.1. Operadores aritméticos

Los operadores aritméticos en C son los mismos que en la mayoría de lenguajes de programación, a continuación se definen estos operadores:

+ suma.
- resta.
/ división.
* multiplicación.
% módulo de división (residuo de la división).

Si el símbolo / es aplicado a valores enteros, el resultado de la operación será entera.

Ejemplo:

10/3 dará como resultado 3.

El operador % permite obtener el residuo de una división, es válido únicamente para datos de tipo entero.

Ejemplo:

7%3 dará como resultado 4.

1.4.2. Operadores de incremento y decremento.

C ofrece dos operadores que otros lenguajes no poseen, son de gran utilidad para el manejo aritmético de algunas variables. Estos dos operadores son: ++ , -- (incremento y decremento respectivamente).

El operador ++ incrementa en uno (1) una variable, x = x + 1 es igual a decir x++ ; este operador también se puede escribir antes de la variable ++x.

El uso del operador -- decrementa en 1 la variable, x = x - 1 es igual a decir x-- ó --x.

Sin embargo, existe una una diferencia cuando se utiliza ++x o x+ +
Si se utiliza x + + primero escribe el contenido de la variable x y después incrementa su valor.

Ejemplo:

```
x = 10;
y = x++;
x = 10;
y = ++x;
```

En la variable y queda almacenado el valor de 11.

También es posible incrementar o decrementar el valor de una variable con valores diferentes a 1 mediante los operedores += y -=.

Ejemplos:

x+=5; equivale a x=x+5.
x- - =3; equivale a x=x-3;

1.4.3. Operadores de relación

Los operadores de relación son aquellos que permiten relacionar las variables con sus contenidos:

> Mayor que
< Menor que
== Igual que
<= Menor o igual que
>= Mayor o igual que
!= Diferente de

1.4.4. Operadores lógicos

Los operadores lógicos permiten determinar el enlace entre dos condiciones, dando como respuestas valores booleanos ya sea verdadero ó falso. La precedencia de los operadores lógicos es menor que la de los operadores aritméticos.

Los operadores lógicos son:

&& Y
|| O
! No

1.5. Sentencia if

El objetivo de la sentencia if en cualquier lenguaje de programación es la de poder realizar tomas de decisiones a través del programa. Al hablar de toma de decisiones se refiere a las evaluaciones que el programa hará a determinadas condiciones. Cuando se evalúa una condición sólo se espera uno de dos valores como respuesta, falso o verdadero. Si la condición se cumple, el valor será verdadero, de lo contrario el valor será falso.
Una condición es la relación lógica que se establece entre variables o variables y valores. En general existen tres tipos de tomas de decisiones.

1.5.1. if

La sentencia if determina una evaluación donde se ejecutará un determinado número de instrucciones si la condición se cumple. Si la condición no se cumple simplemente el programa no ejecutará ordenes. La forma general de esta forma es:

if (condición) instrucción;

Si el número de instrucciones que se debe ejecutar es mayor a 1 en caso de que la condición sea verdadera, dichas sentencias se deben asociar entre los símbolos {}, así:

```
if (condición)
{ instrucción 1;
  ...
  instrucción n;
}
```

1.5.2. if con else

En este caso la condición posee dos grupos de instrucciones a ejecutar, el primero de ellos se realizará cuando la condición que se evalúe sea verdadera, el segundo se realizará cuando la condición que se evalúe sea falso.

La estructura general de esta forma es:

```
if (condición)
{
  instrucción 1;
  ...
  instrucción n;
}
else
{
  instrucción 1;
  ...
  instrucción n;
}
```

1.5.3 if anidado.

La estructura if anidado consiste en poder evaluar condiciones dentro de una parte verdadera o una parte falsa ubicada en una evaluación superior, estas estructuras son muy comunes para los programadores de lenguajes estructurados. C proporciona una forma fácil de manejo para este tipo de estructura. Su forma general puede ser:

```
if(condición 1)
{
  if(condición 2)
```

```
  {  instrucción1;
     ...
     instrucciónN;
  }
  else
  {  instrucción1;
     ...
     instrucciónN;
  }

}
else
{  instrucción1;
   ...
   instrucciónN;
}
```

Ejemplo:

```
#include "stdio.h"
#include "conio.h"
int a,b,c,mayor;
main( )
{ clrscr();
  gotoxy(10,5);
  printf("Entre el número.....:");
  gotoxy(10,6);
  printf("Entre el número.....:");
  gotoxy(10,7);
  printf("Entre el número.....:");
  gotoxy(50,5);
  scanf("%d",&a);
  gotoxy(50,6);
  scanf("%d",&b);
  gotoxy(50,7);
  scanf("%d",&c);
  if(a<b) mayor = b;
  else mayor = a;
  if(c>mayor) mayor = c;
  gotoxy(10,10);
  printf("De %d , %d y %d el número mayor es %d",a,b,c,mayor);
}
```

El resultado del programa es:

```
Entre el número.....:    8
Entre el número.....:    5
Entre el número.....:    10

De 8 , 5 y 10 el número mayor es 10
```

El operador ?

El C permite un operador muy conveniente a la vez que potente, y puede usarse para sustituir sentencias de la forma if else. El operador ternario (Expresión compuesta por 3 elementos), toma la siguiente forma general:

Expl? Exp2: Exp3;

Donde Expl, Exp2 y Exp3 son expresiones. El operador ? actúa así:
evalua Expl, si es cierto, evalúa Exp2 y ese es el valor de la expresión Expl, si Expl es falsa, evalúa Exp3 y ése es el valor de la expresión Expl.

1.6. Estructuras de control de Ciclos

1.6.1. Sentencia for

La sentencia for permite ejecutar un determinado número de veces una o más instrucciones. En esta estructura se determina un valor inicial, una condición que se debe cumplir para que se ejecute cl bucle de for, y por último un incremento con el cual se varía cl valor inicial de la variable.

for(valor inicial; condición; incremento)

Si se desea utilizar una sola sentencia o instrucción a ejecutar, el formato será:

for (valor inicial; condición; incremento) instrucción;

Si se desea utilizar más de una instrucción a ejecutar, el formato es:

```
for(valor inicial; condición; incremento)
{
  instrucción 1;
  ...
  instrucción n;
}
```

La sentencia for puede ser utilizada sin cuerpo de ejecución, es decir, sin ninguna instrucción en su estructura. Esto ayuda a los programadores a hacer mucho más eficientes los programas.

Un ejemplo de la instrucción for sin cuerpo es:

```
for(x= 1; x< 100; x++ );
```

El for infinito es aquel que no posee parámetros de inicio, condición y de incremento, de esta manera se torna en un ciclo sin fin:

```
for(;;)
{
  instrucción 1;
  ...
  instrucción n;
}
```

Ejemplo:

```
#include "stdio.h"
#include "conio.h"
int tabla, resp, mult;
char enter;
main()
{
        clrscr();
        gotoxy(10,5);
        printf("ESCRIBA EL NUMERO DE LA TABLA.....");
        gotoxy(60,5);
        scanf("%d",&tabla);
        for(mult = 1; mult <= 10; mult++)
        {
                resp = mult*tabla;
                gotoxy(30,mult+8);
                printf("%d X %d = %d", tabla, mult,resp);
                gotoxy(30,22);
        }
}
```

El resultado del programa es:

ESCRIBA EL NUMERO DE LA TABLA 7

7 X 1 = 7
7 X 2 = 14
7 X 3 = 21
7 X 4 = 28
7 X 5 = 35
7 X 6 = 42
7 X 7 = 49
7 X 8 = 56
7 X 9 = 63
7 X 10 = 70

1.6.2. Sentencia while

La sentencia while permite ejecutar una o más instrucciones mientras una condición se cumpla. El compilador verificará la condición, si esta se llega a cumplir (verdadero), se ejecutarán las instrucciones que se encuentran dentro del bucle, de lo contrario se omite su ejecución y el control de programa pasará a la instrucción ubicada inmediatamente después del bucle. La condición es verdadera cuando posee un valor distinto de cero. La forma general del bucle while es la siguiente.

```
while(condición)
   instrucción;
```

Si se desea agrupar más de una instrucción, se hace uso de los símbolos {}. Así:

```
while(opc>1 && opc <3)
{  clrscr( );
   gotoxy(20,81);
   printf("  M E N U    P R I N C I P A L");
   gotoxy(18,10);
   printf("1. Entrada de datos ");
   gotoxy(18,ll);
   printf("2. Consulta ");
   gotoxy(18,12);
   printf("3. Finalizar ");
   gotoxy(18,14);
   printf("DIGITE LA OPCION DESEADA");
   gotoxy(45,14);
```

```
  scanf("%d",&opc);
  if (opc == 1)
  { printf("Primera opción del menú ");
    scanf("%s",&enter);
  }
  if (opc == 2)
  { printf("Segunda opción del menú");
    scanf("%s",&enter);
  }
  if (opc == 3)
    printf("Tercera opción del menú");
  if (opc < 1 ¡ opc>3)
  { printf("ESTA OPCION NO EXISTE EN EL MENU");
    printf("DIGITE OTRA VEZ LA OPCION");
    scanf("%s",&enter);
 }
 else  if (opc = = 1 ii ~ = = 2)        , 4; )
```

1.6.3. Sentencia do-while

La sentencia do-while permite realizar una o más instrucciones en forma repetitiva mientras que una condición determinada se cumple. La diferencia de esta sentencia con la sentencia while es que el programa ejecuta la condición después de haber ejecutado el grupo de órdenes de la condición definida. Lo anterior significa que el programa entrará por lo menos una vez a este bucle. La forma general de esta estructura es:

```
do
{
  instrucción(es);
}
while(condición)
```

El uso de los corchetes en esta estructura se hace obligatorio.

Ejemplo:

```
#include "stdio.h"
#include "conio.h"
int sal , suma, y;
char enter;
main()
{ clrscr();
 suma=0;
 y=1;
```

```
 do
 {
  printf("DIGITE UN NUMERO ---> ");
  gotoxy(50,y);
  scanf("%d",&sal);
  suma = suma + sal;
  y++;
 }
 while(y <= 6);
 printf("LA SUMA DE LOS NUMEROS ES --->");
 gotoxy(50,y);
 printf("%d",suma);
}
```

El resultado del programa es:

```
DIGITE UN NUMERO --->                    2
DIGITE UN NUMERO --->                    4
DIGITE UN NUMERO --->                    8
DIGITE UN NUMERO --->                    3
DIGITE UN NUMERO --->                    1
DIGITE UN NUMERO --->                    11
LA SUMA DE LOS NUMEROS ES --->           29
```

1.7 Sentencia exit()

La sentencia exit() sirve para dar fin a un programa y obliga a retornar al sistema operativo. Esta función generalmente va acompañada normalmente con el argumento (), indicando que la terminación es normal.

2. SISTEMA DE NUMEROS COMPLEJOS

Un número complejo tiene la forma a + b *i*, donde a y b son números reales que se denominan parte real e imaginaria respectivamente, *i* es denominada unidad imaginaria y equivale a la raíz cuadrada de –1. Se considera el conjunto de los números reales como un subconjunto de los números complejos. Al realizar operaciones con complejos, podemos hacerlo como se hace con el álgebra de los números reales, teniendo en cuenta que donde se encuentre i^2 se reemplaza por –1.

2.1. Operaciones con números complejos

2.1.1. La suma

Análisis

Dados dos complejos

C1= a1 + b1 *i*, y
C2= a2 + b2 *i*;

La suma esta definida por

C1 + C2 = C3
a1 + b1 *i* + a2 + b2 *i* = (a1 + a2) + (b1 + b2) *i*

Ejemplo:

C1= 5+4 *i*
C2= 8+3 *i*
C1+C2=5+4 *i* + 8+3 *i*
C3= 13+7 *i*

Diseño del algoritmo para sumar dos números complejos

LEER a1, b1, a2, b2
a3 = a1 + a2
b3 = b1 + b2
ESCRIBIR a3 + b3 i

Programas en lenguaje C para sumar dos números complejos

Versión 1

```
#include "stdio.h"
void main()
{ float a1,b1,a2,b2,a3,b3;
  printf(" Entre la parte real del complejo C1      --> ");
  scanf("%f",&a1);
  printf(" Entre la parte imaginaria del complejo C1 --> ");
  scanf("%f",&b1);
  printf(" Entre la parte real del complejo C2      --> ");
  scanf("%f",&a2);
  printf(" Entre la parte imaginaria del complejo C2 --> ");
  scanf("%f",&b2);
  a3=a1+a2;
  b3=b1+b2;
  printf(" La suma de %5.2f %+5.2f i %+5.2f %+5.2f i = %5.2f %+5.2f i"
        ,a1,b1,a2,b2,a3,b3);
}
```

El resultado del programa es:

```
Entre la parte real del complejo C1          --> 5
Entre la parte imaginaria del complejo C1 --> 3
Entre la parte real del complejo C2          --> 7
Entre la parte imaginaria del complejo C2 --> 8
La suma de 5.00 + 3.00 i +7.00 +8.00 i = 12.00 + 11.00 i
```

Versión 2

```
#include "stdio.h"
#include "complex.h"
void main()
{ complex c1,c2,c3;
  float a,b;
  printf(" Entre la parte real del complejo C1      --> ");
  scanf("%f",&a);
  printf(" Entre la parte imaginaria del complejo C1 --> ");
  scanf("%f",&b);
  c1=complex(a,b);
  printf(" Entre la parte real del complejo C2      --> ");
  scanf("%f",&a);
  printf(" Entre la parte imaginaria del complejo C2 --> ");
  scanf("%f",&b);
  c2=complex(a,b);
```

```
  c3=c1+c2;
  printf(" La suma de %5.2f %+5.2f i ",c1,imag(c1));
  printf(" %+5.2f %+5.2f i ",c2,imag(c2));
  printf(" = %5.2f %+5.2f i ",c3,imag(c3));
}
```

El resultado del programa es:

Entre la parte real del complejo C1 --> 4
Entre la parte imaginaria del complejo C1 --> 3.2
Entre la parte real del complejo C2 --> 2.1
Entre la parte imaginaria del complejo C2 --> 9
La suma de 4.00 + 3.20 i +2.10 +9.00 i = 6.10 + 12.20 i

2.1.2. La resta

Análisis

Dados dos complejos

C1= a1 + b1 *i*, y
C2= a2 + b2 *i*;

La diferencia esta definida por

C1 - C2 = C3
a1 + b1 *i* – (a2 + b2 *i*) = (a1 - a2) + (b1 - b2) *i*

Ejemplo:

C1= 8-4 *i*; C2= -5+13 *i*
C1-C2=8-4 *i* –(-5+13 *i*)
C3= 13-17 *i*

Diseño del algoritmo

LEER a1, b1, a2, b2
a3 = a1 - a2
b3 = b1 - b2
ESCRIBIR a3 + b3 i

Programas en lenguaje C para restar complejos

Versión 1

```
#include "stdio.h"
void main()
{  float a1,b1,a2,b2,a3,b3;
   printf(" Entre la parte real del complejo C1      --> ");
   scanf("%f",&a1);
   printf(" Entre la parte imaginaria del complejo C1 --> ");
   scanf("%f",&b1);
   printf(" Entre la parte real del complejo C2      --> ");
   scanf("%f",&a2);
   printf(" Entre la parte imaginaria del complejo C2 --> ");
   scanf("%f",&b2);
   a3=a1-a2;
   b3=b1-b2;
   printf(" La diferencia de (%5.2f %+5.2f i)-(%5.2f %+5.2f i) = %5.2f %+5.2f i"
,a1,b1,a2,b2,a3,b3);
}
```

El resultado del programa es:

```
Entre la parte real del complejo C1          --> 8
Entre la parte imaginaria del complejo C1 --> 2
Entre la parte real del complejo C2          --> 6
Entre la parte imaginaria del complejo C2 --> 4
La diferencia de (5.00 + 2.00 i )-(6.00 +4.00 i )= -1.00 + -2.00 i
```

Versión 2

```
#include "stdio.h"
#include "complex.h"
void main()
{  complex c1,c2,c3;
   float a,b;
   printf(" Entre la parte real del complejo C1      --> ");
   scanf("%f",&a);
   printf(" Entre la parte imaginaria del complejo C1 --> ");
   scanf("%f",&b);
   c1=complex(a,b);
   printf(" Entre la parte real del complejo C2      --> ");
   scanf("%f",&a);
   printf(" Entre la parte imaginaria del complejo C2 --> ");
   scanf("%f",&b);
   c2=complex(a,b);
   c3=c1-c2;
   printf("La diferencia (%5.2f %+5.2f i)",c1,imag(c1));
   printf("-(%5.2f %+5.2f i )",c2,imag(c2));
   printf("=%5.2f %+5.2f i" ,c3,imag(c3));
}
```

El resultado del programa es:

Entre la parte real del complejo C1 --> 3
Entre la parte imaginaria del complejo C1 --> 1
Entre la parte real del complejo C2 --> 8
Entre la parte imaginaria del complejo C2 --> -4
La diferencia de (3.00 + 1.00 i)-(8.00 -4.00 i)= -5.00 + 5.00 i

2.1.3. El producto

Análisis

Dados dos complejos
C1= a1 + b1 *i*, y
C2= a2 + b2 *i*;

El producto esta definido

C1 * C2 = C3
(a1 + b1 *i*) * (a2 + b2 *i*) = a1 * a2 + a1 * b2 *i* + a2 * b1 *i* + (b1 * b2) i^2
= (a1 * a2 – b1 * b2) + (a1 * b2 + a2 * b1) *i*

Ejemplo:

C1= 2-2 *i*
C2= -3+1 *i*
C1*C2=2*-3+2 *i*+6 *i*+2=-4 +8 *i*
C3= -4+8 *i*

Diseño del algoritmo

LEER a1, b1, a2, b2
a3 = a1 * a2 – b1 * b2
b3 = a1 * b2 + a2 * b1
ESCRIBIR a3 + b3 i

Programas en C para multiplicar dos complejos

Versión 1

```
#include "stdio.h"
void main()
{ float a1,b1,a2,b2,a3,b3;
  printf(" Entre la parte real del complejo C1      --> ");
  scanf("%f",&a1);
```

```
  printf(" Entre la parte imaginaria del complejo C1 --> ");
  scanf("%f",&b1);
  printf(" Entre la parte real del complejo C2      --> ");
  scanf("%f",&a2);
  printf(" Entre la parte imaginaria del complejo C2 --> ");
  scanf("%f",&b2);
  a3=a1*a2-b1*b2;
  b3=a1*b2+a2*b1;
  printf(" El producto de (%5.2f %+5.2f i)*(%5.2f %+5.2f i)= %5.2f %+5.2f i"
         ,a1,b1,a2,b2,a3,b3);
}
```

El resultado del programa es:

```
Entre la parte real del complejo C1          --> 5
Entre la parte imaginaria del complejo C1 --> 3
Entre la parte real del complejo C2          --> 4
Entre la parte imaginaria del complejo C2 --> 2
El producto de (5.00 + 3.00 i )*(4.00 +2.00 i )= 14.00 + 22.00 i
```

Versión 2

```
#include "stdio.h"
#include "complex.h"
void main()
{ complex c1,c2,c3;  float a,b;
  printf(" Entre la parte real del complejo C1      --> ");
  scanf("%f",&a);
  printf(" Entre la parte imaginaria del complejo C1 --> ");
  scanf("%f",&b);
  c1=complex(a,b);
  printf(" Entre la parte real del complejo C2      --> ");
  scanf("%f",&a);
  printf(" Entre la parte imaginaria del complejo C2 --> ");
  scanf("%f",&b);
  c2=complex(a,b);
  c3=c1*c2;
  printf(" El producto de (%5.2f %+5.2f i)",c1,imag(c1));
  printf("*(%+5.2f %+5.2f i )",c2,imag(c2));
  printf("=%5.2f %+5.2f i"  ,c3,imag(c3));
}
```

El resultado del programa es:

Entre la parte real del complejo C1 --> 2
Entre la parte imaginaria del complejo C1 --> -2
Entre la parte real del complejo C2 --> -3
Entre la parte imaginaria del complejo C2 --> 1
El producto de (2.00 - 2.00 i)*(-3.00 +1.00 i)= -4.00 + 8.00 i

2.1.4. El cociente

Análisis

Dados dos complejos

C1= a1 + b1 *i*, y
C2= a2 + b2 *i*;

El cociente está definido:

C1 / C2 = C3

Se toma el conjugado del denominador en esta fracción y se multiplica a la vez por el numerador y por el denominador, logrando así obtener en el denominador una expresión real.

(a1+b1 *i*) / (a2+b2 *i*) = (a1 + b1 *i*) * (a2 – b2 *i*) / (a2 + b2 *i*) * (a2 – b2 *i*)
= (a1*a2-a1*b2 *i* +a2*b1 *i* -b1*b2 i^2) / (a2*a2 - a2* b2 *i*+ a2* b2 *i* -b2*b2 i^2)
= (a1*a2+b1*b2+(a2*b1-a1*b2) *i*) / (a2*a2+b2*b2)

Obsérvese que el denominador de la anterior expresión es un real.

Ejemplo:

C1= -4-8 *i*
C2= 2+1 *i*
(a1*a2+b1*b2) / (a2*a2+b2*b2)=-3.2
(a2*b1-a1*b2) / (a2*a2+b2*b2)=-2.4

C1/C2=-3.2 –2.4 *i*
C3=-3.2 –2.4 *i*

Diseño del algoritmo

LEER a1, b1, a2, b2
denom=a2*a2+b2*b2
a3 = (a1*a2+b1*b2) / denom

b3 = (a2*b1-a1*b2) / denom
ESCRIBIR a3 + b3 i

Programas en lenguaje C para dividir complejos

Versión 1

```
#include "stdio.h"
void main()
{ float a1,b1,a2,b2,a3,b3, denom;
  printf(" Entre la parte real del complejo C1      --> ");
  scanf("%f",&a1);
  printf(" Entre la parte imaginaria del complejo C1 --> ");
  scanf("%f",&b1);
  printf(" Entre la parte real del complejo C2      --> ");
  scanf("%f",&a2);
  printf(" Entre la parte imaginaria del complejo C2 --> ");
  scanf("%f",&b2);
  denom=a2*a2+b2*b2;
  a3 = (a1*a2+b1*b2) / denom;
  b3 = (a2*b1-a1*b2) / denom;
  printf(" El cociente de (%5.2f %+5.2f i ) / (%5.2f %+5.2f i )= %5.2f %+5.2f i"
         ,a1,b1,a2,b2,a3,b3);
}
```

El resultado del programa es:

```
Entre la parte real del complejo C1          --> -4
Entre la parte imaginaria del complejo C1 --> -8
Entre la parte real del complejo C2          --> 2
Entre la parte imaginaria del complejo C2 --> 1
El cociente de (-4.00 - 8.00 i )/(2.00 +1.00 i )= -3.20 - 2.40 i
```

Versión 2

```
#include "stdio.h"
#include "complex.h"
void main()
{ complex c1,c2,c3;
  float a,b;
  printf(" Entre la parte real del complejo C1     (dividendo)  ---> ");
  scanf("%f",&a);
  printf(" Entre la parte imaginaria del complejo C1  (dividendo) ---> ");
  scanf("%f",&b);
  c1=complex(a,b);
  printf(" Entre la parte real del complejo C2      (divisor)   ---> ");
  scanf("%f",&a);
```

```
  printf(" Entre la parte imaginaria del complejo C2 (divisor)   ---> ");
  scanf("%f",&b);
  c2=complex(a,b);
  c3=c1/c2;
  printf(" El cociente de (%5.2f %+5.2f i)",c1,imag(c1));
  printf("/( %5.2f %+5.2f i )" ,c2,imag(c2));
  printf("=%5.2f %+5.2f i" ,c3,imag(c3));
}
```

El resultado del programa es:

```
Entre la parte real del complejo C1       (dividendo)     --> 5
Entre la parte imaginaria del complejo C1  (dividendo)  --> 3
Entre la parte real del complejo C2           (divisor)       --> 6
Entre la parte imaginaria del complejo C2 (divisor)       --> 4
El cociente de (5.00 + 3.00 i )/(6.00 +4.00 i )= 0.81 - 0.04 i
```

2.1.5. La Potencia n

Análisis

Dado un número complejo
C1= a1 + b1 *i*, y
n un número entero

La potencia n de C1
$C1^n$= C1*C1*C1*C1.....n veces=C2

Lo que significa que corresponde a una serie de multiplicaciones.

Ejemplos:

C1= 3-2 *i*
n=2

$C2=C1^2$ =(3-2 *i*)*(3-2 *i*)
=5 –12 *i*

C1= 2- *i*
n=3

$C2= C1^3$ =(2- *i*)*(2- *i*)*(2- *i*)
=(3-4 *i*) *(2- *i*)
=2-11 *i*

Diseño del algoritmo

```
LEER a1, b1,n
a2=a1
a2=b1
PARA k DESDE 1 HASTA n
  a2a=a2
  a2=a1*a2+b1*b2;
  b2=a1*b2+a2a*b1;
ESCRIBIR a2 + b2 i
```

Nótese dentro del ciclo la necesidad de la variable a2a para que almacene el valor de a2 antes del calculo de b2.

Programas en lenguaje C para elevar a una potencia entera un complejo

Versión 1

```
#include "stdio.h"
void main()
{  float a1,b1,a2,b2,a2a;
  int n,k;
  printf(" Entre la parte real del complejo C1      --> ");
  scanf("%f",&a1);
  printf(" Entre la parte imaginaria del complejo C1 --> ");
  scanf("%f",&b1);
  printf(" Entre la potencia a elevar C1           --> ");
  scanf("%d",&n);
  a2=a1;
  b2=b1;
  for (k=1; k<n; ++k)
  { a2a=a2;
    a2=a1*a2-b1*b2;
    b2=a1*b2+a2a*b1;
  }
  printf(" La potencia %d de (%5.2f %+5.2f i)= %5.2f %+5.2f i"
         ,n,a1,b1,a2,b2);
}
```

El resultado del programa es:

```
Entre la parte real del complejo C1          ---> 3
Entre la parte imaginaria del complejo C1    --> -2
Entre la potencia a elevar C1                 --> 2
La potencia 2 de (3.00 - 2.00 i )= 5.00 - 12.00 i
```

Versión 2

```
#include "stdio.h"
#include "complex.h"
void main()
{ complex c1,c2;
  float a,b;
  int n,k;
  printf(" Entre la parte real del complejo C1      --> ");
  scanf("%f",&a);
  printf(" Entre la parte imaginaria del complejo C1 --> ");
  scanf("%f",&b);
  c1=complex(a,b);
  printf(" Entre la potencia a elevar C1           --> ");
  scanf("%d",&n);
  c2=c1;
  for (k=1; k<n; ++k)
    c2=c2*c1;
  printf(" La potencia %d de (%5.2f %+5.2f i)" ,n,c1,imag(c1));
  printf("= %5.2f %+5.2f i",c2,imag(c2));
}
```

El resultado del programa es:

```
Entre la parte real del complejo C1          ---> 2
Entre la parte imaginaria del complejo C1    --> -1
Entre la potencia a elevar C1                --> 3
La potencia 3 de (2.00 - 1.00 i )= 2.00 - 11.00 i
```

2.2. Matrices de números complejos

Bajo el mismo análisis de los números reales, una matriz de números complejos consiste en un arreglo bidimensional de números complejos. Es decir que hay una disposición de filas y columnas, donde cada elemento se corresponde con una fila y con una columna únicamente.

Las mismas operaciones que se realizan con matrices de números reales, se aplican a las matrices de números complejos, no podemos perder de vista que los reales son un conjunto propio de los complejos.

Ejemplo de una matriz de complejos:

M1=
(a11,b11), (a12,b12), (a13,b13), (a14,b14), (a15,b15);
(a21,b21), (a22,b22), (a23,b23), (a24,b24), (a25,b25);
(a31,b31), (a32,b32), (a33,b33), (a34,b34), (a35,b35);
(a41,b41), (a42,b42), (a43,b43), (a44,b44), (a45,b45);

Se tiene una matriz compleja de 4 filas y 5 columnas, se puede observar que en la representación cada fila se separa de otra fila con signos de punto y coma, mientras que en la enumeración dentro de la fila se separan los elementos con el signo de coma.

Esta misma matriz en la notación convencional será:

M1=

a11+b11 i	**a12+b12 i**	**a13+b13 i**	**a14+b14 i**	**a15+b15 i**
a21+b21 i	**a22+b22 i**	**a23+b23 i**	**a24+b24 i**	**a25+b25 i**
a31+b31 i	**a32+b32 i**	**a33+b33 i**	**a34+b34 i**	**a35+b35 i**
a41+b41 i	**a42+b42 i**	**a43+b43 i**	**a44+b44 i**	**a45+b45 i**

Esta matriz con valores particulares puede ser:

M1=

8.2+6.3 i	**8.0 +7.3 i**	**-3.5+2.9 i**	**4.3 -14.5 i**	**5.26+5.0 i**
13.1+2.1 i	**9.4-22.2 i**	**23.4+35 i**	**6.34+8.2 i**	**45.2+2.5 i**
15.4+2.8 i	**45.2+2.6 i**	**3.3+4.3 i**	**76.6+5.6 i**	**78.6+8.5 i**
45.2+2.1 i	**42.2+69 i**	**84.5+43 i**	**54.1+4.4 i**	**95.0+4.5 i**

2.3. Operaciones con matrices de complejos

2.3.1. Suma

Análisis

Dadas dos matrices M1 y M2 de tamaño nxm, es decir de n filas y m columnas, se define la suma:

M3=M1+M2

Cada elemento de la matriz M3 en la fila i (con i que va de 1 hasta n) columna j (con j que va de 1 hasta m) es el resultado de la suma entre el elemento de la matriz M1 en la fila i (con i que va de 1 hasta n) columna j (con j que va de 1 hasta m) más el elemento de la matriz M2 en la fila i (con i que va de 1 hasta n) columna j (con j que va de 1 hasta m).

Ejemplo:

M1=

8.2+6.3 i	**8.0 +7.3 i**	**-3.5+2.9 i**
13.1+2.1 i	**9.4-22.2 i**	**23.4+35 i**
15.4+2.8 i	**45.2+2.6 i**	**3.3+4.3 i**
45.2+2.1 i	**42.2+69 i**	**84.5+43 i**

M2=

4.3+5.3 i	5.4 -5.7 i	+1.7+8.7 i
18.8+6.5 i	4.7-62.2 i	-63.1+72 i
10.1 -2.7 i	23.7+1.4 i	9.3-1.8 i
27.2+4.3 i	72.0-49 i	84.5+43 i

M3=M1+M2

M3=

12.5+11.6 i	13.4 +1.6 i	-1.8+11.6 i
31.9+ 8.6 i	14.1-84.4 i	-39.7+107 i
25.5+ 0.1 i	68.9+ 4.0 i	12.6+ 2.5 i
72.4+ 6.4 i	114.2+20 i	169+ 86.0 i

Diseño del algoritmo

```
LEER n,m
LEER M1, M2
PARA i desde 1 hasta n
   PARA j desde 1 hasta m
      M3[i][j]=M1[i][j]+M2[i][j]
ESCRIBIR M3
```

Programa en lenguaje C para sumar matrices complejas

```
#include "stdio.h"
#include "complex.h"
void main()
{ complex M1[5][5],M2[5][5],M3[5][5],eleme;
 float a,b;
 int i,j,m,n;
 printf("Entre el número de filas de las matrices a sumar ---> ");
 scanf("%d",&m);
 printf("Entre el número de columnas de las matrices a sumar ---> ");
 scanf("%d",&n);
 printf("\n LEYENDO LA MATRIZ M1 \n");
 for (i=0; i<m; ++i)
 {
  printf("Leyendo la fila %d \n",i+1);
  for (j=0; j<n; ++j)
  {
   printf("Entre parte real de fila %d columna %d ---> ",i+1,j+1);
   scanf("%f",&a);
   printf("Entre parte imaginaria de fila %d columna %d ---> ",i+1,j+1);
   scanf("%f",&b);
   M1[i][j]=complex(a,b);
   }
```

```
 }
 printf("\n LEYENDO LA MATRIZ M2 \n");
 for (i=0; i<m; ++i)
 {
    printf("Leyendo la fila %d \n",i+1);
    for (j=0; j<n; ++j)
    {
    printf("Entre  parte  real de  fila %d columna %d   ---> ",i+1,j+1);
    scanf("%f",&a);
    printf("Entre parte imaginaria de fila %d columna %d ---> ",i+1,j+1);
    scanf("%f",&b);
    M2[i][j]=complex(a,b);
    }
 }
 for (i=0; i<m; ++i)
    for (j=0; j<n; ++j)
         M3[i][j]=M1[i][j]+M2[i][j];
printf("\ ESTA ES LA MATRIZ M1 \n\n");
 for (i=0; i<m; ++i)
 {
    for (j=0; j<n; ++j)
         printf("%5.1f %+5.1f i ",M1[i][j]);
    printf("\n");
 }
 printf("\n\n ESTA ES LA MATRIZ M2 \n\n");
 for (i=0; i<m; ++i)
 {
    for (j=0; j<n; ++j)
         printf("%5.1f %+5.1f i ",M2[i][j]);
    printf("\n");
 }
 printf("\n\n ESTA ES LA MATRIZ M3=M1+M2 \n\n");
 for (i=0; i<m; ++i)
 {
    for (j=0; j<n; ++j)
         printf("%5.1f %+5.1f i ",M3[i][j]);
    printf("\n");
 }
}
```

El resultado del programa es:

```
Entre el número  de  filas  de las matrices a sumar      --->2
Entre el número de columnas de las matrices a sumar  --->2

 LEYENDO LA MATRIZ M1
Leyendo la fila 1
```

Entre parte real de fila 1 columna 1 --->5
Entre parte imaginaria de fila 1 columna 1 --->3
Entre parte real de fila 1 columna 2 --->6
Entre parte imaginaria de fila 1 columna 2 --->4
Leyendo la fila 2
Entre parte real de fila 2 columna 1 --->3
Entre parte imaginaria de fila 2 columna 1 --->2
Entre parte real de fila 2 columna 2 --->3
Entre parte imaginaria de fila 2 columna 2 --->3

LEYENDO LA MATRIZ M2
Leyendo la fila 1
Entre parte real de fila 1 columna 1 --->5
Entre parte imaginaria de fila 1 columna 1 --->3
Entre parte real de fila 1 columna 2 --->6
Entre parte imaginaria de fila 1 columna 2 --->5
Leyendo la fila 2
Entre parte real de fila 2 columna 1 --->2
Entre parte imaginaria de fila 2 columna 1 --->1
Entre parte real de fila 2 columna 2 --->8
Entre parte imaginaria de fila 2 columna 2 --->9
ESTA ES LA MATRIZ M1

5.0 +3.0 i 6.0 +4.0 i
3.0 +2.0 i 3.0 +3.0 i

ESTA ES LA MATRIZ M2

5.0 +3.0 i 6.0 +5.0 i
2.0 +1.0 i 8.0 +9.0 i

ESTA ES LA MATRIZ M3=M1+M2

10.0 +6.0 i 12.0 +9.0 i
5.0 +3.0 i 11.0 +12.0 i

2.3.2. Resta

Análisis

Dadas dos matrices M1 y M2 de tamaño nxm, es decir de n filas y m columnas, se define la resta:

M3=M1-M2

Cada elemento de la matriz M3 en la fila i (con i que va de 1 hasta n) columna j (con j que va de 1 hasta m) es el resultado de la suma entre el elemento de la matriz M1 en la fila i (con i que va de 1 hasta n) columna j (con j que va de 1 hasta m) menos el elemento de la matriz M2 en la fila i(con i que va de 1 hasta n) columna j (con j que va de 1 hasta m).

Ejemplo:

M1=

```
   8.2+6.3 i   8.0 +7.3  i   -3.5+2.9 i
  13.1+2.1 i   9.4-22.2  i   23.4+35 i
  15.4+2.8 i   45.2+2.6 i   3.3+4.3  i
  45.2+2.1 i   42.2+69  i   84.5+43 i
```

M2=

```
   4.3+5.3 i   5.4 -5.7  i   +1.7+8.7 i
  18.8+6.5 i   4.7-62.2  i   -63.1+72 i
  10.1 -2.7 i   23.7+1.4 i    9.3-1.8  i
  27.2+4.3 i   72.0-49  i   84.5+43 i
```

M3=M1-M2

M3=

```
   3.9  +  1.0 i     2.6 +13.0  i   -5.2    -5.8 i
  -5.7 + -4.4 i     4.7  -40.0  i   +86.5-37.0 i
   5.3  +  5.5 i     21.5+  1.2 i   - 6.0+ 26.1 i
  18.0  -  2.2 i  -29.8+118.0 i    +  0.0+ 0.0 i
```

Diseño del algoritmo

```
LEER n,m
LEER M1, M2
PARA i desde 1 hasta n
   PARA j desde 1 hasta m
      M3[i][j]=M1[i][j]+M2[i][j]
ESCRIBIR M3
```

Programa en lenguaje C para restar matrices complejas

```
#include "stdio.h"
#include "complex.h"
void main()
{ complex M1[5][5],M2[5][5],M3[5][5],eleme;
  float a,b;
  int i,j,m,n;
```

```
printf("Entre el número de filas de las matrices a sumar  ---> ");
scanf("%d",&m);
printf("Entre el número de columnas de las matrices a sumar  ---> ");
scanf("%d",&n);
printf("\n LEYENDO LA MATRIZ M1 \n\n");
for (i=0; i<m; ++i)
{
   printf("\nLeyendo la fila %d \n",i+1);
   for (j=0; j<n; ++j)
   { printf("Entre  parte  real de  fila %d columna %d ---> ",i+1,j+1);
     scanf("%f",&a);
     printf("Entre parte imaginaria de fila %d columna %d ---> ",i+1,j+1);
     scanf("%f",&b);
     M1[i][j]=complex(a,b);
   }
}
printf("\n LEYENDO LA MATRIZ M2 \n\n");
for (i=0; i<m; ++i)
{
   printf("\nLeyendo la fila %d \n",i+1);
   for (j=0; j<n; ++j)
   {
     printf("Entre  parte  real de  fila %d columna %d ---> ",i+1,j+1);
     scanf("%f",&a);
     printf("Entre parte imaginaria de fila %d columna %d ---> ",i+1,j+1);
     scanf("%f",&b);
     M2[i][j]=complex(a,b);
   }
}
for (i=0; i<m; ++i)
   for (j=0; j<n; ++j)
        M3[i][j]=M1[i][j]-M2[i][j];
printf("\ ESTA ES LA MATRIZ M1 \n\n");
for (i=0; i<m; ++i)
{
   for (j=0; j<n; ++j)
        printf("%5.1f %+5.1f i ",M1[i][j]);
   printf("\n");
}
printf("\n\n ESTA ES LA MATRIZ M2 \n\n");
for (i=0; i<m; ++i)
{
   for (j=0; j<n; ++j)
        printf("%5.1f %+5.1f i ",M2[i][j]);
   printf("\n");
}
printf("\n\n ESTA ES LA MATRIZ M3=M1-M2 \n\n");
```

```
 for (i=0; i<m; ++i)
 {
    for (j=0; j<n; ++j)
          printf("%5.1f %+5.1f i ",M3[i][j]);
    printf("\n");
 }
}
```

2.3.3. Trasposición

Análisis

Dada una matriz: M1 de tamaño nxm, es decir de n filas y m columnas, se define la matriz M' transpuesta de mxn, en la cual se intercambia filas por columnas.

M1=

a11+b11 i	a12+b12 i	a13+b13 i	a14+b14 i	a15+b15 i
a21+b21 i	a22+b22 i	a23+b23 i	a24+b24 i	a25+b25 i
a31+b31 i	a32+b32 i	a33+b33 i	a34+b34 i	a35+b35 i
a41+b41 i	a42+b42 i	a43+b43 i	a44+b44 i	a45+b45 i

M'=

a11+b11 i	a21+b21 i	a31+b31 i	a41+b41 i
a12+b12 i	a22+b22 i	a32+b32 i	a42+b42 i
a13+b13 i	a23+b23 i	a33+b33 i	a43+b43 i
a14+b14 i	a24+b24 i	a34+b34 i	a44+b44 i
a15+b15 i	a25+b25 i	a35+b35 i	a45+b45 i

Ejemplo:

M1=

4.3+5.3 i	5.4 -5.7 i	+1.7+8.7 i
18.8+6.5 i	4.7-62.2 i	-63.1+72 i
10.1 -2.7 i	23.7+1.4 i	9.3-1.8 i
6.2+8.2 i	67.1+9.3 i	8.4-5.2 i

Obsérvese que M1 tiene 4 filas y 3 columnas

M'=

4.3 +5.3 i	18.8 +6.5 i	10.1 –2.7 i	6.2 +8.2 i
5.4 –5.7 i.	4.7 –62.2 i	23.7+1.4 i	67.1+9.3 i
1.7 +8.7 i	-63.1 +72.0 i	9.3 -1.8 i	8.4 -5.2 i

Esta matriz transpuesta tiene 3 filas y 4 columnas

Diseño del algoritmo

```
LEER n,m
LEER M1
PARA i desde 1 hasta m
   PARA j desde 1 hasta n
      Mt[i][j]=M1[j][i]
ESCRIBIR Mt
```

Programa en lenguaje C para transponer matrices complejas

```
#include "stdio.h"
#include "complex.h"
void main()
{ complex M1[5][5],Mt[5][5];
 float a,b;
 int i,j,m,n;
 printf("Entre el número de filas de la matriz a transponer ---> ");
 scanf("%d",&m);
 printf("Entre el número de columnas de la matriz a transponer ---> ");
 scanf("%d",&n);

 printf("\n LEYENDO LA MATRIZ M1 \n\n");
 for (i=0; i<m; ++i)
 {
    printf("\nLeyendo la fila %d \n",i+1);
    for (j=0; j<n; ++j)
    { printf("Entre parte real de fila %d columna %d ---> ",i+1,j+1);
     scanf("%f",&a);
     printf("Entre parte imaginaria de fila %d columna %d ---> ",i+1,j+1);
     scanf("%f",&b);
     M1[i][j]=complex(a,b);
    }
 }
for (i=0; i<m; ++i)
    for (j=0; j<n; ++j)
         Mt[j][i]=M1[i][j];
 printf("\ ESTA ES LA MATRIZ M1 \n\n");
 for (i=0; i<m; ++i)
 {
    for (j=0; j<n; ++j)
         printf("%5.1f %+5.1f i ",M1[i][j]);
    printf("\n");
 }
```

```
 printf("\n\n ESTA ES LA MATRIZ Mt \n\n");
 for (i=0; i<n; ++i)
 {
    for (j=0; j<m; ++j)
         printf("%5.1f %+5.1f i ",Mt[i][j]);
    printf("\n");
 }
}
```

2.3.4. Multiplicación

Análisis

Dadas dos matrices: M1 de tamaño nxm, es decir de n filas y m columnas y M2 de tamaño mxp; obsérvese que el número de columnas de M1 debe ser igual al número de filas de M2, se define el producto:

M3=M1*M2

Cada elemento de la matriz M3 en la fila i (con i que va de 1 hasta n) columna j (con j que va de 1 hasta m) es el resultado de la sumatoria de los productos de elemento por elemento entre la fila i (con i que va de 1 hasta n) de la matriz M1 con la columna j (con j que va de 1 hasta n).de la matriz M2.

Ejemplo:

M1=

```
   8.2+6.3 i   8.0 +7.3  i   -3.5+2.9 i
  13.1+2.1 i   9.4-22.2  i   23.4+35 i
  15.4+2.8 i   45.2+2.6 i   3.3+4.3  i
  45.2+2.1 i   42.2+69  i   84.5+43 i
```

M2=

```
   4.3+5.3 i   5.4 -5.7  i   +1.7+8.7 i
  18.8+6.5 i   4.7-62.2  i   -63.1+72 i
  10.1 -2.7 i   23.7+1.4 i    9.3-1.8  i
```

M3=M1*M2

M3=

```
   77.3+ 298.5 i   484.8 –412.2  i   -1098.6+230.7 i
  697.1+  12.5 i  -748.4 + 109.9 i   1289.9+2478.5 i
  929.2+ 470.9 i   545.5 -2765.4 i  -2999.1+3263.1 i
 1497.6+2026.2 i   6688.6-1409.4 i   -6709.0 –670.9 i
```

Diseño del algoritmo

```
LEER n,m,p
LEER M1, M2
PARA i desde 1 hasta n
   PARA j desde 1 hasta p
      M3[i][j]=0+0i
      PARA k desde 1 hasta m
         M3[i][j]=M1[i][k]+M2[k][j]
ESCRIBIR M3
```

Programa en lenguaje C para multiplicar dos matrices complejas

```
#include "stdio.h"
#include "complex.h"
void main()
{ complex M1[5][5],M2[5][5],M3[5][5],eleme;
 float a,b;
 int i,j,k,m,n,p;
 printf("Entre el número  de  filas  de las matrices M1 a multiplicar  ---> ");
 scanf("%d",&m);
 printf("Entre el número de columnas de las matrices M1 a multiplicar  ---> ");
 scanf("%d",&n);
 printf("Entre el número de columnas de las matrices M2 a multiplicar  ---> ");
 scanf("%d",&p);

 printf("\n LEYENDO LA MATRIZ M1 \n\n");
 for (i=0; i<m; ++i)
 {  printf("\nLeyendo la fila %d \n",i+1);
    for (j=0; j<n; ++j)
    {
         printf("Entre  parte  real de  fila %d columna %d ---> ",i+1,j+1);
         scanf("%f",&a);
         printf("Entre parte imaginaria de fila %d columna %d ---> ",i+1,j+1);
         scanf("%f",&b);
         M1[i][j]=complex(a,b);
    }
 }
 printf("\n LEYENDO LA MATRIZ M2 \n\n");
 for (i=0; i<n; ++i)
 {  printf("\nLeyendo la fila %d \n",i+1);
    for (j=0; j<p; ++j)
    {    printf("Entre  parte  real de  fila %d columna %d ---> ",i+1,j+1);
         scanf("%f",&a);
         printf("Entre parte imaginaria de fila %d columna %d ---> ",i+1,j+1);
         scanf("%f",&b);
```

```
            M2[i][j]=complex(a,b);
        }
    }
    for (i=0; i<m; ++i)
        for (j=0; j<p; ++j)
        {   M3[i][j]=complex(0,0);
            for (k=0; k<n; ++k)
                M3[i][j]+=M1[i][k]*M2[k][j];
        }
printf("\ ESTA ES LA MATRIZ M1 \n\n");
    for (i=0; i<m; ++i)
    {   for (j=0; j<n; ++j)
            printf("%5.1f %+5.1f i ",M1[i][j]);
        printf("\n");
    }
    printf("\n\n ESTA ES LA MATRIZ M2 \n\n");
    for (i=0; i<n; ++i)
    {   for (j=0; j<p; ++j)
            printf("%5.1f %+5.1f i ",M2[i][j]);
        printf("\n");
    }
    printf("\n\n ESTA ES LA MATRIZ M3=M1*M2 \n\n");
    for (i=0; i<m; ++i)
    {   for (j=0; j<p; ++j)
            printf("%5.1f %+5.1f i ",M3[i][j]);
        printf("\n");
    }
}
```

2.3.5. Inversión

Análisis

Dada una matriz: M1 de tamaño nxn, es decir de igual número de filas y columnas, se define la matriz inversa como:

$M1^{-1}$

Se define además la matriz identidad compleja, como la matriz identidad real, una matriz cuadrada en donde todos los elementos de la diagonal son unos y el resto de los elementos son ceros, además que todas las partes imaginarias son ceros:

I=

1+0 i	0 +0 i	0+0 i	0 +0 i	0+0 i
0+0 i	1 +0 i	0+0 i	0 +0 i	0+0 i
0+0 i	0 +0 i	1+0 i	0 +0 i	0+0 i

0+0 i	0 +0 i	0+0 i	1 +0 i	0+0 i
0+0 i	0 +0 i	0+0 i	0 +0 i	1+0 i

Esta matriz I tiene orden 5, quiere decir que su tamaño es de 5x5.

La matriz inversa $M1^{-1}$ está dada por aquella matriz cuyo producto de $M1 * M1^{-1} = I$

Realizando ciertas operaciones entre filas sobre la matriz M1 se puede convertir ésta en la matriz identidad. Habiendo realizado exactamente esas mismas operaciones a la matriz I, ésta se convierte en la matriz $M1^{-1}$.

Dichas operaciones entre filas consisten en: a- una fila reemplazarla por esa fila multiplicada (todos sus elementos) o dividida por una constante (un número complejo); y b- una fila reemplazarla por la diferencia (o suma) miembro a miembro entre esa fila (todos sus elementos) y otra fila (todos sus elementos).

Ejemplo:

M1=

8.2+6.3 i	8.0 +7.3 i	-3.5+2.9 i
13.1+2.1 i	9.4-22.2 i	23.4+35 i
15.4+2.8 i	45.2+2.6 i	3.3+4.3 i

Tenemos una matriz compleja de orden 3, o sea de tamaño 3x3, a la cual se requiere obtener la matriz inversa $M1^{-1}$.

Se toma M1 e I

M1=

8.2+6.3 i	8.0 +7.3 i	-3.5+2.9 i
13.1+2.1 i	9.4-22.2 i	23.4+35 i
15.4+2.8 i	45.2+2.6 i	3.3+4.3 i

I=

1+0 i	0+0 i	0+0 i
0+0 i	1+0 i	0+0 i
0+0 i	0+0 i	1+0 i

La fila 1 de la matriz M1 se divide entre la constante compleja 8.2+6.3i quedando:

M1=

1.0+ 0.0 i	1.04358+ 0.08847 i	-0.09754 +0.42860 i
13.1+2.1 i	9.4 -22.2 i	23.4 +35.0 i
15.4+2.8 i	45.2 +2.6 i	3.3 +4.3 i

La fila 1 de la matriz I se divide entre la constante compleja 8.2+6.3i quedando:

I=

0.07669-0.05892 i	0.0 +0.0 i	0.0+0.0 i
0.0 +0.0 i	1.0 +0.0 i	0.0+0.0 i
0.0 +0.0 i	0.0 +0.0 i	1.0+0.0 i

La fila 2 de la matriz M1 se reemplaza por la diferencia entre la fila 2 y la fila 1 multiplicada por la constante 13.1+2.1 i quedando:

M1=

1.0+ 0.0 i	1.04358+ 0.08847 i	-0.09754 +0.42860 i
0.0+0.0 i	-4.08511-25.55046 i	25.57784+29.59020 i
15.4+2.8 i	45.2 +2.6 i	3.3 +4.3 i

La fila 2 de la matriz I se reemplaza por la diferencia entre la fila 2 y la fila 1 multiplicada por la constante 13.1+2.1 i quedando:

I=

0.07669-0.05892 i	0.0 +0.0 i	0.0+0.0 i
-2.24196-0.83864 i	1.0 +0.0 i	0.0+0.0 i
0.0 +0.0 i	0.0 +0.0 i	1.0+0.0 i

La fila 3 de la matriz M1 se reemplaza por la diferencia entre la fila 3 y la fila 1 multiplicada por la constante 15.4+2.8 i quedando:

M1=

1.0+ 0.0 i	1.04358+ 0.08847 i	-0.09754 +0.42860 i
0.0+ 0.0 i	-4.08511-25.55048 i	25.57783+29.59017 i
0.0+ 0.0 i	29.37658 -1.68445 i	6.00220 - 2.02730 i

La fila 3 de la matriz I se reemplaza por la diferencia entre la fila 3 y la fila 1 multiplicada por la constante 15.4+2.8 i quedando:

I=

0.07669-0.05892 i	0.0 +0.0 i	0.0+0.0 i
-2.24196-0.83864 i	1.0 +0.0 i	0.0+0.0 i
0.0 +0.0 i	0.0 +0.0 i	1.0+0.0 i

Diseño del algoritmo

Programa en lenguaje C para invertir matrices

```
#include "stdio.h"
#include "complex.h"
void main()
```

```
{ complex M[5][5],M1[5][5],Minv[5][5],uno,cero;
 float a,b;
 int i,j,k,m;
 printf("Entre el número de filas de las matriz a invertir ---> ");
 scanf("%d",&m);
 printf("\n LEYENDO LA MATRIZ M \n\n");
 for (i=0; i<m; ++i)
 {
   printf("\nLeyendo la fila %d \n",i+1);
   for (j=0; j<m; ++j)
   {
        printf("Entre parte real de fila %d columna %d ---> ",i+1,j+1);
        scanf("%f",&a);
        printf("Entre parte imaginaria de fila %d columna %d ---> ",i+1,j+1);
        scanf("%f",&b);
        M[i][j]=complex(a,b);
        M1[i][j]=complex(a,b);
        if (i==j)
          Minv[i][j]=complex(1,0);
        else
        Minv[i][j]=complex(0,0);
   }
 }
 for (i=0; i<m; ++i)
 {  uno=M1[i][i];
   for (j=0; j<m; ++j)
   {  M1[i][j]/=uno;
        Minv[i][j]/=uno;
   }
   for (k=0; k<m; ++k)
   {  if (i != k)
      {  cero=M1[k][i];
          for (j=0; j<m; ++j)
          {
          M1[k][j]-=cero*M1[i][j];
               Minv[k][j]-=cero*Minv[i][j];
          }
        }
   }
 }
 printf("\ ESTA ES LA MATRIZ M a invertir \n\n");
 for (i=0; i<m; ++i)
 {
   for (j=0; j<m; ++j)
        printf("%5.1f %+5.1f i ",M[i][j]);
   printf("\n");
 }
```

```
 printf("\n\n ESTA ES LA MATRIZ M1 inversa \n\n");
 for (i=0; i<m; ++i)
 {
    for (j=0; j<m; ++j)
         printf("%+7.5f %+7.5fi  ",Minv[i][j]);
    printf("\n");
 }
}
```

2.3. Vectores

Se define un vector de números complejos como un conjunto finito de esos mismos números. Desde un punto de vista práctico es ventajoso considerar que un número complejo es una pareja ordenada de la forma (a, b) de números reales donde a y b se ajustan a ciertas normas operativas semejantes a las reglas de los números reales.

Por ejemplo, definimos el vector:

V1=(a1,b1), (a2,b2), (a3,b3), (a4,b4).
Como representación del conjunto de números reales
V1=a1+b1 *i* , a2+b2 *i* , a3+b3 *i* , a4+b4 *i*.

3. RAICES DE FUNCIONES COMPLEJAS

Encontrar una raíz de una función, consiste en averiguar el valor que toma la variable independiente cuando la variable dependiente sea nula, esto es: dada la función y=f(x), una raíz está dada por el valor de x tal que f(x)=0.

3.1. La raíz para la función de la forma f(x)=ax+b donde a es diferente de 0

Análisis

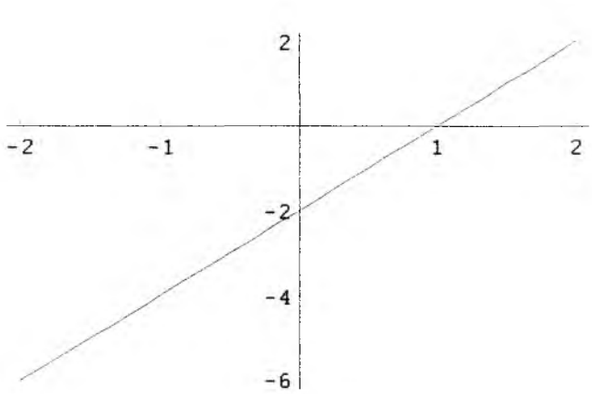
f(x)=ax+b=0

x=-b/a

Ejemplo:

Para la función f(x)=2x-2
La raíz es: x=-(-2)/2=1
Se prueba que: f(1)=0
entonces: x=1 es la raíz de f(x) =2x-2.

En la gráfica 3.1.1. se puede observar que la línea que representa esta función en el plano x,y cruza al eje x en 1:



Gráfica 3.1.1. Función f(x) =2x-2

Diseño del Algoritmo

LEER a, b
x=-b/a
ESCRIBIR x

Programa en lenguaje C para calcular la raíz la función f(x)=ax+b donde a es diferente de 0

```
#include "stdio.h"
void main()
{  float a, b, x;
   printf("Entre el coeficiente de x: a= ") ;
   scanf("%f", &a);
   printf("Entre el término independiente: b= ") ;
   scanf("%f", &b);
   x=-b/a ;
   printf(" LA RAIZ ES:\n");
   printf("x=%8.4f ",x) ;
}
```

3.2. Raíces para la función de la forma $f(x)=ax^2+bx+c$, donde a diferente de 0

Análisis

$f(x)=ax^2+bx+c=0$
$4a(ax^2+bx) + b^2= 4a(-c)+b^2$
$4a^2x^2+4abx+ b^2= b^2-4ac$
$(2ax+b)^2= b^2-4ac$

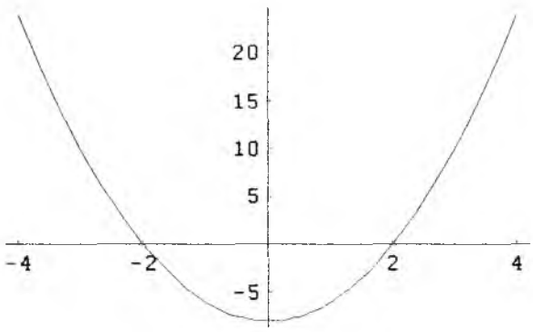
Sacando raíz cuadrada a ambos miembros, tenemos dos raíces:

$x1=(-b+(b^2-4ac)^{1/2}/(2\ a)$
$x2=(-b-(b^2-4ac)^{1/2}/(2\ a)$

Para el caso que $b^2-4ac >= 0$, hay dos raíces reales.

Ejemplo

La función $f(x)=2x^2-8$, posee dos raíces reales en:
x=-2 ; x=2 ;



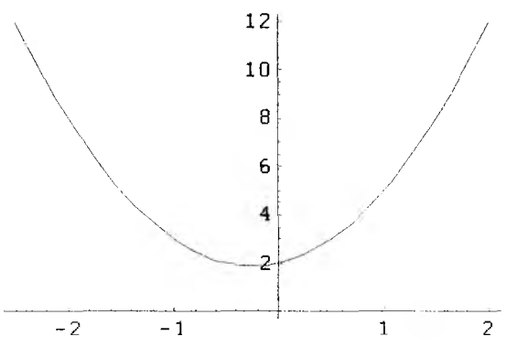
Gráfica 3.2.1 Función $f(x)=2x^2-8$

Para el caso: $b^2-4ac < 0$, las dos raíces son números complejos

Ejemplo:

La función $f(x)=2x^2+x+2$, posee 2 raíces complejas:

$x_1=-0.25 + 0.9682\,i$;
$x_2=-0.25 - 0.9682\,i$



Gráfica 3.2.2. Función $f(x)=2x^2+x+2$

Diseño del Algoritmo

```
LEER a,b,c
cs=b*b-4*a*c ;
SI cs>0
  x1=(-b+sqrt(cs))/(2*a)
  x2=(-b+sqrt(cs))/(2*a)
  ESCRIBIR x1, x2
SINO
  x1=-b/(2*a)
  x2=x1 ;
  x1i=sqrt(-cs)/(2*a)
  x2i=-sqrt(-cs)/(2*a)
ESCRIBIR x1, x1i, x2, x2i
```

Programa en lenguaje C para calcular las raíces de la función $f(x)=ax^2+bx+c$, donde a diferente de 0.

```
#include "stdio.h"
#include "math.h"
void main()
{  float a, b, c, cs, x1, x2, x1i ,x2i ;
   printf("Entre el coeficiente de x^2    (a) ---> ") ;
   scanf("%f", &a);
   printf("Entre el coeficiente de x      (b) ---> ") ;
   scanf("%f", &b);
   printf("Entre el término independiente (c) ---> ") ;
   scanf("%f", &c);
   cs=b*b-4*a*c ;
   printf(" LAS RAICES SON:\n");
   if (cs>=0)
   {   x1=(-b+sqrt(cs))/(2*a) ;
       x2=(-b-sqrt(cs))/(2*a) ;
          printf(" x1=%8.4f\n x2=%8.4f",x1,x2) ;
   }
   else
   {   x1=-b/(2*a) ;  x2=x1 ;
          x1i=sqrt(-cs)/(2*a) ;
          x2i=-x1i ;
          printf("x1=%8.4f %+8.4f i\n",x1,x1i) ;
          printf("x2=%8.4f %+8.4f i",x2,x2i) ;
   }
}
```

3.3. Raíces para la función de la forma $f(x)=a_1x^n+a_2x^{n-1}+a_3x^{n-2}+...+a_nx+a_{n+1}$

Análisis

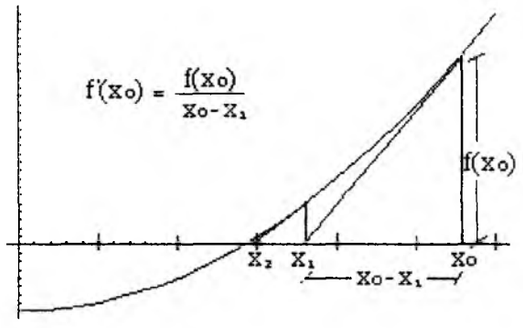
Una función de esta forma es ampliamente conocida como función polinomial, se puede ver que es la generalización de los numerales anteriores (polinomios de grado uno y dos respectivamente), cuando el grado del polinomio es mayor a 4, es difícil encontrar una fórmula tal como se hizo en la sección 3.2.1con el polinomio de grado 2.

Un método ampliamente conocido para encontrar una raíz real de una función de esta forma consiste:
Se da un valor inicial a la variable $x=x_0$, se calcula la función en x_0, esto es $f(x_0)$.

Luego se calcula la función derivada (polinomio derivado) en x_0, esto es $f'(x_0)$, donde:

$$f'(x)=na_1x^{n-1}+(n-1)a_2x^{n-2}+(n-2)a_3x^{n-3}+...+a_n$$

El valor de $f'(x_0)$ corresponde a la pendiente de la función en x_0. Si trazamos una recta tangente a la función en x_0 encontramos que esa recta corta al eje x en el punto x_1.



Gráfica 3.3.1. Tangente de la curva en x_0

Ahora se calcula un nuevo valor de x, x_1, restándole a x_0 el cociente de $f(x_0)$ entre $f'(x_0)$., así:

$$x_1 = x_0 - f(x_0)/f'(x_0)$$

Se repite este proceso encontrando nuevos valores de x: $x_2, x_3, x_4, \ldots x_n$, donde:

$$x_n = x_{n-1} - f(x_{n-1})/f'(x_{n-1})$$

Un conjunto de nuevos valores de x , pueden aproximarse o alejarse a una raíz real del polinomio, para el primer caso se dice que el método converge, en el otro caso se dice que diverge. Si x_n es una raíz de la función cuando $f(x_n)$, cuando se aproxima suficientemente a 0, esto es $f(x_n)$->0. Una aproximación suficiente estará determinada por un error que definiremos como un número real positivo cercano a cero, por ejemplo 10^{-5}, 10^{-7}, 10^{-12} , ahora como la aproximación a la raíz se puede dar por debajo (negativa) o por encima (positiva), se toma como criterio que x_n es raíz de f(x) cuando: valor absoluto de $f(x_n)$ sea menor o igual que error.

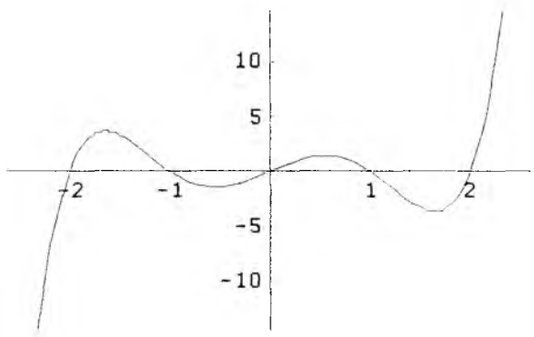
Ejemplo:

La función $f(x)= x^5-5x^3+4x$, posee 5 raíces en:

x=-2 ; x=-1 ; x=0 ; x=1 ; x=2 ;

La función derivada es:

$f'(x)=5x^4-15x^2+4$;



Gráfica 3.3.1. función $f(x)= x^5-5x^3+4x$

Se da un valor inicial a x: x_0 =3,

f(3)=120;
f'(3)=274;
x_1=3-120/274=2.562;

f(2.562)=36.5465;
f'(2.562)=120.963;
x_2=2.562-36.5465/120.963=2.26;

f(2.26)=10.282;
f'(2.26)=57.824;
x_3=2.26-10.282/57.824=2.082;

f(2.082)=2.324;
f'(2.082)=32.928;
x_4=2.082-2.324/32.928=2.011;

f(2.011)=0.270;
f'(2.011)=25.113;
x_5=2.011-0.270/25.113=2.00024;

Se observa que el valor 2.00024 para x es una aproximación a la raíz. Dicha aproximación se puede hacer mayor en la medida que se necesite: si se requiere mayor aproximación, entonces se determina un nivel de error menor.

Diseño del Algoritmo

```
LEER P(x)
DERIVAR P(x)
Xo=0
Ciclos=0
Error=0.001
REPITA
  Xn=Xo- P(Xo)/P'(Xo)
  Xo=Xn
  Ciclos=Ciclos + 1
HASTA (que Abs(P(Xo)) <= Error) ó (Ciclos > 30)
SI Ciclos <= 30
   ESCRIBIR Xo
SINO
   ESCRIBIR "No se encontró raíz en 30 ciclos"
```

Programas en lenguaje C para calcular raíces de la función
$f(x)=a_1x^n+a_2x^{n-1}+a_3x^{n-2}+...+a_nx+a_{n+1}$

Versión 1.

```
#include "stdio.h"
#include "math.h"
float co[20],res;
int i,g;
float p(float x)
{ res=0;
  for(i=0; i<g+1; i++)
    res=res+co[i]*pow(x,i);
  return(res);
}
float pp(float x)
{ res=0;
  for(i=1; i<g+1; i++)
    res= res+i*co[i]*pow(x,i-1);
  return(res);
}
void main()
{ float xo=-10,xn,pxn,e=0.00001;
  int j,c;
  printf("Entre el grado del polinomio -> ");
  scanf("%d",&g);
  for(j=0; j<g+1; j++)
  { printf("Entre el coeficiente de x^%d  =",j);
    scanf("%f",&co[j]);
  }
  c=0;
  do
  {  xn=xo-p(xo)/pp(xo);
     xo=xn;
     c++;
     pxn=p(xn);
     if (pxn <0) pxn= -pxn;
  } while (pxn>e && c<30);
  if (c<30)
      printf("la raiz es %8.3f ",xn);
  else printf("NO hubo raiz en 30 ciclos ");
  scanf("%d",&g);
}
```

Versión 2

```
#include "stdio.h"
```

```
#include "math.h"
void scom(float a, float ac, float b, float bc, float *r, float *rc)
{
 *r=a+b;
 *rc=ac+bc;
}
void mcom(float a, float ac, float b, float bc, float *r, float *rc)
{  *r=a*b-ac*bc;
   *rc=b*ac+a*bc;
}
void dcom(float a, float ac, float b, float bc, float *r, float *rc)
{  float p,pc,q,qc;
   mcom(a,ac,b,-bc,&p,&pc);
   mcom(b,bc,b,-bc,&q,&qc);
   *r=p/q;
   *rc=pc/q;
}
main()
{ float aprox = 0.00001, A[20], Ac[20],B[19],Bc[19],C[20],
      Cc[20],x,xc,z,zc,y,yc,r,rc;
  int ngrad,n,m,i,nr,l,nmi=30;
  printf("Entre el grado polinomio  ---> ");
  scanf("%d",&ngrad);
  n=ngrad;
  m=n+1;
  for(i=0; i<m; i++)
  {  printf("Entre el valor del coeficiente de x a la %d ---> ",n-i);
     scanf("%f",&A[i]);
     Ac[i]=0;
  }
  for (nr=1; nr<=ngrad; nr++)
  {
     x=0.1;
     xc=1;
     l=0;
     do
     {  B[0]=A[0];
          Bc[0]=Ac[0];
          for(i=1; i<m; i++)
          { mcom(x,xc,B[i-1],Bc[i-1],&z,&zc);
            scom(A[i],Ac[i],z,zc,&B[i],&Bc[i]);
          }
          C[0]=B[0];
          Cc[0]=Bc[0];
          for(i=1; i<n; i++)
          { mcom(x,xc,C[i-1],Cc[i-1],&z,&zc);
```

```
            scom(B[i],Bc[i],z,zc,&C[i],&Cc[i]);
          }
          dcom(B[m-1],Bc[m-1],C[n-1],Cc[n-1],&z,&zc);
          scom(x,xc,-z,-zc,&y,&yc);
          dcom(x-y,xc-yc,y,yc,&r,&rc);
          r=pow(pow(r,2)+pow(rc,2),0.5);
          if (r<0) r=-r;
          l++;
          x=y;
          xc=yc;
      } while((l<nmi) && (r>aprox));
      if (r<=aprox)
      {
       printf("La raiz es %8.4f%+8.4fi \n",x,xc);
       for( i=1; i<=n; i++)
       {
            A[i]=B[i];
            Ac[i]=Bc[i];
       }
       n=n-1;
       m=m-1;
      }
      else printf("No hubo raiz");
    }
}
```

3.4. Raíces para la función de la forma
$f(x)=(a_1+b_1\ i)x^n+(a_2+b_2\ i)x^{n-1}+(a_3+b_3\ i)x^{n-2}+...+(a_n+b_n\ i)x+(a_{n+1}+b_{n+1}\ i)$

Análisis

Una función de esta forma es un polinomio con coeficientes complejos, se puede ver que es la generalización de los numerales anteriores (polinomios con coeficientes reales).

Se generaliza el mismo método :

Se da un valor inicial a la variable $x=ax_0+bx_0\ i$, se calcula la función en $(ax_0+bx_0\ i)$, esto es $f(ax_0+bx_0\ i)$.

Luego se calcula la función derivada (polinomio derivado) en $(ax_0+bx_0\ i)$, esto es: $f'(ax_0+bx_0\ i)$, donde:

$$f'(x)=n(a_1+b_1\ i)x^{n-1}+(n-1)\ (a_2+b_2\ i)x^{n-2}+(n-2)\ (a_3+b_3\ i)x^{n-3}+...+(a_n+b_n\ i)$$

El valor de $f'(ax_0+bx_0\ i)$ corresponde a la derivada de la función en $ax_0+bx_0\ i$.

Se encuentra un nuevo valor complejo en $ax_1+bx_1\ i$ así:

$$ax_1+bx_1\ i=(ax_0+bx_0\ i)-f(ax_0+bx_0\ i)/f'(ax_0+bx_0\ i)$$

Se repite este proceso encontrando nuevos valores de :

$(ax_2+bx_2\ i)$, $(ax_3+bx_3\ i)$, $(ax_4+bx_4\ i)$,... $(ax_n+bx_n\ i)$, donde:

$$(ax_n+bx_n\ i)=(ax_{n-1}+bx_{n-1}\ i)-f(ax_{n-1}+bx_{n-1}\ i)/f'(ax_{n-1}+bx_{n-1}\ i)$$

Un conjunto de nuevos valores de $(ax+bx\ i)$, pueden aproximarse o alejarse a una raíz compleja del polinomio, para el primer caso se dice que el método converge, en el otro caso se dice que diverge. $(ax_n+bx_n\ i)$ es una raíz de la función cuando $f(ax_n+bx_n\ i)=0+0\ i$, cuando se aproxima suficientemente a $0+0\ i$, esto es $f(ax_n+bx_n\ i)->0+0\ i$. Una aproximación suficiente estará determinada por un error que definiremos como un número real positivo cercano a cero, por ejemplo 10^{-5}, 10^{-7}, 10^{-12} , ahora como la aproximación a la raíz se da a través de un número complejo, se toma como criterio que $ax_n+bx_n\ i$ es raíz de $f(ax+bx\ i)$ cuando:

La magnitud de $f(ax_n+bx_n\ i)$ sea menor o igual que error, esto es: $((ax_n)^2+(bx_n)^2)^{1/2}$ <= error.

Programas en lenguaje C para calcular raíces de la función
$f(x)=(a_1+b_1\,i)x^n+(a_2+b_2\,i)x^{n-1}+(a_3+b_3\,i)x^{n-2}+...+(a_n+b_n\,i)x+(a_{n+1}+b_{n+1}\,i)$

Versión 1

```
#include "stdio.h"
#include "math.h"
#include "complex.h"
complex co[20],res;
int i,g;
complex p(complex x)
{ res=complex(0,0);
  for(i=0; i<g+1; i++)
    res=res+co[i]*pow(x,i);
}return(res);
complex pp(complex x)
{ res=complex(0,0);
  for(i=1; i<g+1; i++)
    res= res+i*co[i]*pow(x,i-1);
  return(res);
}
void main()
{ complex xo,xn,pxn;
  float a,b, e=0.001;
  int j,c;
  printf("Entre el grado del polinomio");
  scanf("%d",&g);
  for(j=0; j<g+1; j++)
  { printf("Entre el coeficiente  real de x^%d ---> ",j);
    scanf("%f",&a);
    printf("Entre el coeficiente imaginario de x^%d  ---> ",j);
    scanf("%f",&b);
    co[j]=complex(a,b);
  }
  c=0;
  xo=complex(-15,0);
  do
  {  xn=xo-p(xo)/pp(xo);
     xo=xn;
     c++;
     pxn=p(xn);
  } while (abs(pxn)>e && c<30);
  if (c<30) printf("la raiz es %8.3f %+8.3f",xn,imag(xn));
  else printf("NO hubo raiz en 30 ciclos ");
```

```
}
```

Versión 2

```
#include "stdio.h"
#include "math.h"
#include "complex.h"
main()
{ float aprox = 0.00001,a,b,r;
  complex A[20],B[19],C[20],x,z,y;
  int ngrad,n,m,i,nr,l,nmi=30;
  printf("Entre el grado polinomio  ---> ");
  scanf("%d",&ngrad);
  n=ngrad;
  m=n+1;
  for(i=0; i<m; i++)
  { printf("Entre el valor  real  del  coeficiente de x a la %d ---> ",n-i);
    scanf("%f",&a);
    printf("Entre el valor imaginario del coeficiente de x a la %d ---> ",n-i);
    scanf("%f",&b);
    A[i]=complex(a,b);
  }
  for (nr=1; nr<=ngrad; nr++)
  {
    x=complex(0.1,0);
    l=0;
    do
    {  B[0]=A[0];
          for(i=1; i<m; i++)
          {
            z=B[i-1]*x;
            B[i]=A[i]+z;
          }
          C[0]=B[0];
          for(i=1; i<n; i++)
          {
            z=C[i-1]*x;
            C[i]=B[i]+z;
          }
        z=B[m-1]/C[n-1];
        y=x-z;
        r=abs(((x-y)/y));
          l++;
          x=y;
    } while((l<nmi) && (r>aprox));
    if (r<=aprox)
```

```
    {
     printf("La raiz es %8.4f%+8.4fi \n",x,imag(x));
     for( i=1; i<=n; i++)
          A[i]=B[i];
     n=n-1;
     m=m-1;
    }
    else printf("No hubo raiz");
   }
}
```

4. SISTEMAS DE ECUACIONES SIMULTANEAS LINEALES COMPLEJAS

Análisis

Un sistema de ecuaciones simultáneas lineales de números complejos tiene la siguiente forma general:

$$(a11+b11\ i)\ X1+(a12+b12\ i)X2+(a13+b13\ i)\ X3\ \ldots\ +(a1n+b1n\ i)\ Xn = ac1+bc1\ i$$
$$(a21+b21\ i)\ X1+(a22+b22\ i)X2+(a23+b23\ i)\ X3\ \ldots\ +(a2n+b2n\ i)\ Xn = ac2+bc2\ i$$
$$(a31+b31\ i)\ X1+(a32+b32\ i)X2+(a33+b33\ i)\ X3\ \ldots\ +(a3n+b3n\ i)\ Xn = ac3+bc3\ i$$

.
.
.

$$(an1+bn1\ i)X1+(an2+bn2\ i)X2+(an3+bn3\ i)X3\ \ldots\ +(ann+bnn\ i)\ Xn = acn+bcn\ i$$

En este sistema de ecuaciones las variables incógnitas son X1, X2, X3, ...Xn.
Los coeficientes de las variables complejas son conocidas.

Realizando ciertas operaciones en las ecuaciones del sistema, éste se puede convertir ésta en un sistema que tenga la siguiente forma:

$$(1+0\ i)\ X1 + (0+0\ i)\ X2 +(0+0\ i)\ X3\ \ldots\ +(0+0\ i)\ Xn = a1+b1\ i$$
$$(0+0\ i)\ X1 + (1+0\ i)\ X2 +(0+0\ i)\ X3\ \ldots\ +(0+0\ i)\ Xn = a2+b2\ i$$
$$(0+0\ i)\ X1 + (0+0\ i)\ X2 +(1+0\ i)\ X3\ \ldots\ +(0+0\ i)\ Xn = a3+b3\ i$$

.
.
.

$$(0+0\ i)\ X1 + (0+0\ i)\ X2+(0+0\ i)\ X3\ \ldots\ +(1+0\ i)\ Xn = an+bn\ i$$

Quedando este sistema equivalente, se hace explícita la solución:

$$X1 = a1+b1\ i$$
$$X2 = a2+b2\ i$$
$$X3 = a3+b3\ i$$

.
.
.

$$Xn = an+bn\ i$$

Ejemplo:

$$(\ 8.2+6.3\ i)\ X1 + (\ 8.0+7.3\ i)\ X2 + (-3.5+2.9\ i)\ X3+ (4.3\ -14.5\ i)\ X4 = 5.26+5.0\ i$$
$$(13.1+2.1\ i)\ X1 + (\ 9.4-22.2\ i)\ X2 + (\ 23.4+35\ i)\ X3+ (6.34+8.2\ i)\ X4 = 45.2+2.5\ i$$
$$(15.4+2.8\ i)\ X1 + (45.2+2.6\ i)\ X2 + (\ 3.3+4.3\ \ i)\ X3+ (76.6+5.6\ i)\ X4 = 78.6+8.5\ i$$
$$(45.2+2.1\ i)\ X1 + (42.2+69\ \ i)\ X2 + (\ 84.5+43\ i)\ X3+ (54.1+4.4\ i)\ X4 = 95.0+4.5\ i$$

Dichas operaciones entre ecuaciones consisten en: a- una ecuación reemplazarla por esa ecuación multiplicada (todos sus elementos) o dividida por una constante (un número complejo); y b- una ecuación reemplazarla por la diferencia (o suma) miembro a miembro entre esa ecuación (todos sus elementos) y otra ecuación (todos sus elementos).

Diseño del algoritmo

```
LEER m
LEER Sistema S
ESCRIBIR Sistema S
PARA i DESDE 1 HASTA m
   Uno=S[i][i]
   PARA j DESDE 1 HASTA m+1
      S[i][j]=S[i][j]/uno
   PARA k DESDE 1 HASTA m
   SI k <> i
      Cero=S[k][i]
      PARA j DESDE 1 HASTA m+1
         S[k][j]=S[k][j]-Cero*S[i][j]
ESCRIBIR Solución
```

Programa en lenguaje C para resolver sistemas de ecuaciones lineales simultáneas complejas

```
#include "stdio.h"
#include "complex.h"
void main()
{ complex S[5][6],uno,cero;
  float a,b;
  int i,j,k,m;
  printf("Entre el número de ecuaciones del sistema ---> ");
  scanf("%d",&m);
  printf("\n Leyendo los coeficientes del sistema \n\n");
  for (i=0; i<m; ++i)
  {
     printf("\nLeyendo los coeficientes de la ecuación %d \n",i+1);
     for (j=0; j<m; ++j)
     {
           printf("Entre real del coeficiente de la ecuación %d X%d --->
",i+1,j+1);
          scanf("%f",&a);
           printf("Entre imaginaria de coeficiente de ecuación %d X%d --->
",i+1,j+1);
          scanf("%f",&b);
```

```
            S[i][j]=complex(a,b);
        }
        printf("\nLeyendo coeficientes independientes del la ecuación %d \n",i+1);
       printf("Entre  real del coeficiente independiente de la ecuación %d --->
",i+1);
       scanf("%f",&a);
       printf("Entre imaginaria coeficiente independiente de la ecuación %d --->
",i+1);
       scanf("%f",&b);
       S[i][m]=complex(a,b);
}
printf("\ ESTE ES EL SISTEMA DE ECUACIONES COMPLEJAS \n\n");
 for (i=0; i<m; ++i)
 {
    for (j=0; j<m; ++j)
    {     printf("+(%5.1f %+5.1f i)",S[i][j],imag(S[i][j]));
          printf(" X%d ",j+1);
    }
    printf("=%5.1f %+5.1f i\n",S[i][m],imag(S[i][m]));
 }
 for (i=0; i<m; ++i)
 {  uno=S[i][i];
    for (j=0; j<m+1; ++j)
      S[i][j]/=uno;
    for (k=0; k<m; ++k)
    {  if (i != k)
       {  cero=S[k][i];
            for (j=0; j<m+1; ++j)
               S[k][j]-=cero*S [i][j];
        }
    }
 }
printf("\n\n ESTA ES LA SOLUCION DEL SISTEMA \n\n");
 for (i=0; i<m; ++i)
 {
    printf("X%d = %+7.5f %+7.5f i ",i+1,S[i][m],imag(S[i][m]));
    printf("\n");
 }
}
```

BIBLIOGRAFIA

CHAPRA Steven C. y Raymond P. Canale, Métodos Numéricos para Ingenieros, McGraw Hill, México, 1992.

SCHILDT Herbert, C Guia de Autoenseñanza, Osborne mcGraw Hill, Madrid, 1998.

SCHILDT Herbert, C++ Para Programadores, Osborne mcGraw Hill, Madrid, 1996.

GOTTFRIED Byron, Programación en C, McGraw Hill, Madrid, 1998.