Laboratorio de Simulación

Maynor Ballina

maynor.ballina@ecfm.usac.edu.gt
comentario, mensaje en classroom y meet

SAPERE AUDE

Primer Semestre 2021

Algoritmos de ordenación y búsqueda

Ante cualquier proyecto tenemos la necesidad de ordenar los datos obtenidos, para una mejor presentación o para facilitar los cálculos a realizar.
El ordenamiento de los datos puede ser de forma ascendente o descendente.
Para poder escoger el algoritmo correcto de ordenamiento, dependerá de:

- menor tiempo de ejecución
- menor numero de instrucciones

Estos se dividen en dos:

- Directos
  - burbuja
  - selección
  - inserción
- Indirectos
  - shell
  - ordenación rápida
  - ordenación por mezcla
  - radixsort

Existen dos técnicas de ordenación fundamentales en gestión de datos: ordenación de listas y ordenación de archivos. Los métodos de ordenación se conocen como internos o externos según que los elementos a ordenar estén en la memoria principal o en la memoria externa.

burbuja

En el caso de un arreglo (lista) con n elementos, la ordenación por burbuja requiere hasta n-1 pasadas. Por cada pasada se comparan elementos adyacentes y se intercambian sus valores cuando el primer elemento es mayor que el segundo elemento. Al final de cada pasada, el elemento mayor ha "burbujeado" hasta la cima de la sublista actual. Por ejemplo, después que la pasada 0 está completa, la cola de la lista A[n-1] está ordenada y el frente de la lista permanece desordenado. Las etapas del algoritmo son:

- En la pasada 0 se comparan elementos adyacentes
  - $(A[0], A[1]), (A[1], A[2]), (A[2], A[3]), ...(A[n-2], A[n-1])$
  - Se realizan n-1 comparaciones, por cada pareja $(A[i], A[i+1])$ se intercambian los valores si $A[i+1] < A[i]$.
  - Al final de la pasada, el elemento mayor de la lista está situado en $A[n-1]$.
- En la pasada 1 se realizan las mismas comparaciones e intercambios terminando con el elemento segundo mayor valor en $A[n-2]$.
- El proceso termina con la pasada $n-1$, en la que el elemento más pequeño se almacena en $A[0]$.

burbuja

En este ejemplo gráfico vemos como se realizan las iteraciones y los cambios por cada iteración. A=amarillo, N= naranja Dado a que el numero de elementos N=5 las pasada son N-1=4.

Primera pasada

vector inicial	10	30	1	80	6
A<N no cambia	10	30	1	80	6
A>N cambia	10	30	1	80	6
A<N no cambia	10	1	30	80	6
A>N cambia	10	1	30	80	6
Vector final	10	1	30	6	80

Segunda pasada

vector inicial	10	1	30	6	80
A>N cambia	10	1	30	6	80
A<N no cambia	1	10	30	6	80
A>N cambia	1	10	30	6	80
Vector final	1	10	6	30	80

Tercera pasada

vector inicial	1	10	6	30	80
A<N no cambia	1	10	6	30	80
A>N cambia	1	10	6	30	80
Vector final	1	6	10	30	80

Cuarta pasada (n-1)

vector inicial	1	6	10	30	80
A<N no cambia	1	6	10	30	80
Vector final	1	6	10	30	80

selección

Si el arreglo A tiene n elementos, se trata de ordenar los valores del arreglo de modo que el dato contenido en A[0] sea el valor más pequeño, el valor almacenado en $A[1]$ el siguiente más pequeño, y así hasta $A[n-1]$ que ha de contener el elemento de mayor valor.

El algoritmo de selección se apoya en sucesivas pasadas que intercambian el elemento más pequeño sucesivamente con el primer elemento de la lista, A[0] en la primera pasada. En síntesis, se busca el elemento más pequeño de la lista y se intercambia con A[0] , primer elemento de la lista. $A[0]A[1]A[2]....A[n-1]$

Después de terminar esta primera pasada, el frente de la lista está ordenado y el resto de la lista $A[1], A[2]...A[n-1]$ permanece desordenado.

La siguiente pasada busca en esta lista desordenada y selecciona el elemento más pequeño, que se almacena entonces en la posición A[1].

De este modo los elementos A[0] y A[1] están ordenados y la sublista $A[2], A[3]...A[n-1]$ desordenada; entonces, se selecciona el elemento más pequeño y se intercambia con A[2].

El proceso continúa n-1 pasadas; en ese momento la lista desordenada se reduce a un elemento (el mayor de la lista) y el arreglo completo ha quedado ordenado.

selección

En este ejemplo gráfico vemos como se realizan las iteraciones y los cambios por cada iteración. amarillo=menor valor, naranja=espacio seleccionado, verde=sin cambio. Dado a que el numero de elementos N=5 las pasada son N-1=4.

vector inicial	10	30	1	80	6
primera pasada el menor es V[2]					
Cambia V[0]	10	30	1	80	6
Vector intermedio	1	30	10	80	6
segunda pasada el menor es V[4]					
Cambia V[1]	1	30	10	80	6
Vector intermedio	1	6	10	80	30
tercera pasada el menor es V[2]					
Cambia V[2]	1	6	10	80	30
Vector intermedio	1	6	10	80	30
cuarta pasada (n-1) el menor es V[4]					
Cambia V[3]	1	6	10	80	30
Vector final	1	6	10	30	80

inserción

El algoritmo correspondiente a la ordenación por inserción contempla los siguientes pasos:

1. El primer elemento A[0] se considera ordenado es decir, la lista inicial consta de un elemento.

2. Se inserta A[1] en la posición correcta delante o detrás de A[0] , dependiendo de que sea menor o mayor.

3. Por cada bucle o iteración i (desde i $= 1$ hasta n-1) se explora la sublista $A[i-1]..A[0]$ buscando la posición correcta de inserción; a la vez se mueve hacia la derecha en la sublista una posición todos los elementos mayores que el elemento a insertar A[i] , para dejar vacía esa posición.

4. Insertar el elemento en la posición correcta.

inserción

En este ejemplo gráfico vemos como se realizan las iteraciones y los cambios por cada iteración. amarillo=datos insertados, naranja=valor seleccionado. Dado a que el numero de elementos N=5 las pasada son 5.

	datos						**vector a moverse**				
POSICIONES	v[0]	v[1]	v[2]	v[3]	v[4]		v[0]	v[1]	v[2]	v[3]	v[4]
primera seleccion	10	30	1	80	6	inserta v[0]	10				
segunda seleccion		30	1	80	6	valor>v[0] inserta v[1]	10	30			
tercera seleccion			1	80	6	valor<v[0] corre valores e inserta v[0]	1	10	30		
cuarta seleccion				80	6	valor>v[0], valor>v[1], valor>v[2], inserta v[3]	1	10	30	80	
ultima seleccion					6	valor>v[0], valor<v[1] corre valores inserta v[1]	1	6	10	30	80

Quicksort

El método se basa en dividir los n elementos de la lista a ordenar en dos partes o particiones separadas por un elemento: una partición izquierda, un elemento central denominado pivote o elemento de partición, y una partición derecha.

La partición o división se hace de tal forma que todos los elementos de la primera sublista (partición izquierda) son menores que todos los elementos de la segunda sublista (partición derecha).

Las dos sublistas se ordenan entonces independientemente.

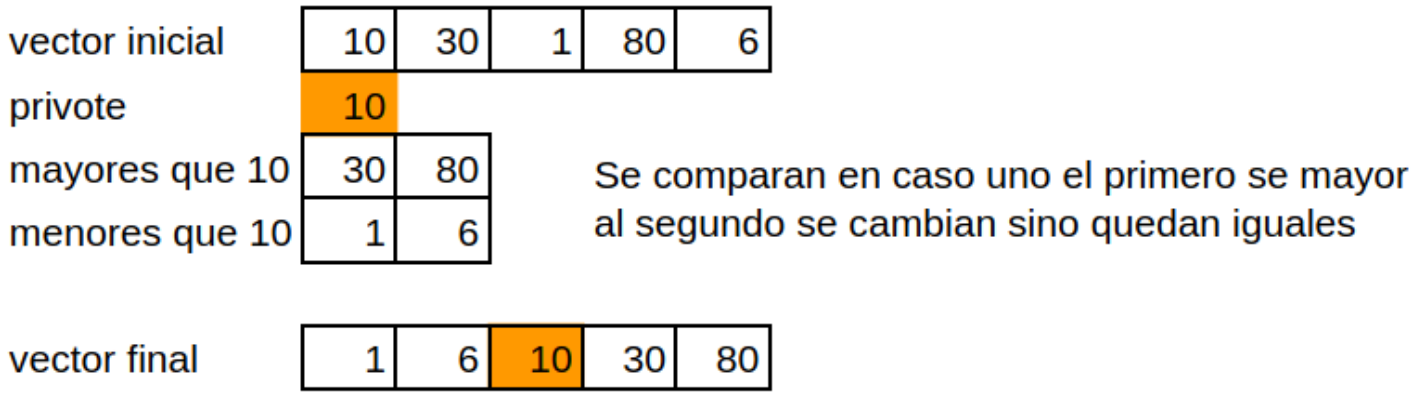
Para dividir la lista en particiones (sublistas) se elige uno de los elementos de la lista y se utiliza como pivote o elemento de partición.

Si se elige una lista cualquiera con los elementos en orden aleatorio, se puede escoger cualquier elemento de la lista como pivote, por ejemplo el primer elemento de la lista. Si la lista tiene algún orden parcial, que se conoce, se puede tomar otra decisión para el pivote.

Idealmente, el pivote se debe elegir de modo que divida la lista exactamente por la mitad, de acuerdo con el tamaño relativo de las claves.

Quicksort

En este ejemplo gráfico vemos como se realizan las iteraciones y los cambios por cada iteración. se elige un valor pivote, dividimos el vector en 3 vectores, pivote, mayores y menores. se compara cada vector y luego se vuelve a unir de forma ordenada.



Estructuras

Desventaja de los vectores: **solo un tipo de datos**.

Una estructura es una colección de uno a mas tipos de elementos denominados miembros, cada uno puede tener un tipo de dato diferente.
Equivale a realizar tablas en excel por lo cual cada miembro (campo) debe de tener un nombre único. Estas se declaran con la sentencia struct y dentro de ella definimos los miembros y el tipo de dato de cada uno. Ejemplos:

```
//estructura simple: un tipo de datos dos elementos independientes.
struct complejo
{
    float p_real, p_imaginaria;
};

//estructura compleja: varios tipo de datos
struct alumnos
{
    char nombre[30];
    unsigned int DPI;
    int carnet;
    float nota;
};
```

Definición de variables de estructuras

definición vs declaración

Declaración: especifica el nombre y el formato, no reserva espacio en memoria.
Definición: crea un espacio en memoria para almacenar los datos de acuerdo al formato.

La definición de cada variable crea un espacio en memoria en donde los datos se almacenan de acuerdo con el formato de la estructura declarada de las siguientes formas:

- Listándolas después de la declaración

```
struct cancion_CD
{
    char titulo[30];
    float duracion;
} cd1, cd2, cd3;
```

- Listando el tipo de estructura y sus variables en cualquier lugar antes de utilizarse.

```
struct cancion_CD cd1, cd2, cd3;
```

```
//tambien se puede inicializar
struct cancion_CD
{
    char titulo[30];
    float duracion;
} cd1={"Los dias de hoy",3.45};
```

Opciones de estructuras

Se puede utilizar el operador *sizeof* en un tipo o una variable para determinar el espacio que la estructura ocupa en memoria.

```
#include <stdio.h>
struct persona
{
    char nombre[30];
    int edad;
    float altura, peso;
} mar;
void main()
{
    printf("Sizeof(persona): %d \n",sizeof(mar));
}
```

Para almacenar o recuperar información de una estructura: utilizar los operadores punto (.) o apuntador ($->$)
Asignación con punto: *nombre_variable.nombre_miembro* = datos;
Asignación con apuntador: para poder utilizar este elemento primero se debe de inicializar la variable de tipo apuntador.
nombre_variable$->$*nombre_miembro* = datos;

La sentencia *typedef* crea un sinónimo de un tipo ya creado.

Estructuras anidadas

Se utiliza una estructura pivote para enlazar otras.

```
struct info
{
    char direccion[30];
    char municipio[20]
    char departamento[20];
};
struct trabajor
{
    char nombre[30];
    struct info info_trabajador;
    float sueldo;
};
struct supervisor
{
    char nombre[30];
    struct info info_supervisor;
    float sueldo;
};
```

Ojo: Las estructuras anidadas requieren de uso múltiple del operador punto: *trabajador.trab1.info.direccion*

struct trabajador trab1 = { "Juan",{10 calle 3-50, mixco, guatemala},2000}

Estructuras

Arreglos: Así como podemos utilizar vectores para almacenar variables, de las misma forma podemos utilizarlos para almacenar estructuras, son muy útiles para declarar varios miembros.

```
struct nomina //se crea un vector de 100 empleados
{
    char nombre[30];
    char departamento[20];
    float salario;
}empleado[100];
```

union: el tipo union es similar a struct, pero al contrario que una definición struct, que asigna memoria suficiente para contener todos los miembros dato, una union puede contener solo un miembro dato en cualquier momento dado y el tamaño de una unión es, por consiguiente, el tamaño de su miembro mayor. La unión permite reducir espacio de memoria en sus programas.

```
union nomina //se crea un elemento tipo union
{
    char c;
    int entero;
    float decimal;
};
```

Ejercicio Estructuras

Debe de realizar un programa el cual solicite al usuario el numero de vértices de un polígono, luego que solicite las coordenadas xy de esos vértices este programa debe de devolver al usuario el área de este polígono.

Ejercicio Estructuras

Debe de realizar un programa el cual solicite al usuario el numero de vértices de un polígono, luego que solicite las coordenadas xy de esos vértices este programa debe de devolver al usuario el área de este polígono.
el área esta definida por:

$$A = \frac{1}{2}(x_{N-1}y_0 - x_0y_{N-1}) + \frac{1}{2}\sum_{i=0}^{N-2}(x_iy_{i+1} - x_{i+1}y_i)$$

Punteros

Punteros tiene la fama de ser uno de los temas mas difíciles cuando se esta aprendiendo cualquier lenguaje de programación. Ya que tiene una curva de aprendizaje lenta y el uso es complejo. PERO el apuntado hace que C sea un lenguaje muy eficiente.

Dirección de memoria: Al crear una variable le damos 3 atributos fundamentales.

```c
int numero = 10;
```

- nombre *numero*
- tipo *int*
- dirección *0x4fffd34*

```c
// para visualizar el valor en consola
printf("%d",numero);
// para visualizar su direccion utilizamos el operador &
printf("%p",&numero)
```

Punteros

El puntero indica donde puedo encontrar algo, este se rige por las siguientes reglas:

- Es una variable
- contiene una dirección que apunta a otra posición de memoria
- La posición almacena los datos donde apunta el apuntador
- Apunta a una variable de memoria

```
#include <stdio.h>
void main()
{
    //variable comun
    int n  = 75;
    //declaro puntero con la
    direccion de n
    int* p = &n;
}
```

Para declarar un apuntador utilizamos siempre el * en la definición de una variable.

```
//diferentes formas de
    declarar un puntero
int* varible;
long* variable;
char* variable;
float* variable;
```

Se deben de inicializar los apuntadores antes de utilizarlos.

```
int i; //defino variable
int *p; //defino puntero
p = &i; //inicializo puntero
*p = 50; //se asigna valor a
    la direcci n de memoria
    (indireccion)
```

OJO: Un error común es tratar de asignar un valor antes de definir la dirección.

Punteros null y void

El apuntador nulo, no apunta a nadie. Se utiliza para decirle a un programa que el apuntador no direcciona a un dato valido. Nunca se utilizan para referenciar un valor.

```
//primero se debe declarar la cabecera nulo
#define NULL 0
//para inicializar apuntador nulo
char *p = NULL;
//importante para evitar errores realizar prueba
if(p==NULL)
{
    puts("Error de asignacion de memoria");
}
//otra forma de declararlo es asignarle un valor 0
int *p = 0;
//apuntador generico
void *p; //se le puede direccionar cualquier tipo de variable
```

Entradas y salidas por archivos

Un flujo es una abstracción que se refiere a una corriente de datos. Entre el origen y el destino debe de existir una conexión. Por defecto c tiene 3 canales abiertos

- stdin teclado
- stdout pantalla
- stderr mensajes de error

Para abrir un archivo necesitamos un buffer intermedio el cual ya esta determinado un puntero tipo FILE.

```
FILE *pf1 *pf2; //inicializamos el punteros
//definimos nombre del archivo
char nm1[] = "tiempo.txt";
char* nm2 = "dias.txt";
//utilizamos la funcion fopen(nombre_archivo,modo)
pf1 = fopen(nm1, "r");
if (pf1==NULL){
    puts("Error al abrir el archivo");
    exit(1);
}
if((pf2 = fopen(nm2,"w"))==NULL){
    puts("Error al escribir en el archivo");
    exit(-1);
}
```

Entradas y salidas por archivos

Otros modos que podemos utilizar son:

Modo	Significado
`"rt"`	Abre para lectura un archivo de texto, es igual que modo `"r"`.
`"wt"`	Abre para crear nuevo archivo de texto (si ya existe se pierden sus datos), es igual que modo `"w"`.
`"a"`	Abre para añadir al final en un archivo de texto, igual que modo `"at"`.
`"r+t"`	Abre archivo de texto ya existente para modificar (leer/escribir).
`"w+t"`	Crea un archivo de texto para escribir/leer (si ya existe se pierden los datos).
`"a+t"`	Abre el archivo de texto para modificar (escribir/leer) al final. Si no existe es como `w+`.

Otro elemento que puede devolver la función fopen es un EOF, todo archivo que trabajemos debe de cerrarse al finalizar su uso.
En caso nosotros queramos vaciar un buffer podemos utilizar la función **fflush**
Para escribir en un archivo utilizamos putc(c, puntero-archivo) o fputc, donde c es el carácter a escribir.
Para leer en un archivo utilizamos getc(puntero-archivo) o fgetc, donde c es el carácter a escribir.
Siempre debemos de cerrar nuestros archivos con la opción **fclose**.

Introducción a Gnuplot

Introducción

Para instalar gnuplot desde nuestra consola utilizamos.

```
$ sudo apt install gnuplot
#inicializamos con el comando gnuplot
$ gnuplot
```

Gnuplot contiene varias funciones por defecto las cuales podemos utilizar para trabajar en caso las queramos utilizar solo la escribimos.

```
> plot sin(x)
```

Este es un listado de funciones que tiene gnuplot el resto lo pueden consultar en el **manual de usuario**.

Math library functions		
Function	Arguments	Returns
abs(x)	any	absolute value of x, $\|x\|$; same type
abs(x)	complex	length of x, $\sqrt{\text{real}(x)^2 + \text{imag}(x)^2}$
acos(x)	any	$\cos^{-1} x$ (inverse cosine)
acosh(x)	any	$\cosh^{-1} x$ (inverse hyperbolic cosine) in radians
airy(x)	any	Airy function Ai(x)
arg(x)	complex	the phase of x
asin(x)	any	$\sin^{-1} x$ (inverse sin)
asinh(x)	any	$\sinh^{-1} x$ (inverse hyperbolic sin) in radians

Modificar gráfica

Los elementos mas útiles en la modificación de su gráfico son los siguientes:

```
#titulo del grafico
> set title "nombre"
#etiquetas de los ejes
> set xlabel "nombre"
> set ylabel "nombre"
#rango de cada eje
> set xrange [inicio:fin]
> set yrange [inicio:fin]
#para que gnuplot determine la escala
> set autorange
#define la posicion del la descripcion
> set ket at x,y
> unset key
#para convertir la escala a logaritmica
> set logscale ("x o Y")

> unset logscale

#para aplicar las modificaciones se debe utilizar la instruccion:
> replot
```

jugando con gnuplot

El mayor trabajo que se realiza en gnuplot es en base a leer y crear archivos:

```
# graficar datos de un archivo
> plot "nombre_archivo"
# para especificar una columna del documento, gnuplot asume que es
# un archivo separado por tabulaciones
> plot "nombre_archivo" using 1:3
# si se utiliza un archivo csv se debe de utilizar primero
> set datafile separator ','
# en el caso de que quiera graficar dos series diferentes de datos
> plot "nombre_archivo" using 1:3, "nombre_archivo_2" using 1:2
# para graficar aplicando formulas matematicas definimos la columna
# a utilizar con $numero_columna
> plot "nombre_archvio" using (sqrt($1**2 + $2**2))
# al guardar la grafica definir la extension del archivo (.png etc)
# con un set terminal para indicar el tipo de archivo que se genera
> set terminal jpeg
> set output "nombre_archivo.jpg"
# tambien se puede utilizar postscript para crear pdf
> set terminal postscript enhanced color
> set output '| ps2pdf - plot.pdf'
# para almacenar un archivo .tex
> set terminal epslatex
> set output "nombre_archvio.tex"
```

tipos de gráfica

Existen diferentes estilos para graficar:

- lines
- points
- linespoints
- dots
- impulses
- yerrorbars
- xerrorbars
- steps
- fsteps
- histeps
- boxes
- boxerrorbars
- boxxyerrorbars
- vector
- financerbars
- candlesticks
- error lines
- xerrorlines
- yerrorlines
- xyerrorlines

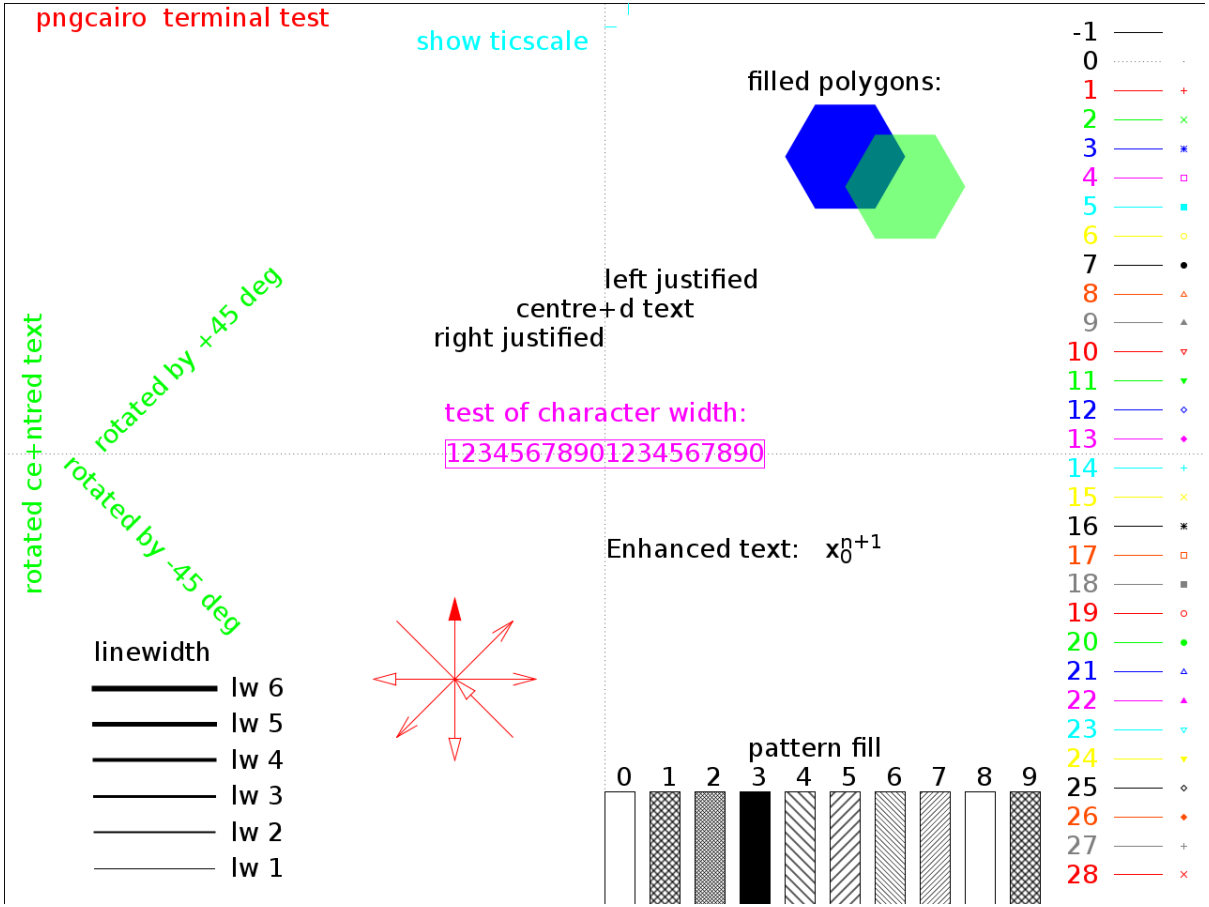
```
# utilizarlos con with despues del elemento a graficar
> plot "nombre_archivo" with lines
```

Luego de definir el estilo de gráfica podemos modificar elementos propios del gráfico, para cada elemento existe una forma rapida de escribirlo.

- linestyle ls
- linetype lt
- linewidth lw
- linecolor lc
- pointtype pt
- pointsize ps
- fill fs
- palette

tipos de gráfica

Se ilustra las diversar modificaciones que se pueden realizar a una grafica.




```
# ejemplo: rango x y, datos, tipo grafico, tipo punto, size punto.
> plot [-15:15][-1:1] sin(x) w p pt 4 ps 2
# datos con barras de error
> plot "nombre_archivo" using 1:2 with errorbars
```

tipos de gráfica

Siendo el caso que necesite automatizar o definir un estilo preciso de gráfico, puede crear un archivo el cual contenga instrucciones de gnuplot para crear sus gráficos.

```
unset label
clear
set terminal epslatex
set output "ej1.tex"
set title "${Prueba de grafico}"
set xlabel "x"
set ylabel "seno de x"
set grid
set style data linespoints
plot sin(x) linewidth 3
```

```
# el archivo anterior se puede ejecutar en consola utilizando
$ gnuplot "nombre_archivo"
```

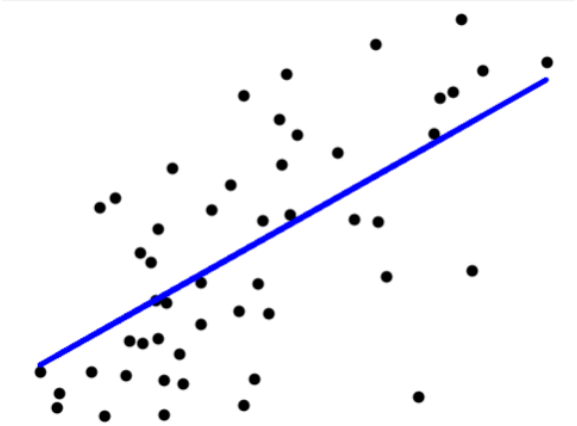
Esto nos creara los archivos que nosotros definimos.

Métodos numéricos

Mínimos cuadrados

El método de mínimos cuadrados nos permite obtener una ecuación recta la cual se ajusta de mejor forma a un grupo de datos obtenidos.

$$y = mx + b$$



Donde los valores m y b se obtienen:

$$m = \frac{n\sum_{k=1}^{n}(x_k y_k) - \sum_{k=1}^{n} x_k * \sum_{k=1}^{n} y_k}{n\sum_{k=1}^{n} x_k^2 - (\sum_{k=1}^{n} x_k)^2}$$

$$b = \frac{\sum_{k=1}^{n} y_k - m\sum_{k=1}^{n} x_k}{n}$$

Mínimos cuadrados

Dado a que toda medida experimental es inútil sin incierta, tomaremos el error de la variable x como ϵ

$$\Delta m = \frac{\sqrt{n}\epsilon}{\sqrt{n\sum_{k=1}^{n} x_k^2 - (\sum_{k=1}^{n} x_k)^2}}$$

$$\Delta b = \frac{\epsilon}{\sqrt{n}}$$

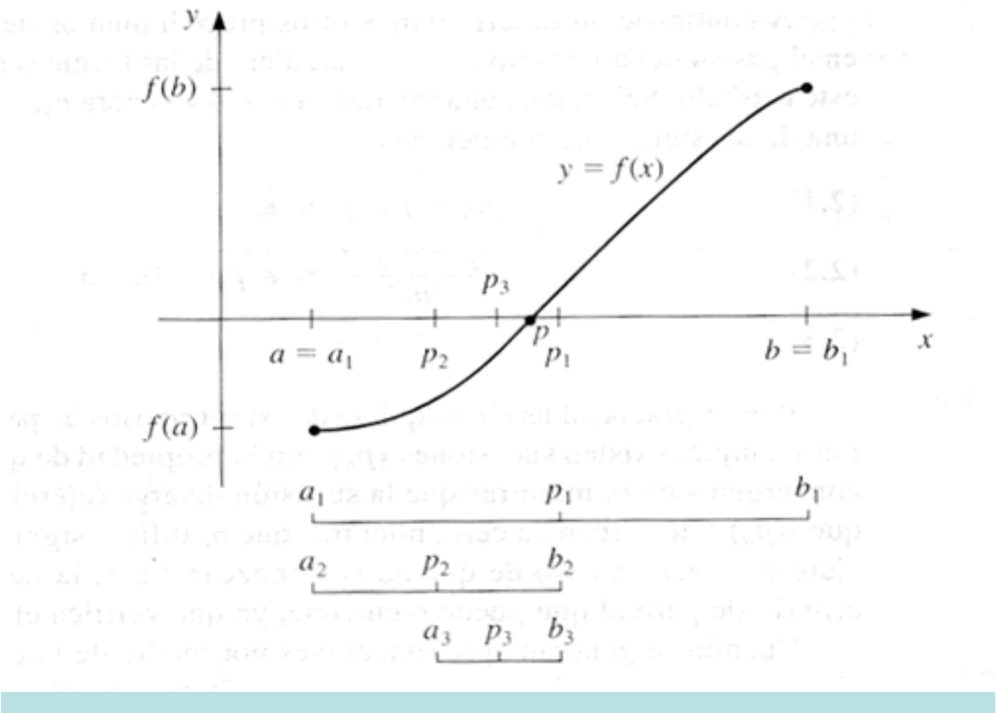
El coeficiente de correlación lo obtenemos con:

$$r = \frac{n\sum_{k=1}^{n}(x_k y_k) - \sum_{k=1}^{n} x_k * \sum_{k=1}^{n} y_k}{\sqrt{(n\sum_{k=1}^{n} x_k^2 - (\sum_{k=1}^{n} x_k)^2) * (n\sum_{k=1}^{n} y_k^2 - (\sum_{k=1}^{n} y_k)^2)}}$$

Recordemos que el coeficiente de correlación es un valor entre 1 y -1.

Método de Bisección

Divide y vencerás, es una forma básica de obtener la raíz de una función $f(x) = 0$
Lo primero que debemos de hacer es tabular y definir donde se encuentra una variación del signo de la función. Esos serán nuestro valores a y b.



La búsqueda viene dada por:

$$p_1 = \frac{a_1 + b_1}{2}$$

Método de Bisección

Al nosotros encontrar estos valores, valuamos $f(x_a)$ y $f(x_p)$ para verificar si:

$$f(x_a) * f(x_p) > 0$$

entonces remplazamos el valor de p por valor de a, de lo contrario si:

$$f(x_a) * f(x_p) < 0$$

entonces remplazamos el valor de p por valor de b, por ultimo si es igual a 0 entonces hemos encontrado nuestra raíz.
El error va estar dado por:

$$\epsilon = \frac{p_n - p_{n-1}}{p_n} * 100\,\%$$

Otra forma de encontrar las raíces es iterar hasta llegar al 1 %

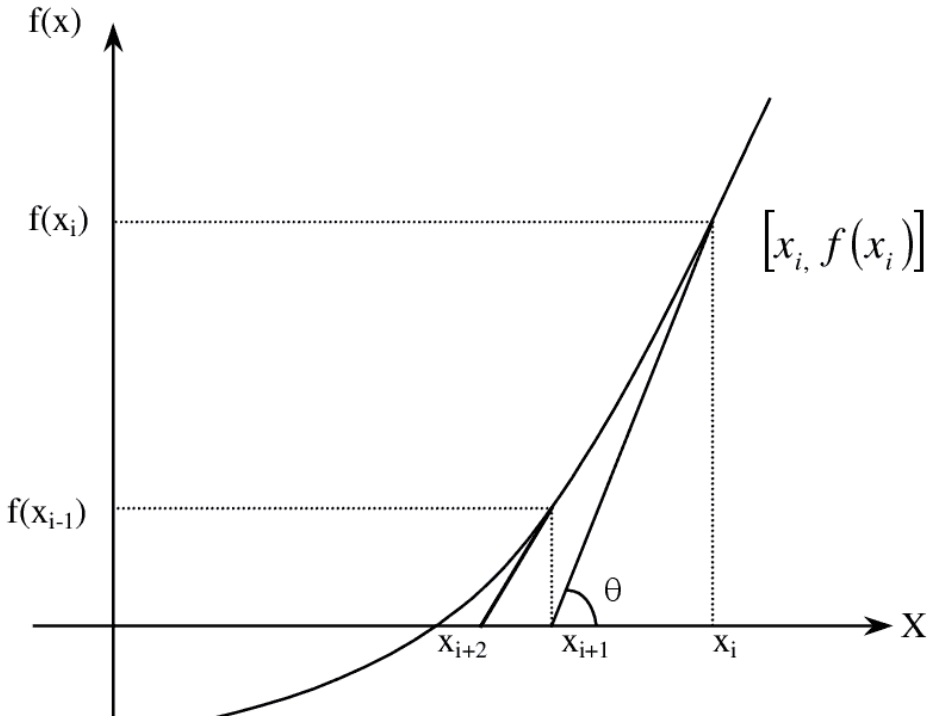
Métodos numéricos parte 2

Newton-Raphson

El método numérico el cual resuelve la búsqueda de raíces $f(x) = 0$, por medio de varias iteraciones acercándose a un valor cercano a la raíz necesitada.

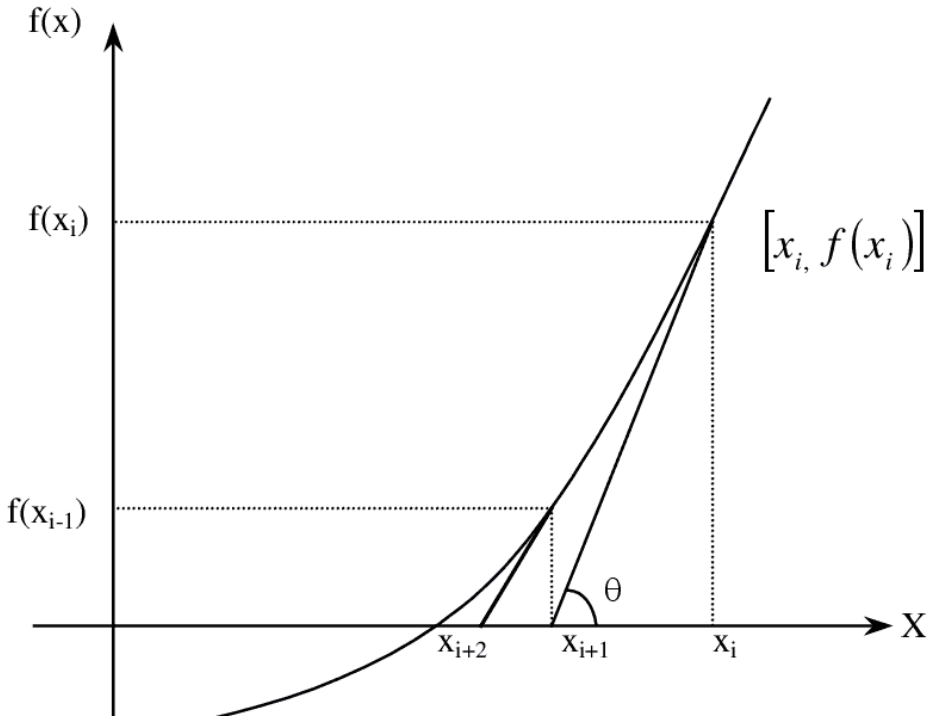
Newton-Raphson

El método numérico el cual resuelve la búsqueda de raíces $f(x) = 0$, por medio de varias iteraciones acercándose a un valor cercano a la raíz necesitada.



Newton-Raphson

El método numérico el cual resuelve la búsqueda de raíces $f(x) = 0$, por medio de varias iteraciones acercándose a un valor cercano a la raíz necesitada.



Se utiliza la primera aproximación y esa la usamos para la segunda y así sucesivamente:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Siempre que $f'(x_i) \neq 0$

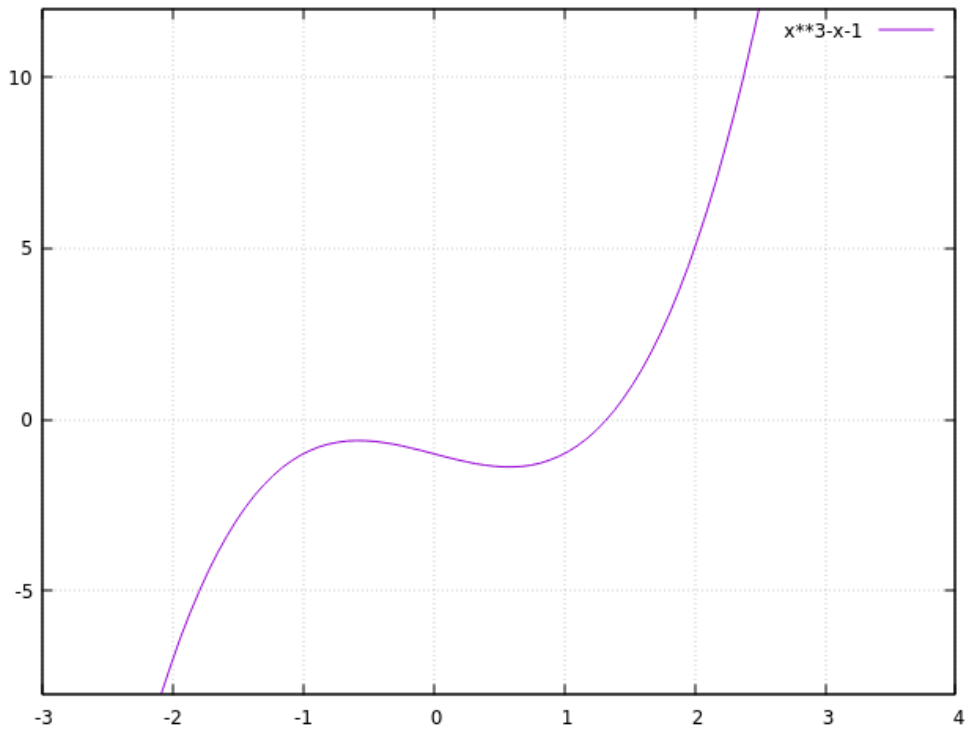
Newton-Raphson

Encontrar una raíz de $f(x) = x^3 - x - 1$

Newton-Raphson

Encontrar una raíz de $f(x) = x^3 - x - 1$
Se analiza la función de forma gráfica.

```
$ gnuplot
> set grid
> plot [-3:4] [-10:10] x^3-x-1
```



Desde el punto de vista gráfico se puede apreciar que los puntos iniciales pueden ser x=1 o x=2.

Newton-Raphson

Al aplicar el método:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Newton-Raphson

Al aplicar el método:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Se obtiene la síguete ecuación

$$x_{i+1} = x_i - \frac{x_i^3 - x_i - 1}{3 * x_i^2 - 1)}$$

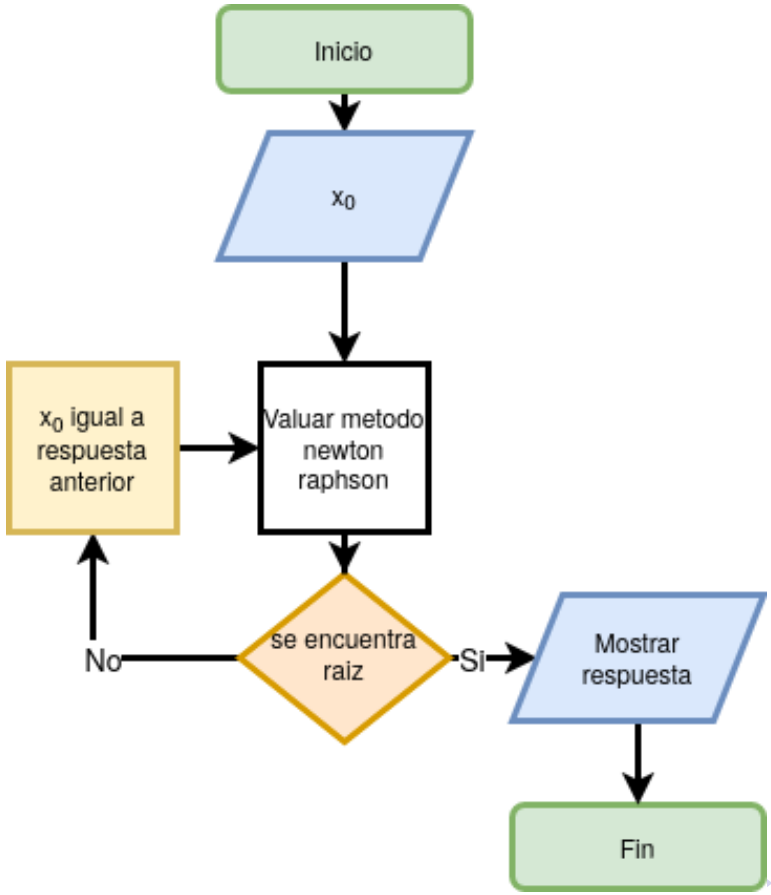
Newton-Raphson

Al aplicar el método:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Se obtiene la síguete ecuación

$$x_{i+1} = x_i - \frac{x_i^3 - x_i - 1}{3 * x_i^2 - 1)}$$



Newton-Raphson

Variables de entrada:

- x_inicial float
- tolerancia float
- iteraciones int

Variables de salida

- xS float (solución del problema)
- "mensaje de error"string

Newton-Raphson

Variables de entrada:

- x_inicial float
- tolerancia float
- iteraciones int

Variables de salida

- xS float (solución del problema)
- "mensaje de error"string

Pseudocódigo del problema anterior

```
"Paso 1" Definir i=1
"Paso 2" Leer valor x_inicial, tolerancia, iteraciones
"Paso 3" Mientras i<= iteraciones pasos del 4 al 7
    "Paso 4" tomar x=x_inicial - f(x_inicial)/f'(x_inicial)
             (calcular xi)
    "Paso 5" si |x-x_inicial|<tolerancia entonces
                xS = x
                Salida(xS)
                Parar
    "Paso 6" Aumentar cuenta i = i + 1
    "Paso 7" subtituir x_inicial = x
"Paso 8" Salida("El metodo fracaso despues de n iteraciones")
         Parar
```

Newton-Raphson código

```
//prototipos de funciones
float f(float x);
float df(float x);
void NewtonRaphson(float x0, float tol, int maxiter, int *actiter,
    float *sol);

//ecuacion
float f(float x)
{
    float res =0;
    res = x*x*x-x-1;
    return res;
}

//primera derivada
float df(float x)
{
    float res =0;
    res=3*(x*x)-1;
    return res;
}
```

Newton-Raphson código

```
//solucion problema
void NewtonRaphson(float x0, float tol, int maxiter, int *actiter,
    float *sol)
{
    //definir variables locales
    float xant, x, dif;
    int i=1;
    xant=x0;
    //solucionar primera iteracion
    x=xant-f(xant)/df(xant);
    //verificar si la resta es menor a 0
    dif = x-xant;
    (dif<0)?dif=-dif:dif;
    //realizar todas las iteraciones
    while (dif<tol && i<maxiter)
    {
        xant=x;
        x=xant-f(xant)/df(xant);
        i++;
    }
    //apuntar a la memoria para la respuesta
    *sol=x;
    *actiter = i;}
```

Newton-Raphson código

```
void main (void)
{
    //definir variables
    float x_inicial, tolerancia, xS;
    int iteraciones, Aiteracion;
    //obtener datos
    touch("Ingrese el valor aproximado de x");
    scanf("%f",&x_inicial);
    touch("Ingrese el valor de tolerancia");
    scanf("%f",&tolerancia);
    touch("Ingrese el valor maximo de iteraciones");
    scanf("%d",&iteraciones);
    //solucionar el problema
    NewtonRaphson(x_inicial, tolerancia, iteraciones, &Aiteracion,
    &xS);
    if (Aiteracion == iteraciones)
        printf("No hay solucion despues de %d iteraciones\n",
    iteraciones);
    else
    {
        printf("Luego de %d iteraciones la solucion es %.4f\n",
    Aiteracion,xS);
    }
}
```

Ecuaciones Diferenciales Ordinarias

Ecuaciones Diferenciales Ordinarias

El método numérico de Euler nos permite encontrar una solución aproximada a ecuaciones diferenciales.

$$\frac{dy}{dt} = f(t, y) \qquad a \leq y \leq b \qquad y(a) = \alpha$$

El método define una aproximación numérica dentro de un intervalo [a,b].
El algoritmo nos define que se debe de iterar una regla de recurrencia.

$$y(t_{k+1}) = y(t_k) + hf(y(t_k), t_k)$$

$$t_{k+1} = t_k + h$$

El algoritmo se detiene cuando $t_k \geq t_final$

Algoritmo método Euler

Variables de entrada:

- extremos a,b
- N numero iteraciones
- condición inicial α

Variables de salida

- $y(t)$ aproximación

Pseudocódigo

```
"Paso 1" Tomar h= (b-a)/N
                t=a
                y(t) = alfa
                Salida(t,y(t))
"Paso 2" Para i=1, 2, 3, ... N pasos 3,4
    "Paso 3" y(t)=y(t)+hf(t,y(t)) calcule y(t_i)
             t = a + ih
    "Paso 4" Salida(t,y(t))
    "Paso 5" Parar
```

Ejemplo/Ejercicio

Tiene un tanque de una área de $1m^2$ y una altura de 10 metros, se le realiza un agujero en la parte inferior. Teniendo un caudal de salida dado a la gravedad de $Q_e = K\sqrt{h}$ donde K es una constante de proporcionalidad dependiente del liquido, en este caso $K = 0{,}1$ si la ecuación de describe el proceso de descarga del agua es:

$$A\frac{dh}{dt} = Q_e - Q_s$$

Si el paso de integración dado es **h**=0.1. Encuentre la altura del liquido pasado 3 segundos, el tiempo total en el que se vacía el tanque, realice una gráfica del proceso de descarga

Crear el diagrama de flujo

Ejemplo/Ejercicio

Variables de entrada

- Ninguna

Variables de salida

- altura en t=3
- tiempo en h=0
- tiempo y altura en archivo “alturas”

Ejemplo/Ejercicio

Variables de entrada

- Ninguna

Variables de salida

- altura en t=3
- tiempo en h=0
- tiempo y altura en archivo “alturas”

Variables internas

- A=1 área
- K=0.1 constante de descarga
- h=10 altura tanque
- delta=0.1 paso de integración para no confundir con altura
- t=0 tiempo inicial

Ejemplo/Ejercicio

Dado a que el caudal de entrada es nulo obtenemos:

$$\frac{dh}{dt} = \frac{0 - K\sqrt{h}}{A}$$

Aplicando el método de Euler obtenemos:

$$h(t_{k+1}) = h(t_k) + deltaf(h(t_k), t_k)$$

$$h(t_{k+1}) = h(t_k) + delta * -\frac{K\sqrt{h}}{A}$$

El tiempo esta dado por $t_{k+1} = t_k + delta$, escriba el código para $f(h(t_k), t_k)$

```
//le llamare df a la derivada de la funcion
float df(float A, float K, float h)
{
    float res;
    res = - (K*sqrt(h))/A
    return res;
}
```

Metodo de Taylor de orden superior

Parecido al método de Euler con la deferencia que ahora minimizamos el error por medio del aumento de polinomios.

$$\frac{dy}{dt} = f(t, y) \qquad a \leq y \leq b \qquad y(a) = \alpha$$

El método define una aproximación numérica dentro de un intervalo [a,b]

$$h = \frac{b-a}{N} \qquad t_i = a + i * h \qquad i = 0, 1, 2...N \qquad \omega_0 = \alpha$$

Para facilitar la lectura denominaremos $\omega = y(t_k)$, el método de Euler es el método de Taylor de orden uno. N determina los polinomios.

$$\omega_{i+1} = \omega_i + hT^n(t_i, \omega_i)$$

$$T^n(t_i, \omega_i) = f(t_i, \omega_i) + \frac{h}{2}f'(t_i, \omega_i) + ... + \frac{h^{n-1}}{n!}f^{(n-1)}(t_i, \omega_i)$$

Algoritmo método Taylor

Variables de entrada:

- extremos a,b
- N numero iteraciones
- condición inicial α

Variables de salida

- $y(t)$ aproximación

Pseudocódigo

```
"Paso 1" Tomar h= (b-a)/N
                t=a
                y(t) = alfa
                Salida(t,y(t))
"Paso 2" Para i=1, 2, 3, ... N pasos 3,4
    "Paso 3" y(t)=y(t)+h T^n (t,y(t)) calcule y(t_i)
             t = a + ih
    "Paso 4" Salida(t,y(t))
    "Paso 5" Parar
```

Método de Runge-Kutta

Método de Taylor tiene error local de truncamiento de orden alto, necesita el calculo y valuación de las derivadas. Costo computacional alto. El método Runge-Kutta también tiene error local de truncamiento de orden alto, pero permite prescindir del calculo y la devolución de las derivadas.
Definiremos un termino llamado error de convergencia $O(h)$.
Método de Runge-Kutta de primer orden es el método de Euler con $O(h)$.
Método de Runge-Kutta de segundo orden, punto medio con $O(h^2)$

$$T^2(t,y) = f(t,y) + \frac{h}{2}f'(t,y)$$

$$\omega_{i+1} = \omega_i + hf(t_i + \frac{h}{2}, \omega_i + \frac{h}{2}f(t_i,\omega_i))$$

$$\omega_0 = \alpha$$

$$K_1 = hf(t_i,\omega_i)$$

$$K_2 = hf(t_i + \frac{h}{2}, \omega_i + \frac{K_1}{2})$$

$$\omega_{i+1} \approx \omega_i + K_2$$

Comparar

Método de Euler Modificado.

$$\omega_0 = \alpha$$

$$\omega_{i+1} = \omega_i + \frac{h}{2}[f(t_i, \omega_i) + f(t_{i+1}, \omega_i + f(t_i, \omega_i))]$$

Método de Heun

$$\omega_0 = \alpha$$

$$\omega_{i+1} = \omega_i + \frac{h}{4}[f(t_i, \omega_i) + 3f(t_i + \frac{2h}{3}, \omega_i + \frac{2}{3}hf(t_i, \omega_i))]$$

Método de Runge-Kutta

Método de Runge-Kutta de cuarto orden, punto medio con $O(h^4)$

$$\omega_0 = \alpha$$

$$K_1 = hf(t_i, \omega_i)$$

$$K_2 = hf(t_i + \frac{h}{2}, \omega_i + \frac{K_1}{2})$$

$$K_3 = hf(t_i + \frac{h}{2}, \omega_i + \frac{K_2}{2})$$

$$K_4 = hf(t_{i+1}, \omega_i + K_3)$$

$$\omega_{i+1} = \omega_1 + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

Ejercicio

Realiza un programa que resuelva $\frac{dy}{dt} = y - t^2 + 1$ en el rango $[0 : 2]$ donde $y(0) = 0{,}5$, utilizando los métodos de punto medio, Euler modificado y Heun. Realice una tabla que compare cada método, recuerde para todos $\omega_0 = \alpha$.

Punto medio

$$\omega_{i+1} = \omega_i + hf(t_i + \frac{h}{2}, \omega_i + \frac{h}{2} f(t_i, \omega_i))$$

$$K_1 = hf(t_i, \omega_i)$$

$$K_2 = hf(t_i + \frac{h}{2}, \omega_i + \frac{K_1}{2})$$

$$\omega_{i+1} \approx \omega_i + K_2$$

Método de Euler Modificado.

$$\omega_{i+1} = \omega_i + \frac{h}{2}[f(t_i, \omega_i) + f(t_{i+1}, \omega_i + f(t_i, \omega_i))]$$

Método de Heun

$$\omega_{i+1} = \omega_i + \frac{h}{4}[f(t_i, \omega_i) + 3f(t_i + \frac{2h}{3}, \omega_i + \frac{2}{3} hf(t_i, \omega_i))]$$

POO

POO

Uno de los paradigmas de programación mas importantes es la programación orientada a objetos.

Para entender las características de la POO, necesitamos unos conceptos básicos.

El modelado de un elemento en el mundo real esta dado por:

- **Atributos** = datos
- Comportamiento = funciones.

Dado a que la unión de estos elementos esta plasmada en la vida real, esto es un **objeto**.

En la POO el programa no se divide en tareas, se divide en modelos de objetos físicos o simulados.

Los objetos se pueden agrupar por categorías, al conjunto de objetos de una misma categoría se le denomina una **Clase**.

POO

Pasos para solucionar problemas utilizando POO:

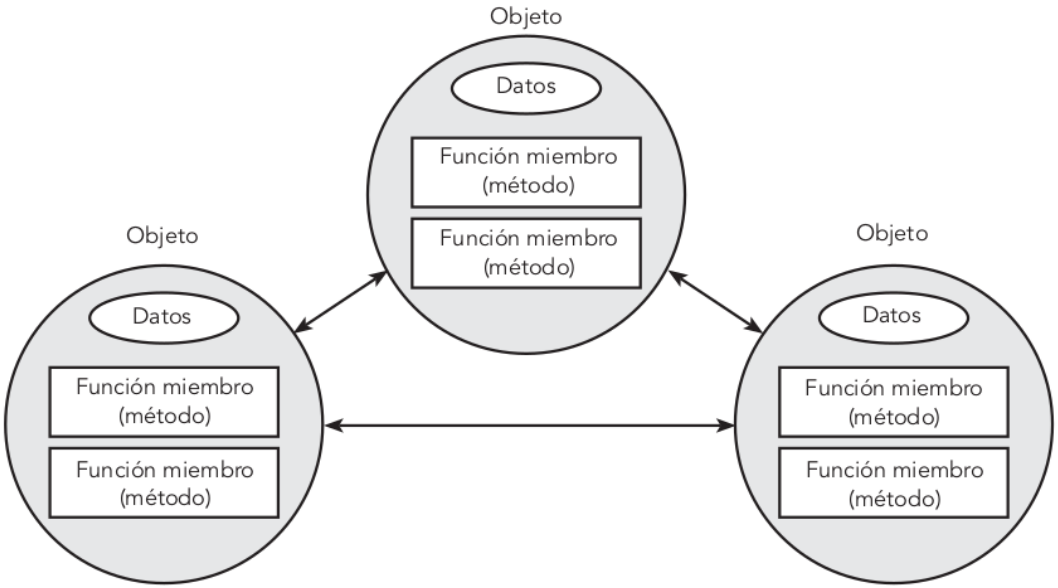
- Identificación de los objetos del problema
- agrupamiento por clases
- identificar datos y operaciones por clase
- identificar relación entre clases

Una clase es la implementación de un tipo abstracto de datos, describe los atributos y operaciones.

Una clase es un tipo de dato, del cual se pueden crear variables de este tipo denominadas **instancias**.

A las operaciones definidas para los objetos las llamamos **métodos**, los cuales activan por medio de *mensajes*.

En pocas palabras: **un objeto es una instancia de una clase**.



	Mundo Real	**En OOP**
Clase Generalización de características (atributos y comportamientos)	**Perro** Raza, Color, Edad, Corre,	**Clase** Define datos y métodos
Objeto Instancia de una clase distinguible por sus características específicas	**Tino** Pastor Alemán Marrón 7 meses Veloz	**Objetos** Ocupa espacio, se crea y se destruye

POO

Todo objeto tiene:
Estado: conjunto de valores de los atributos en un instante de tiempo.
Comportamiento: conjunto de operaciones que puede realizar un objeto.
Identidad: diferencia los objetos no importando los estados de estos.

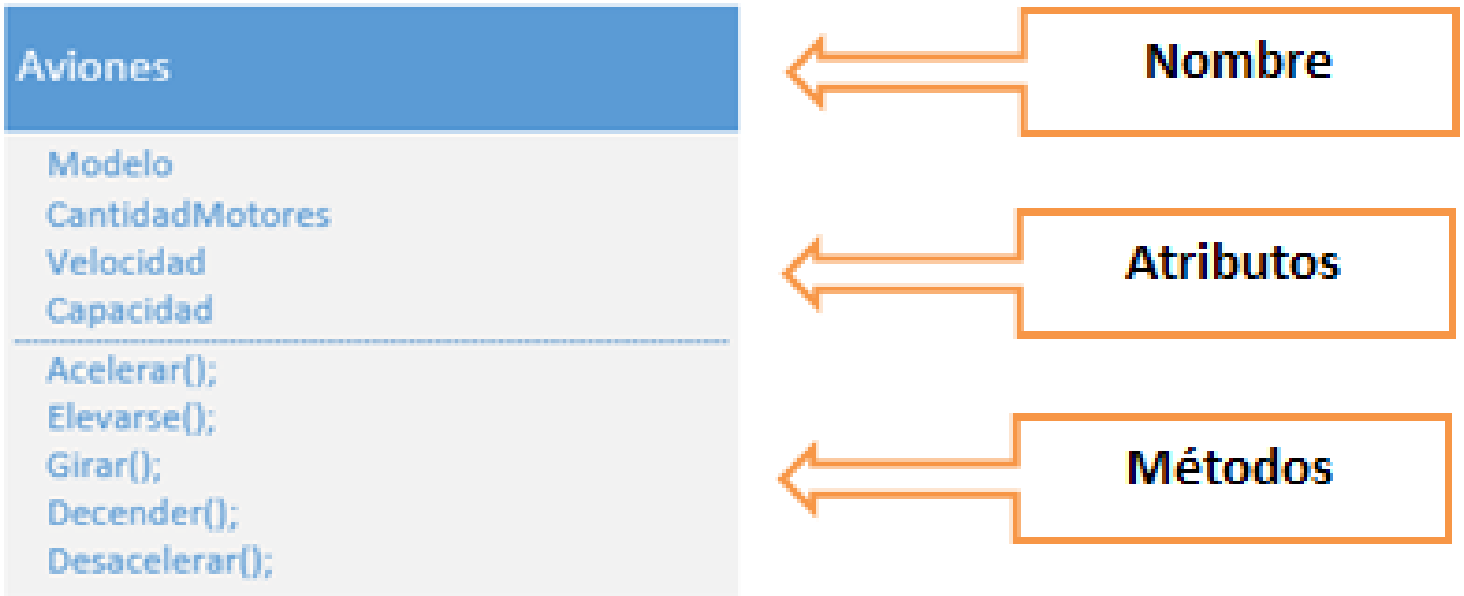
Propiedades fundamentales de este paradigma de programación son:

- **Abstracción:** es una forma de reducir la complejidad de un sistema en características o procesos simples.
- **Encapsulamiento y ocultación de datos:** agrupación de datos y operaciones bajo la misma entidad, las cuales se ocultan de otros objetos.
- **Herencia:** relación de generalización, los objetos tienen que organizarse en jerarquía.
- **Reusabilidad:** las clases la pueden utilizar otros programadores. *Como las bibliotecas*
- **Polimorfismo:** operaciones con el mismo nombre en diferentes clases y diferentes acciones.

Lenguaje unificado de modelado

OBJETO = Estado + Comportamiento + Identidad

La representación gráfica en UML es necesaria para la POO.



Pseudocódigo

```
#formato de clase
clase NombreClase
#elementos de clase
    #Declaracion de atributos constantes o variables
    #[privado| publico|protegido] <tipo>:<nombre>=<valor>
    #pertenece a la clase
    #[estatico][privado| publico|protegido] <tipo>:<nombre>=<valor>
    const
        privado real: PI = 3.1416
    var
        estatico publico real: precio
    #Declaracion de metodos
    constructor NombreClase([lista_de_parametros])
        #declaracion de variables locales
    fin_constructor
    #abstracto para crear clases hermanas y tener metodos
    diferentes
    #[estatico][abstracto][privado| publico|protegido] <
    tipo_retorno> funcion <nombre_funcion>([lista_de_parametros])
    publico entero funcion devolverx()
        inicio
        devolver (x)
    fin_funcion
```

Pseudocódigo

```
    #[estatico][abstracto][privado| publico|protegido]
    procedimiento <nombre_procedimiento>([lista_de_parametros])
    publico procedimiento fijarx(E entero: cx)
        inicio
        x <- cx
    fin_procedimiento
    destructor NombreClase()
        #declaracion de variables locales
    fin_constructor
fin_clase
```

Nota: método con tipo de retorno que no sea void debe de utilizar **return**

Comportamiento de cada tipo de encapsulamiento.

Tipo de miembro	miembro misma clase	amiga	miembro clase derivada	función no miembro
Privado	x	x		
Protegido	x	x	x	
Publico	x	x	x	x

Ejemplo Pseudocódigo

Dimensiones de varios libros, se crea la clase libro y objetos pertenecientes a esta.

```
clase Libro
    var
        real: ancho
        real: alto
        real: profundidad
    constructor Libro (real:a,b,c)
        inicio
            ancho <- a
            alto <- b
            profundidad <- c
    fin_constructor
fin_clase
```

Ya declarada la clase se puede crear objetos.

```
Libro analisisNumerico                  # declaracion del objeto
analisisNumerico = nuevo Libro()        # creacion del objeto
```

Si usar un atributo o un método.

```
analisisNumerico.ancho = 10                 # atributo
analisisNumerico.nostrarDimensiones()       # metodo
```

Ejemplo

Se le solicita un programa el cual le permita ingresar diferentes fechas y mostrarlas en pantalla utilizando los conceptos de programación orientada a objetos.

Ejemplo

Se le solicita un programa el cual le permita ingresar diferentes fechas y mostrarlas en pantalla utilizando los conceptos de programación orientada a objetos.

Atributos:

Métodos:

Ejemplo

Se le solicita un programa el cual le permita ingresar diferentes fechas y mostrarlas en pantalla utilizando los conceptos de programación orientada a objetos.

Atributos:
elementos de fecha

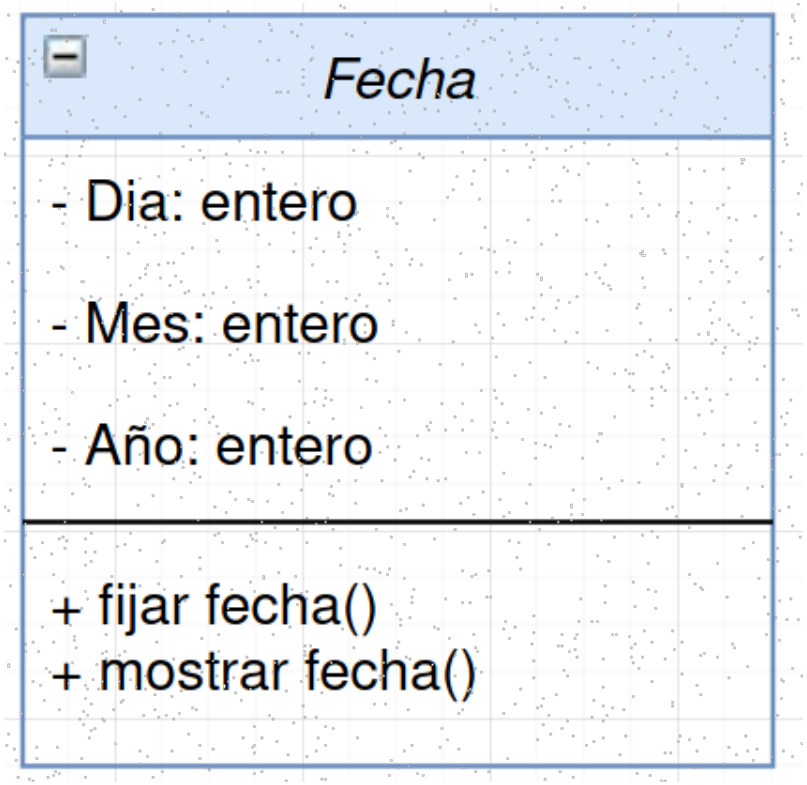
Métodos:
obtener e imprimir la información

Ejemplo

Se le solicita un programa el cual le permita ingresar diferentes fechas y mostrarlas en pantalla utilizando los conceptos de programación orientada a objetos.

Atributos:
elementos de fecha

Métodos:
obtener e imprimir la información



Ejemplo

Pseudocódigo basado en el diagrama UML.

```
clase Fecha
    var
        privado entero: dia, mes, anyo
    procedimiento fijarFecha(E entero: d, m, a)
        inicio
            dia <- d
            mes <- m
            anyo <- a
    fin_procedimiento
    procedimiento mostrarFecha()
        inicio
            escribir(dia,'/',mes,'/',anyo)
    fin_procedimiento
fin_clase
```

Para identificar el tipo de archivo se utiliza la extensión .cpp, creando el ejecutable con el compilador g++ de igual forma que gcc.

```
$ g++ ejemplo.cpp -o ejemplo
#ejecutar el programa
$ ./ejemplo
```

C++

C++ y C son lenguajes de programación similares pero con claras diferencias.

Todo programa en el lenguaje C++, se necesita como cabecera la biblioteca estándar de entrada y salida **iostream**.
5 flujos de datos: cin, cout, cerr, clog, endl.

El canal de entrada **cin** utiliza el operador >> para leer la entrada de la variable.

El canal de salida **cout** utiliza el operador << para encadenar las salidas.

El canal estándar de error **cerr** genera mensajes a usuarios del programa.

El canal estándar **clog** genera información relativa a la ejecución del programa.

El manipulador **endl** ¸comienzo de linea" hace que el cursor se mueva a la siguiente linea.

En este lenguaje de programación existe el tipo de dato **string**.
#include < string >

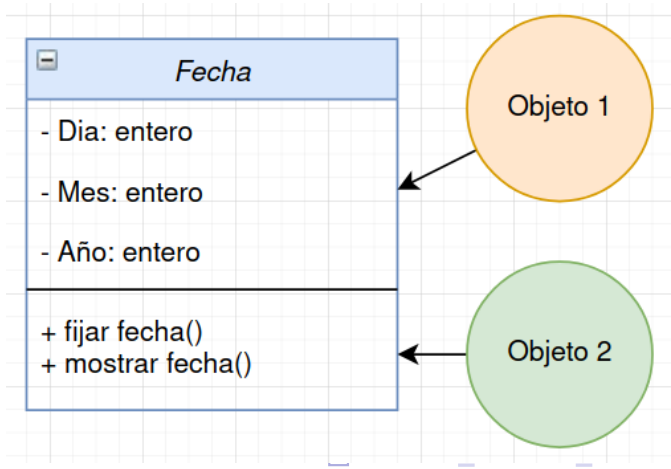
Ejemplo de Pseudocódigo a C++

```
#incluye la libreria iostream.h
#include <iostream>
//Directivas para el uso de elementos de entrada y salida
using namespace std;
//Declaracion de clase con datos y prototipos de funciones
class Fecha
{
    //declaramos atributos
    int dia, mes, anyo;
    //prototipo de metodos
    public:
        void fijarFecha(int,int,int);
        void mostrarFecha();
};
//Para definir la funcion fuera de la clase es necesario el
    operador ::
void Fecha::fijarFecha(int d, int m, int a)
{
    //establece valor de cada atributo
    dia = d;
    mes = m;
    anyo = a;
}
```

Ejemplo de Pseudocódigo a C++

```
void Fecha::mostrarFecha()
{
    cout << dia <<"/"<< mes <<"/"<< anyo;
}
int main() //Luego se utiliza la clase en el programa
{
    Fecha fHoy;
    fHoy.fijarFecha(5,4,2020);
    cout << "La fecha es ";
    fHoy.mostrarFecha();
    cout << endl;  //tambien puede utilizar "\n"
    return 0;
}
```

```
geotoj@geotoj:~/Docume
La fecha es 5/4/2020
geotoj@geotoj:~/Docume
```



Aplicación POO

Ejemplo

Se dispara un poryectil de masa 3kg por medio de un cañon a un ángulo de 30 grados. Se busca acertar a un blanco 4 metros de altura y 10 metros del cañon y 50 cm de diametro. Averiguar la velocidad a la que tiene que salir la bala del cañon para que de en el blanco.

Con tiro parabólico se establece:

$$x = x_0 + v_0 \cos(\theta) t$$

$$y = y_0 + v_0 \sin(\theta) t + \frac{1}{2} g t^2$$