Taxonomía de las estructuras de datos

- Apuntadores
- Vectores
- Matrices
- Cubos
- Listas encadenadas
- Listas doblemente encadenadas
- Arboles
- Montículos (Heaps)
- Grafos

Apuntadores

- Es una variable que indica la localización (dirección) de otra variable.
- Un puntero puede considerarse como una referencia a otra variable.
- Se pueden crear cuando se necesiten y liberarlas cuando se desocupen.
- Se usan para crear estructuras de datos dinámicas que se expandan o se contraigan según se agreguen o eliminen elementos.

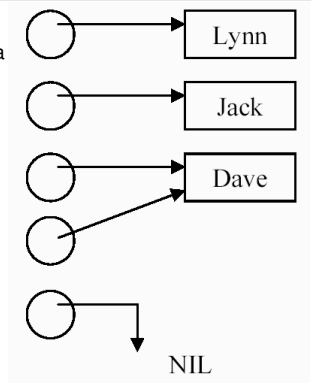
2

Apuntadores

Apuntadores a otras variables

Dos apuntadores pueden "direccionar" al mismo elemento

Un apuntador puede ser "NULO" (referencia a nada)



3

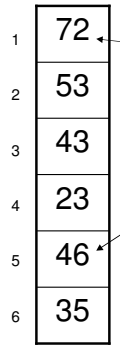
Arreglos

Un arreglo es un conjunto finito de posiciones de memoria consecutivas que tienen el mismo nombre y el mismo tipo de dato.

- Los arreglos de una dimensión (Unidimensionales) se llaman vectores.
- Los arreglos de dos dimensiones (bidimensionales) se llaman matrices.

4

Arreglos Unidimensionales (Vectores)



A[1] = 72

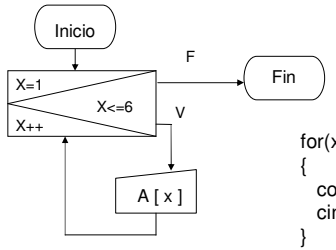
A[5]= 46

Vector “A”

Los elementos de un vector se referencían por el nombre del vector seguido de corchetes o parentesis dentro de los cuales se indica la posición (indice) a la cual hacen referencia.

Arreglos Unidimensionales (vectores)

- Para recorrer, capturar o procesar secuencialmente todos los elementos de un vector, se puede utilizar un ciclo sencillo:




```
for(x=1; x<=6; x++)
{
   cout << “Dame el elemento” << x ;
   cin >> A[x];
}
```

6

Arreglos Bidimensionales (Matrices)

- Una matriz tiene renglones y columnas.
- Sus elementos se referencían por el nombre de la matriz seguido de corchetes o parentesis indicando renglon y columna en donde se encuentra ese elemento.

Matriz “M”

	1	2	3
1	5	1	9
2	8	6	4
3	3	7	2

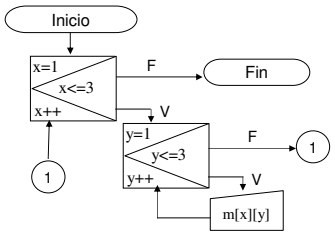
M[3][2] = 7

M[2][3] = 4

7

Arreglos Bidimensionales (Matrices)

- Para recorrer, capturar o procesar secuencialmente todos los elementos de una matriz, se pueden utilizar dos ciclos anidados, uno para los renglones y otro para las columnas:

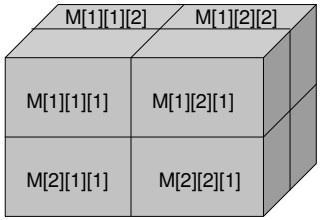



```
for(x=1; x<=3; x++)
{
   for (y=1; y<=3; y++)
   {
      cout << “Dame el elemento”<< x << y;
      cin >> m[x][y];
   }
}
```

8

Arreglos Tridimensionales (Cubos)

- Poseen Renglones, Columnas y Profundidad.
- Requieren de 3 subíndices para localizar sus elementos.
- Pueden usarse 3 ciclos anidados para manipularlos.

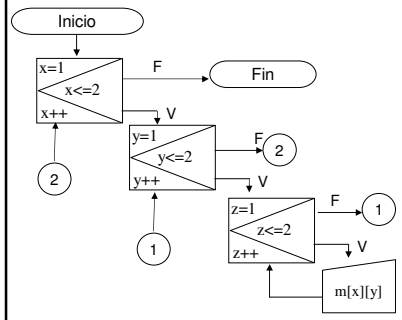


Matriz "M"

9

Arreglos Tridimensionales (Cubos)

- Para recorrer, capturar o procesar secuencialmente todos los elementos de una matriz, se pueden utilizar dos ciclos anidados, uno para los renglones y otro para las columnas:




```
for(x=1; x<=2; x++)
{
  for (y=1; y<=2; y++)
  {
    for (z=1; z<=2; z++)
    {
      cout << "Dame el elemento"<< x << y <<  z;
      cin >> m[x][y][z];
    }
  }
}
```

10

Listas Enlazadas

Una *LISTA* es una colección de elementos llamados *"nodos".* El orden entre los nodos se establece por medio de punteros, es decir, direcciones o referencias a otros nodos.

Un NODO consta de dos campos:

- DATO (Elemento en sí)
- DIRECCION (Del siguiente elemento)

11

El campo "Dirección"

- Relaciona los elementos y establece su secuencia.
- Permite que no sea necesario almacenar físicamente a los nodos en espacios contiguos, ya que basta con seguir el enlace indicado para formar la lista completa de manera ordenada.

12

Ejemplo:

LISTA

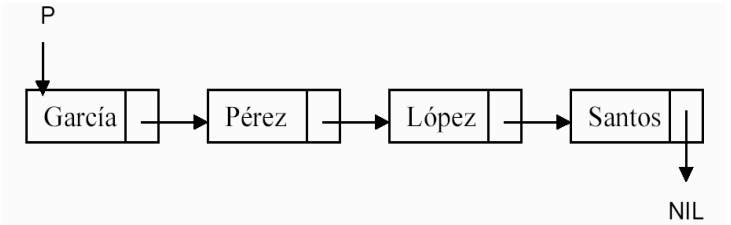
	Dato	Sig
1	CASA	4
2	ZAPATO	0
3	ARBOL	1
4	GATO	2

APEX → (Inicio de la Lista)

Al recorrer la lista de acuerdo a los apuntadores se obtiene la siguiente secuencia de elementos:
ARBOL, CASA, GATO, ZAPATO

13

Otro ejemplo:



Al recorrer la lista se tiene:
García, Pérez, López, Santos

14

Ejercicio: Recorrido de una lista enlazada

En la siguiente lista,

APEX = 4

Determinar el orden de los elementos

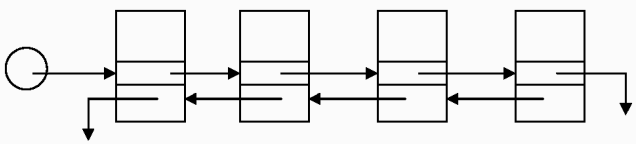
Recorrido:
- •Arco
- •Bote
- •Lodo
- •Mano
- •Niño
- •Oso
- •Rosa
- •Sal
- •Taller
- •Uva

	Dato	Sig
1	Rosa	3
2	Lodo	8
3	Sal	5
4	Arco	7
5	Taller	9
6	Oso	1
7	Bote	2
8	Mano	10
9	Uva	0
10	Niño	6

15

Listas Doblemente enlazadas

- Cada nodo tiene dos apuntadores:
- Uno para el siguiente nodo y otro para el nodo anterior.
- De esta manera es posible moverse en cualquier dirección.



16

PILAS

- Estructura en la que los elementos entran y salen por el mismo extremo utilizando la filosofía LIFO

Last	Ultimo en
Input	Entrar,
First	Primero
Output	en Salir

17

Operaciones posibles en PILAS

- PUSH.- Consiste en introducir un elemento a la pila.
- POP.- Consiste en extraer un elemento de la pila.

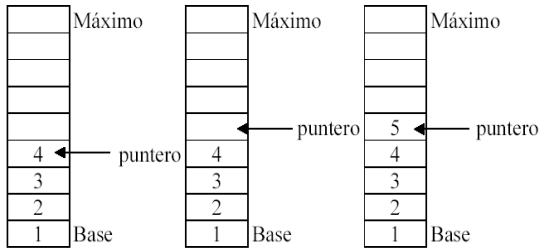
18

PUSH *elemento*

- Si la pila está llena, NO puede realizarse la operación PUSH y se produce un error de OVERFLOW (Desbordamiento de pila).
- Si aún hay espacio en la pila, el apuntador se incrementa en una posicion y se guarda el elemento en esa posición.

19

Ejemplo: PUSH "5"



Pila Antes del PUSH

Pila después del PUSH

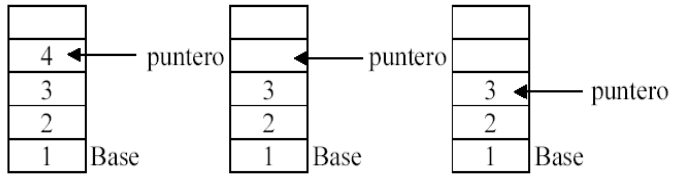
20

POP

- Si la pila está vacía, NO puede realizarse la operación POP y se produce un error de UNDERFLOW.
- Si la pila tiene elementos, se extrae el elemento en la posición indicada por el apuntador, y el apuntador se decrementa en uno.

21

Ejemplo: POP



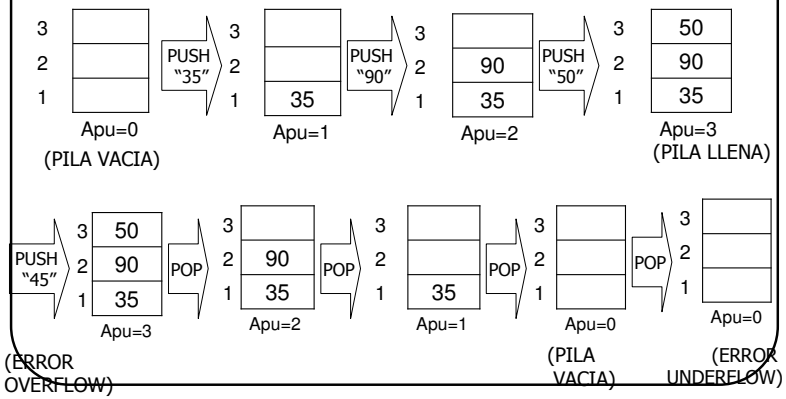
Pila Antes del POP

Pila después del POP

22

EJEMPLO

- Considere una pila numerica de 3 posiciones para realizar las siguientes operaciones consecutivas:



23

COLAS

- Estructura en la que los elementos ingresan por un extremo y salen por otro con filosofía FIFO.

First Input First Output — Primero en Entrar, primero en Salir

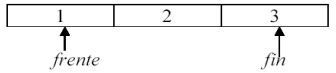
24

COLAS

- Para implementar COLAS en memoria puede utilizarse un vector, y dos variables llamadas “FRENTE” y “FIN” para indicar la posición de los elementos que se encuentran al frente y al fin de la cola, respectivamente.

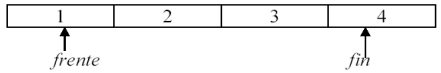
25

Operaciones posibles con COLAS



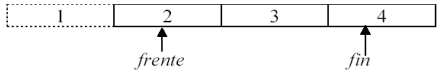
- **Añadir un Elemento**

Añadimos un elemento (4):



- **Extraer un Elemento**

Extraemos un elemento:

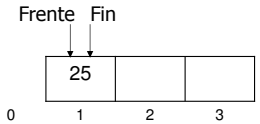


26

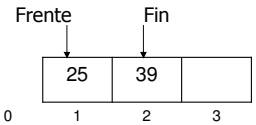
EJEMPLO

- Considere una COLA numerica para realizar las siguientes operaciones consecutivas:

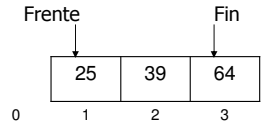
1) Se inicia con una COLA VACIA



2) Ingresa un “25”



3) Ingresa un “39”

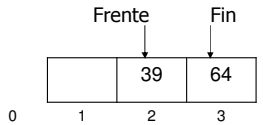


4) Ingresa un “64”

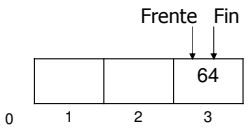
27

EJEMPLO (continuación)

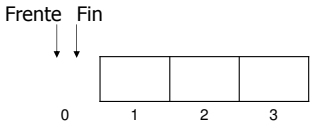
- Considere una COLA numerica para realizar las siguientes operaciones consecutivas:



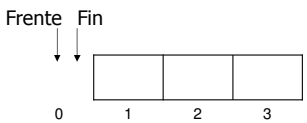
5) Extraer Elemento



6) Extraer Elemento



7) Extraer Elemento
Queda COLA VACIA

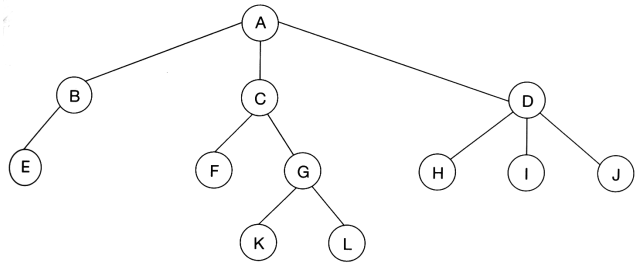


8) Extraer Elemento
ERROR...Cola Vacia

28

Arboles Generales

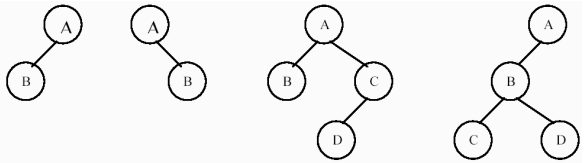
- Estructura de datos con una raíz y varios "hijos" (o subárboles).



29

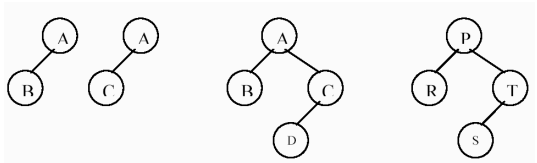
Arboles Binarios

- Estructura de datos homogénea con un nodo raiz y máximo dos nodos hijos: derecho e izquierdo.
- **Arboles Binarios Distintos.-** Si sus estructuras son diferentes.

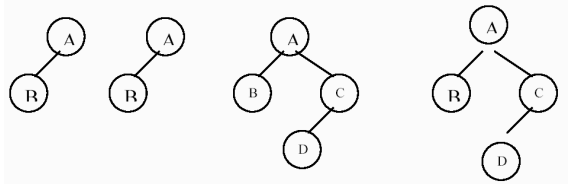


30

- **Arboles Binarios Similares.-** Estructuras idénticas con información diferente.

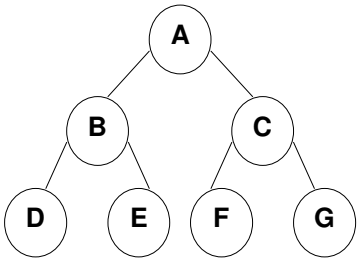


- **Arboles Binarios Equivalentes.-** Son Arboles Binarios Similares con la misma información.



31

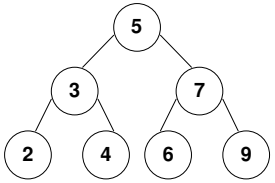
- **Arboles Binarios Completos.-** Todos sus nodos, excepto los del útimo nivel tienen dos hijos.



32

Arboles Binarios de Búsqueda

- Es un árbol binario que tienen las siguientes características:
  - Los elementos a la izquierda de un nodo son menores que él.
  - Los elementos a la derecha de un nodo son mayores que él.



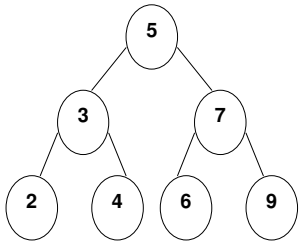
33

Recorridos en Arboles Binarios

- Recorrer significa visitar los nodos del árbol en forma sistemática para que todos los nodos sean visitados una sola vez.
- **Preorden**
  - Visitar la raiz
  - Recorrer el subárbol izquierdo
  - Recorrer el subárbol derecho
- **InOrden**
  - Recorrer el Subárbol izquierdo
  - Visitar la raíz
  - Recorrer el subárbol derecho
- **PostOrden**
  - Recorrer el subárbol izquierdo
  - Recorrer el subárbol derecho
  - Visitar la Raíz.

34

Ejemplo de Recorrido



- Preorden:

5, 3, 2, 4, 7, 6, 9

- InOrden:

2, 3, 4, 5, 6, 7, 9

- PostOrden:

2, 4, 3, 6, 9, 7, 5

35

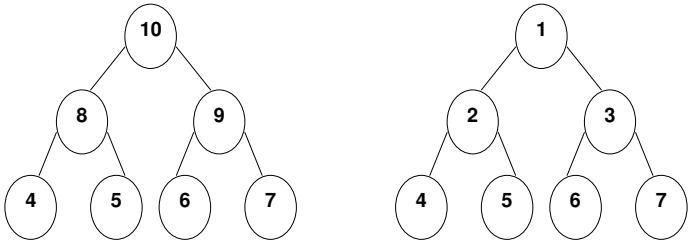
Montículo (Heap)

- Es un Arbol Binario Completo tal que el valor de la clave de cada nodo es mayor o igual (o menor o igual) que las claves de sus nodos hijos (Si los tiene).

- Montículo de máximos (MaxHeaps) (>=)
- Montículo de mínimos (MinHeaps) (<=)

36

Ejemplo de Montículos

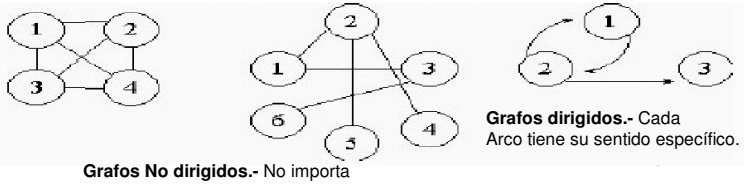


MAXHEAP **MINHEAP**

37

Grafos

- Estructura de datos no lineal utilizadas para representar relaciones arbitrarias (Arcos) entre elementos (Nodos).
- Ejemplos de grafos: Red de carreteras, red de tráfico aéreo, instalaciones eléctricas, rutas de viaje, etc.



Grafos dirigidos.- Cada Arco tiene su sentido específico.

Grafos No dirigidos.- No importa el sentido del Arco

38

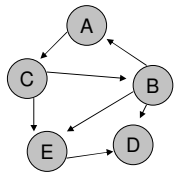
Conceptos de Grafos

- **Orden de un grafo.-** Número de nodos que posee.
- **Camino.-** Secuencia de uno o mas arcos que conectan dos nodos.
- **Longitud de un camino.-** Numero de Arcos que forman un camino.
- **Nodos Adyacentes.-** Si hay un arco que los une.
- **Lazo.-** Arco que conecta un nodo consigo mismo.

39

Representaciones de Grafos

- Matriz de adyacencia

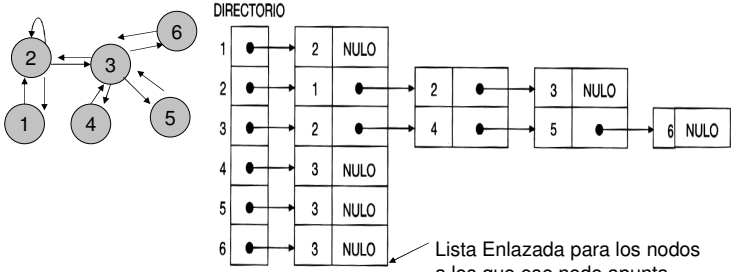


	A	B	C	D	E
A	0	0	1	0	0
B	1	0	0	1	1
C	0	1	0	0	1
D	0	0	0	0	0
E	0	0	0	1	0

40

Representaciones de Grafos

- Lista de adyacencia



41

Archivos

Archivos de Almacenamiento
Dispositivos
Parámetros
Formas de Almacenaje

Accesos
Secuencial
Directo
Secuencial Indexado

Conceptos de Archivos

- **Archivo.-** Colección de información en almacenamiento secundario.
- **Archivo.-** Conjunto de datos estructurados en una colección de registros que constan de diferentes campos. Los datos de un archivo están organizados por un propósito específico.
- **Registro.-** Conjunto de campos.
- **Campo.-** Dato elemental con tamaño y tipo de dato bien definido.

43

Dispositivos de Almacenamiento

- Discos Duros
- Diskettes
- CDs, DVDs
- Memory stick & Flash Drives
- Iomega ZIP 100 Mb
- Jaz Iomega 1GB o mas

44

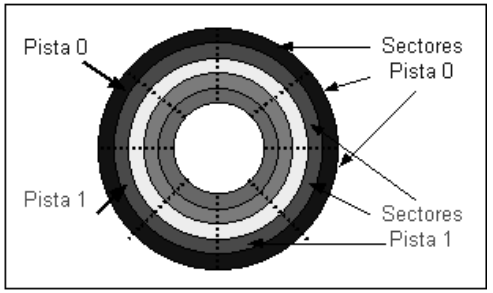
Comparativa CDs/ DVDs

SOPORTE	CAPACIDAD DE ALMACENAMIENTO	DURACIÓN MÁXIMA DE AUDIO	DURACIÓN MÁXIMA DE VÍDEO	NÚMERO DE CDs A LOS QUE EQUIVALE
Disco compacto (CD)	650 Mb	1 h 18 min.	15 min.	1
DVD una cara / una capa	4,7 Gb	9 h 30 min.	2 h 15 min.	7
DVD una cara / doble capa	8,5 Gb	17 h 30 min.	4 h	13
DVD doble cara / una capa	9,4 Gb	19 h	4 h 30 min.	14
DVD doble cara / doble capa	17 Gb	35 h	8 h	26

45

Parámetros de Almacenamiento

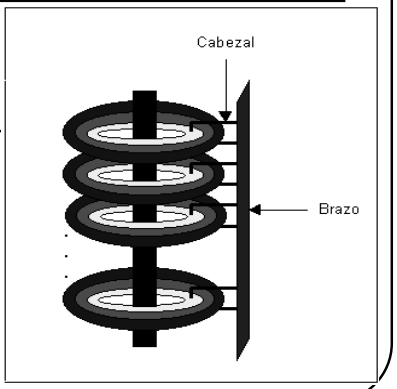
- Pistas y Sectores



46

Parámetros de Almacenamiento

- **Tiempo de Acceso.-** Tiempo que tarda la cabeza del disco en situarse en un lugar determinado y recuperar cierta información. (mSegs).
- **Cilindro.-** Es una pila tridimensional de pistas verticales de los múltiples platos.



47

Formas de Almacenaje (Modo Texto)

- **Modo Texto.-** El contenido del archivo es una secuencia de caracteres ASCII divididos en líneas, cuyo último carácter es el de nueva línea.
- Pueden almacenar canciones, fuentes de programas, archivos simples, etc.
- Los **archivos** en modo **texto** se caracterizan por ser planos. Todas las letras tienen el mismo formato y no hay palabras subrayadas, en negrita, o letras de distinto tamaño o ancho.

48

Formas de Almacenaje (Modo Binario)

- **Modo Binario**.-Los archivos binarios son diferentes a los archivos de texto principalmente porque los datos no estan organizados en líneas y pueden contener cualquier valor de ocho bits.
- Un **archivo binario** es una secuencia de bytes que tienen una correspondencia uno a uno con un dispositivo externo.
- El número de bytes escritos (leídos) será el mismo que los encontrados en el dispositivo externo.
- Ejemplos: fotografías, imágenes, texto con formatos, archivos ejecutables, etc.

49

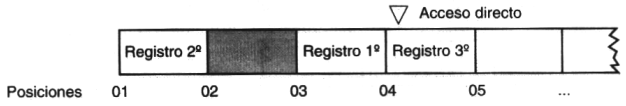
Accesos

- **Secuencial.-** Los registros ocupan posiciones consecutivas y solo se puede acceder a ellos de uno en uno a partir del primero.

50

Accesos

- **Acceso Directo.-** La información se coloca y se accesa aleatoriamente mendiante su posición relativa dentro del conjunto de posiciones posibles.
- Se pueden leer y escribir registros en cualquier orden y en cualquier lugar.
- El orden físico no corresponde al órden lógico.
- Cada registro deben contener una CLAVE que lo identifique de manera única. Todos los registros deben ser del mismo tamaño.
- Hay una correspondencia entre los posibles valores de la clave y las direcciones físicas.



51

Accesos

CLAVE	DIRECCIÓN
15	010
24	020
36	030
54	040

Area de índices

	CLAVE	DATOS
Área principal 010		
011		
012		
019	15	
020		
021		
029	24	
030		
031		
039	36	
040		
041		
049	54	

Secuencial indexado.-

- En el área principal están los datos desordenados.
- En el área de índices se tiene cada clave con la dirección de inicio del bloque donde se encuentra.
- Reduce el tiempo de búsqueda.
- Funciona como un “diccionario”

52

Fundamentos de Algorítmica

Tipos de instrucciones
Secuenciales
Condicionales
Iterativas

Algoritmos
Modulares
Recursivos

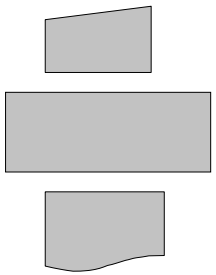
Algoritmo

- Secuencia finita, lógica y ordenada de pasos a seguir para la resolución de un problema particular.

54

Instrucciones Secuenciales

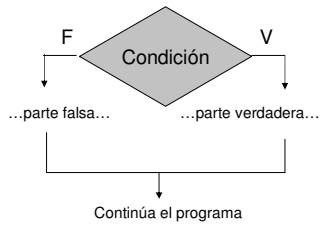
- Entrada
  - Petición de datos (Captura)
- Proceso (Cálculo)
  - Operaciones y asignaciones
- Salida
  - Impresión



55

Instrucciones Condicionales (IF)

```
if (Condicion)
{   parte verdadera
}
else
{   parte falsa
}
```

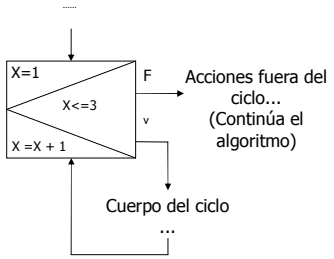


Si la condición se cumple, se ejecuta la “parte verdadera”.
De lo contrario, se ejecuta la “parte falsa”.
NUNCA se ejecutan las dos partes al mismo tiempo.

56

Instrucciones de iteración (FOR)

```
for(x=1; x<=3; x=x+1)
  {   cuerpo del ciclo
  }
  Acciones fuera del ciclo…
```



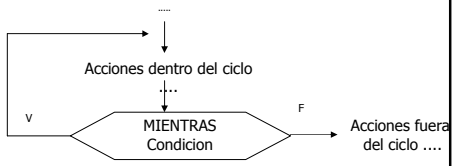
El ciclo FOR siempre tiene una variable de control que toma un valor inicial,
Y en cada "vuelta" del ciclo, incrementa su valor según se especifique.
El ciclo sigue en ejecución MIENTRAS la condición sea verdadera.

57

Instrucciones de iteración (DO WHILE)

```
do
{
   …acciones
   dentro del ciclo…

} while (condicion);
…acciones fuera del ciclo…
```

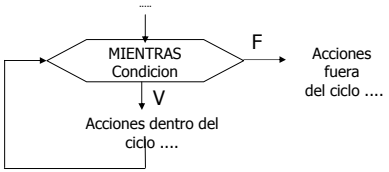


El ciclo DO WHILE se ejecuta siempre al menos una vez.
Seguirá ejecutándose MIENTRAS la condición sea verdadera.

58

Instrucciones de iteración (WHILE)

```
while(condicion)
{  …acciones dentro del ciclo…
}
…acciones fuera del ciclo…
```



El ciclo WHILE se ejecuta Siempre MIENTRAS la condición sea verdadera.
Si la condición no es verdadera al entrar por primera vez, el ciclo no se ejecuta.

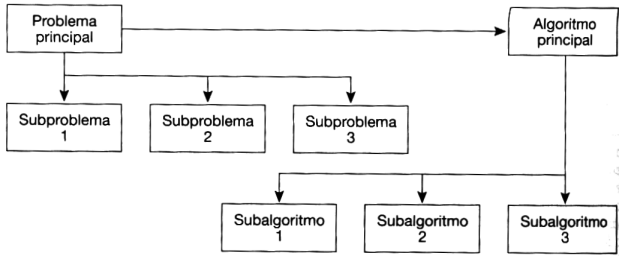
59

Modularidad

- Consiste en dividir un problema en varias subtareas de menor tamaño, haciéndolo mas facil de entender y modificar.
- **MODULO.-** Está constituido por varias instrucciones con un propósito lógico y agrupadas bajo un ***nombre***, el cual puede invocarse desde diferentes partes de un programa.
- Un módulo puede ser:
  - Una función
  - Una subrutina (procedimiento, o subprograma)

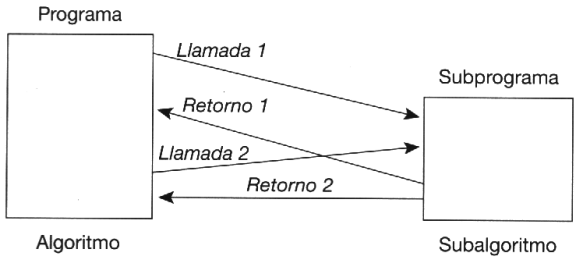
60

Descomposición en módulos



61

Funcionamiento de los Módulos



62

Tipos de módulos

- **SUBRUTINA (Subprograma o Procedimiento).-** Es un módulo que puede recibir pero NO entregar (regresar) parámetros al módulo que lo llama.
- **FUNCION.-** Es un módulo que recibe y/o entrega (regresa) parámetros al módulo llamador. Las funciones pueden regresar datos simples o estructurados.

63

Módulos especiales

- Unidades, bibliotecas, clases, paquetes.-
  - Generalmente son rutinas precompiladas que pueden incluirse en cualquier programa.
  - Se requiere “incluirlas” explícitamente en el programa para poder usar las rutinas definidas dentro de ellas.
  - Promueven la reutilización de código.
  - Permiten que el programador use ciertas funciones sin necesidad de escrbirlas o saber como funcionan internamente.

64

Vinculación estática y dinámica

- **Vinculación estática.-** Las funciones y módulos se incluyen en *tiempo de compilación*.
- **Vinculación dinámica.-** Permite que un programa busque el código necesario de una función en *tiempo de ejecución*.

65

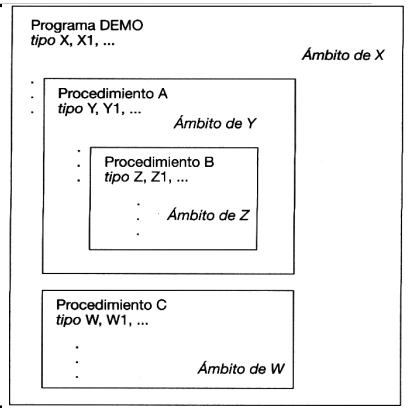
Parámetros

- Los valores que un módulo recibe o entrega (regresa) como resultado se llaman parámetros.
- Generalmente los parámetros son ***variables locales*** (de cualquier tipo: entero, flotante, char, booleano, etc.) que se envían de un módulo a otro para que éste último pueda utilizarlas también, aún sin estar declaradas dentro de él.

66

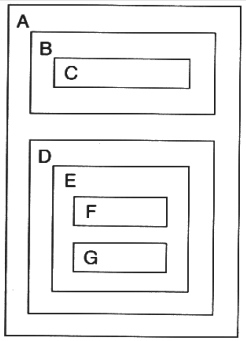
Variables globales y locales

- Las variables ***globales*** pueden utilizarse en todos los módulos.
- Las variables ***locales*** se declaran dentro de un módulo particular y están accesibles solo dentro del cuerpo de ese módulo.



67

Ambito de las variables



Variables definidas en…	Accesibles desde…
A	A,B,C,D,E,F,G
B	B,C
C	C
D	D,E,F,G
E	E,F,G
F	F
G	G

Ambito.- Parte del programa donde se conoce a la variable.

68

Otras formas de ocultar/compartir variables

- Algunos lenguajes de programación soportan el uso de "modificadores de acceso" que le permiten al programador decidir quién accesa las variables.
  - **Public –** Accesibles desde cualquier otra aplicación.
  - **Private –** Accesibles unicamente dentro de un ámbito especifico.
  - **Friend –** Accesibles solo para ciertas aplicaciones.
  - **Internal –** Accesibles para módulos de la misma aplicación.

69

Comunicación con los módulos: Paso de parámetros

- **Parámetros de Entrada.-** Valores que se reciben desde el programa llamador.
- **Parámetros de Salida.-** Resultados enviados al programa llamador.
- **Paso de parámetros por Valor.-** Los valores originales NO se modifican. Se realiza una copia de su valor y con ella se trabaja.
- **Paso de parámetros por Referencia.-** Los cambios realizados SI se reflejan en los valores originales.

70

Recursión

- Ocurre cuando un programa (o módulo) se llama a sí mismo.
- Cuando se usa recursión, SIEMPRE se debe establecer un "Estado Básico", es decir, un momento en el cual la solución sea directa y no recursiva.
- La solución SIEMPRE debe acercarse al "Estado Básico"

71

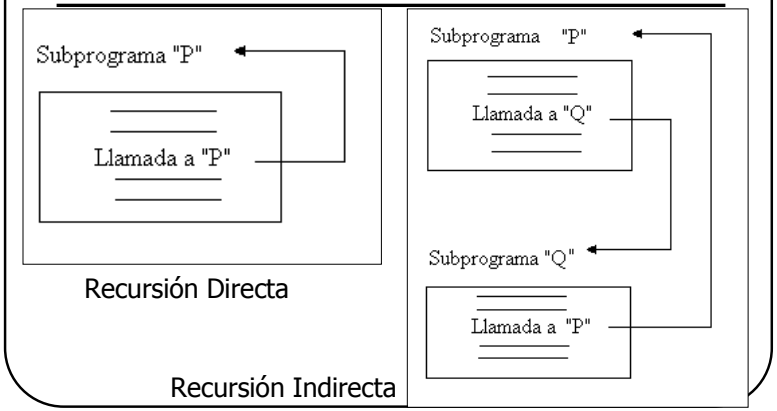
Todos los Programas recursivos...

- Poseen la solución recursiva y además el estado básico (Solución no-recursiva)
- Generalmente se usa una decisión para determinar a cual de ambas recurrir y evitar que el programa se ejecute indefinidamente.

```
if (condicion)
{    …solucion recursiva…
}
else
{   …solucion no-recursiva…[Estado básico]
}
```

72

Tipos de Recursión



73

Métodos de Búsqueda y Ordenación

Ordenamiento Interno
Burbuja
Inserción
Selección
QuickSort
MergeSort

Ordenamiento Externo
Intercalación
Mezcla Directa
Mezcla Natural

Búsqueda
Secuencial
Binaria

Métodos de Ordenamiento

- **Ordenar** significa Reorganizar un conjunto de datos de acuerdo a una secuencia específica.
- Los métodos de ordenamiento pueden ser:
- Internos
  - Se llevan a cabo en memoria principal
  - Generalmente se usan arreglos (Vectores)
- Externos
  - Se llevan a cabo en memoria secundaria
  - Se usan archivos

75

Parámetros de Eficiencia en los métodos de ordenamiento

- Estabilidad.- ¿Cómo se comporta con elementos iguales?
- Complejidad de Implementación
- Tiempo de Ejecución (Rendimiento)
  - Número de comparaciones
  - Número de intercambios
- Requerimientos de memoria

76

Complejidad

- $O(n^2)$ Complejidad Cuadrática
  - Rendimiento regular a pobre [Generalmente Aceptable]
  - Mas sencillos de implementar
- O(n log n) Complejidad Logarítmica
  - Preferibles por su mejor rendimiento
  - Mas complejos de implementar
- "n" es el número de elementos a ordenar
- Se usa "C" para el Num. de Comparaciones
- "M" para el Num. de Movimientos o intercambios.

77

Burbuja

- Consiste en comparar elementos adyacentes de dos en dos. Si un elemento es mayor que el que está en la siguiente posición se intercambian hasta que todos se encuentren ordenados.
- Puede ser implementado para:
  - Llevar los elementos mas pequeños a la parte izquierda
  - Llevar los elementos mas grandes a a la parte derecha.
- Es el mas ineficiente por hacer muchas comparaciones y movimientos.
- Es el mas fácil de implementar

78

Ejemplo

Ordenar "4, 3,5,2,1" por el método de la Burbuja

4	3	5	2	1	
3	4	5	2	1	
3	4	2	5	1	
3	4	2	1	5	Fin Pasada #1
3	2	4	1	5	
3	2	1	4	5	Fin Pasada #2
2	3	1	4	5	
2	1	3	4	5	Fin Pasada #3
1	2	3	4	5	Fin Pasada #4

79

Inserción Directa

- Usado por los jugadores de cartas para ordenarlas.
- Consiste en insertar un elemento del arreglo en la parte izquierda del mismo, que se encuentra ya ordenada. El proceso se repite desde el segundo hasta el n-ésimo elemento.

80

Ejemplo Inserción Directa

#1	15	67	08	16	44	27	12	35
#2	15	67	08	16	44	27	12	35
#3	08	15	67	16	44	27	12	35
#4	08	15	16	67	44	27	12	35
#5	08	15	16	44	67	27	12	35
#6	08	15	16	27	44	67	12	35
#7	08	12	15	16	27	44	67	35
#8	08	12	15	16	27	35	44	67

El elemento de cada posición se va ordenando a la izquierda del arreglo

81

Selección Directa

- Consiste en buscar el menor elemento del arreglo y colocarlo en la primera posicion.
- Luego se busca el segundo elemento mas pequeño y se coloca en la segunda posición.
- El proceso continúa hasta que todos los elementos del arreglo han sido ordenados.

82

Ejemplo Selección Directa

#1	15	67	08	16	44	27	12	35
#2	08	67	15	16	44	27	12	35
#3	08	12	15	16	44	27	67	35
#4	08	12	15	16	44	27	67	35
#5	08	12	15	16	44	27	67	35
#6	08	12	15	16	27	44	67	35
#7	08	12	15	16	27	35	67	44
#8	08	12	15	16	27	35	44	67
#9	08	12	15	16	27	35	44	67

En cada ejecución se selecciona el menor elemento del resto del vector y se ordena hacia la izquierda.

83

QuickSort

- Es uno de los mas eficientes. Consiste en:
  - Elegir el elemento central (X) del arreglo [Se denomina Pivote].
  - Buscar por la izquierda del arreglo un elemento A mayor que X.
  - Buscar por la derecha del arreglo un elemento B menor o igual a X.
  - Intercambiar A y B.
  - Seguir con el proceso hasta que X divida el arreglo en dos subarreglos tales que el izquierdo contiene los elementos <= que X y el derecho todos los elementos > X. Por lo tanto X está ordenado.
  - Hacer QuickSort al subarreglo izquierdo y al Derecho.

84

Ejemplo QuickSort

Elemento Pivote
Elementos a Intercambiar
✓ Elemento ya ordenado

Buscar elemento > Pivote → ← Buscar elemento <= Pivote

#1	80	36	98	62	26	78	22	27	2	45
#2	2	36	98	62	26	78	22	27	80	45
#3	2	22	98	62	26	78	36	27	80	45
#4	2	22	26	62	98	78	36	27	80	45
#5	2	22	26✓	62	98	78	36	27	80	45
#6				27	98	78	36	62	80	45
#7				27	36✓	78	98	62	80	45
#8						45	98	62	80	78
#9						45	62✓	98	80	78
#10								78	80✓	98
#11	2	22	26	27	36	45	62	78	80	98

El 26 ya está ordenado
A la izq solo hay nums <26
A la der solo hay nums > 26
Se crean dos SubVectores (Derecho e Izquierdo)
A cada uno se le aplica el mismo procedimiento QuickSort.

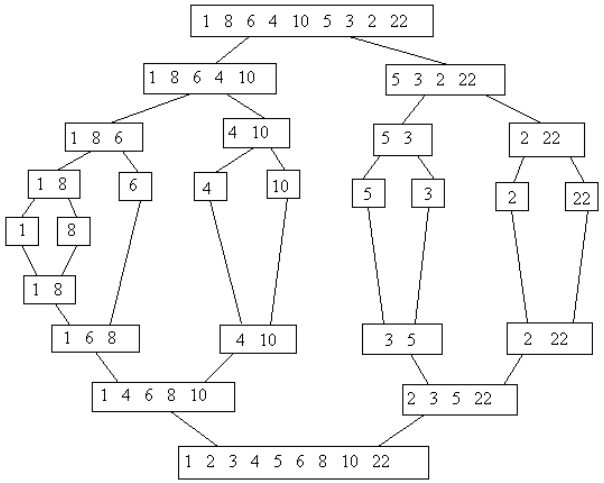
85

MergeSort

- Divide el arreglo original en dos arreglos separados.
- Divide esos arreglos separados en mas arreglos separados.
- Cada arreglo es recursivamente ordenado. Finalmente se unen los elementos en un solo vector.
- Complejidad: O (n log n)

86

Ejemplo MergeSort



87

Comparativa

Método	Ventajas	Desventajas
Burbuja	Fácil	Mas ineficiente Muchas comparaciones
Selección	Fácil Pocos intercambios	Lento Muchas comparaciones
Inserción	Fácil implementar Requiere poca memoria	Lento Muchas comparaciones
QuickSort	Muy Rapido en arreglos desordenados O(n log n)	Lento en arreglos ordenados Implementación compleja Recursivo
MergeSort	Rápido en grandes cantidades de datos	Complejo Recursivo y complejo Usa mucha memoria adicional

88

Número de Comparaciones y Movimientos de algunos métodos

		Ordenado	Desordenado	Orden Inverso
Burbuja	**Comp.**	$(n^2-n) / 2$	$(n^2-n) / 2$	$(n^2-n) / 2$
	Mov.	0	$0.75 * (n^2-n)$	$1.5 * (n^2-n)$
Inserción Directa	**Comp.**	$(n-1)$	$(n^{2+}n-2) / 4$	$(n^2-n) / 2$
	Mov.	0	$(n^2-n) / 4$	$(n^2-n) / 2$
Selección Directa	**Comp.**	$(n^2-n) / 2$	$(n^2-n) / 2$	$(n^2-n) / 2$
	Mov.	n-1	n-1	n-1

89

Ordenamiento externo

- Se realiza en archivos generalmente con muchos registros.
- Algunos métodos conocidos son:
  - Fusión, mezcla o intercalación
  - Mezcla directa
  - Mezcla natural

90

Fusión o Mezcla de Archivos

- Consiste en reunir en un archivo los registros de 2 o mas archivos ordenados por un campo clave.
- El archivo resultante será un archivo ordenado por el campo clave.

F1	12	24	36	37	40	52	*				
F2	3	8	9	20	*						
F3	3	8	9	12	20	24	36	37	40	52	*

91

Mezcla Directa

F: 19 27 2 8 36 5 20 15 6
Se particiona en secuencias de 1
F1: 19 2 36 20 6
F2: 27 8 5 15
Se fusiona cada par de manera ordenada
F: 19 27 2 8 5 36 15 20 6
Se particiona en secuencias de 2
F1: 19 27 5 36 6
F2: 2 8 15 20
Se fusiona cada par de manera ordenada:
F: 2 8 19 27 5 15 20 36 6
Se particiona en secuencias de 4
F1: 2 8 19 27 6
F2: 5 15 20 36
Se fusiona cada par de manera ordenada:
F: 2 5 8 15 19 20 27 36 6
Se particiona en secuencias de 8:
F1: 2 5 8 15 19 20 27 36
F2: 6
Se fusiona cada par de manera ordenada:
F: 2 5 6 8 15 19 20 27 36
[FIN DEL PROCESO]

92

Mezcla Natural

F: 19 27 2 8 36 5 20 15 6

Se particiona aprovechando el orden ascendente natural que puede existir:
[Se deja de incluir elementos en la particion cuando el "orden" se rompe]
F1: 19 27\ 5 20\ 6
F2: 2 8 36\ 15
Se fusionan las particiones de manera ordenada:
F3: 2 8 19 27 36\ 5 15 20\ 6

Se particiona:
F1: 2 8 19 27 36\ 6
F2: 5 15 20
Se fusiona:
F3: 2 5 8 15 19 20 27 36/ 6

Se particiona:
F1: 2 5 8 15 19 20 27 36
F2: 6
Se Fusiona:
F3: 2 5 6 8 15 19 20 27 36
[FIN DEL PROCESO]

93

Búsqueda Secuencial

- Consiste en recorrer el vector elemento por elemento y comparar cada uno con el elemento que se busca.
- Ineficiente para vectores grandes
- Hay dos posibles resultados:
- **EXITO.-** Se encuentra el elemento, se muestra la posición y termina la búsqueda.
- **FRACASO.-** Se terminó de recorrer todo el vector y el elemento no se encuentra.

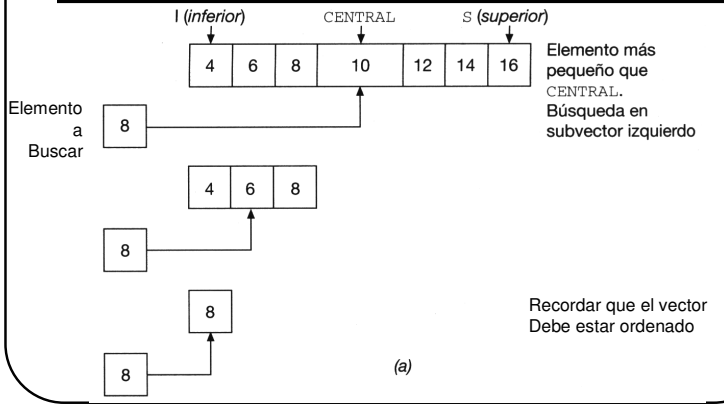
94

Búsqueda Binaria

- Requiere un vector de elementos ordenados para realizar la búsqueda.
- Se examina primero el elemento central del vector.
- Si éste es el elemento buscado, la búsqueda termina.
- Si no, se determina si el elemento buscado está en la primera o segunda mitad de la lista.
- Se repite el proceso utilizando el elemento central de esa sublista.

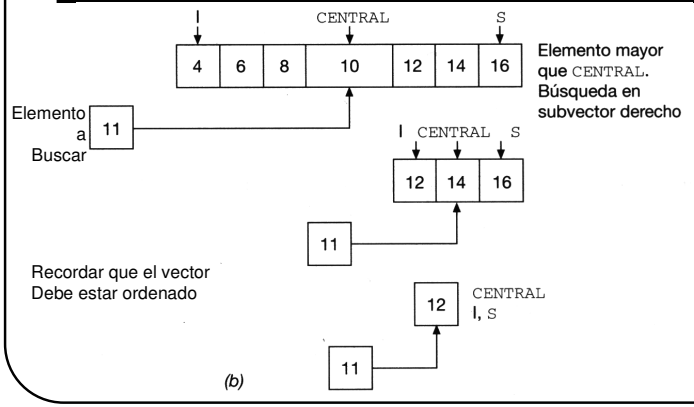
95

Ejemplo Búsqueda Binaria a) Elemento encontrado



96

Ejemplo Búsqueda Binaria
b) Elemento no encontrado



97

Paradigmas de programación

Programación:

- Estructurada (C)
- Orientada a Objetos (C++)
- Funcional (Lisp)
- Lógica (Prolog)
- Visual

Programación Estructurada

- Inició en la década de los 60s
- El código se escribe de manera tal que se pueda leer de principio a fin de manera contínua.
- Se basa en el uso de 3 estructuras lógicas de control:
  - Secuencia
  - Selección Condicional
  - Iteración (Ciclo)

99

Programación Estructurada

- TEOREMA DE LA ESTRUCTURA.-
  - Un programa propio puede ser escrito usando solamente estructuras de secuencia, selección e iteración.
  - Un programa es propio si:
    - Tiene exactamente una entrada y una salida para control del programa.
    - Existen caminos seguibles desde la entrada hasta la salida que conducen por cada parte del programa.
    - No tiene lazos infinitos.
    - No tiene instrucciones que nunca se ejecutan.

100

Programación Estructurada

- A pesar de que se pueden utilizar, las instrucciones GOTO se evitan siempre.
- Se prefiere usar módulos bien diseñados.
- La identación se usa en bloques de código para dar mas legibilidad al programa.

101

Ejemplo Programación Estructurada en C

```
#include <iostream.h>
void main (void)
{
    int NE;
    float salario;
    cout << "Introduce el Numero de Empleado:";
    cin >> NE;
    cout << "Introduce el Salario: ";
    cin >> salario;
    If (salario < 1000)
    {       cout <<" El empleado recibe un bono de 200 pesos";
    }
    else
    {       cout << "El empleado NO recibe BONO";
    }
}
```

102

Programación Orientada a Objetos (POO)

"Es un método de implementación en el que los programas se organizan como colecciones **cooperativas** de **objetos**, cada uno de los cuales representan una **instancia** de alguna **clase** y cuyas clases son todas miembros de una jerarquía de clases unidas mediante relaciones de **herencia**"

Grady Booch

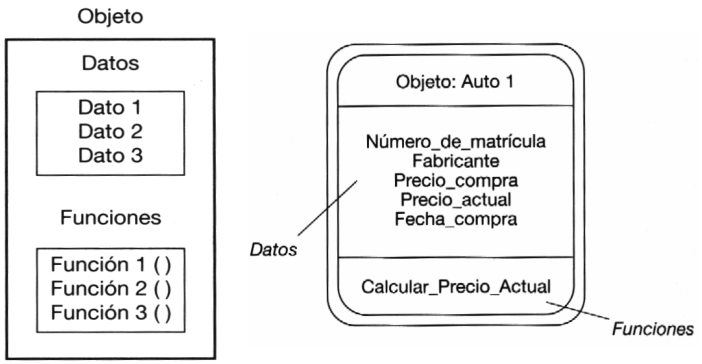
103

Objeto

- Unidad que combina datos y funciones.
  - Datos = Propiedades = Atributos = Características
  - Funciones = Métodos = Procedimientos =Acciones
- Los datos y funciones están **Encapsulados**.
- Posee un nombre único (identificador).
- Un objeto es miembro de una clase
- "Un objeto es la instancia de una clase"
- Ejemplo de Objeto: Cliente, Factura, Persona, etc.

104

Ejemplo notación gráfica de un objeto



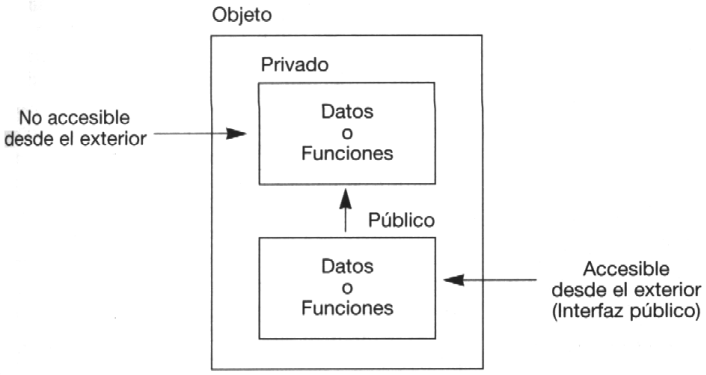
105

Encapsulamiento

- Permite incluir en una sola entidad información y operaciones que operan sobre esa información.
- Permite:
  - Componentes públicos [Accesibles al usuario].
  - Componentes privados [No accesibles].
  - Restricción de accesos indebidos.

106

Ejemplo Encapsulamiento



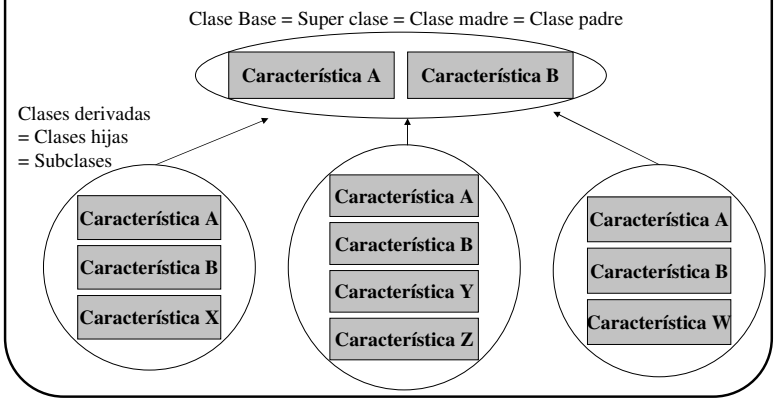
107

Clases

- Una clase es la descripción de un conjunto de objetos. Contiene todas las características comunes de ese conjunto.
- Una clase es la declaración de un tipo objeto.
- Clase = Modelo = Plantilla = Esquema = Descripción de la anatomía de los objetos.
- A partir de una clase se pueden definir [crear] muchos objetos independientes con las mismas características.

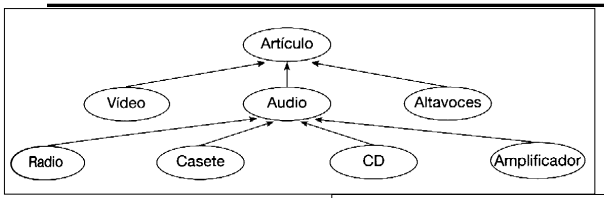
108

Herencia

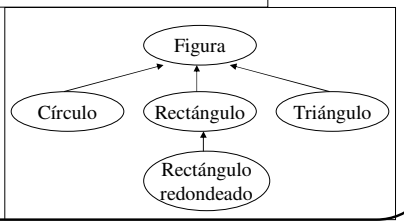


109

Ejemplos de Herencia



Ejemplo 1



Ejemplo 2

110

Ejemplo de POO en C#

```
using System;
class Programa
{
   public static void Main()
   {
      Persona Persona1 = new Persona();
      Persona1.Nombre = "Ramon";
      Console.WriteLine("La Persona1 se llama: " + Persona1.Nombre);
   }
}

class Persona
{   private string nombreLocal;

    public string Nombre
    {  get { return(nombreLocal);  }
       set { nombreLocal = value; }
    }
}
```

111

Programación Funcional (LISP)

- Utilizado en la construcción de sistemas expertos.
- LISP = List Processing (Procesamiento de listas)
- LISP ofrece facilidades para procesamiento simbólico.
- LISP es un lenguaje de programación funcional en el que se definen funciones sencillas que forman funciones mas complejas.
- Funciones en LISP.- Definen procedimientos que describen la acción que se va a tomar cuando se ejecute.

112

Ejemplo en LISP

Programa que calcula el promedio de n Numeros

```
(load common)
(load irratnal)
(clear-screen)
(print “Este programa pide n numeros para promediarse y da el
    resultado”)
(princ “Cuantos nums desea promediar? “)
(setq a (read) )
(setq resultado 0)
( dotimes (n a)
    (princ “Introduzca un numero para promediar: “)
    (setq numero (read) )
    (setq resultado ( + numero resultado ) ) )
(setq resultado ( / resultado a) )
(princ “El promedio es: “ )
(princ resultado)
```

113

Programación Lógica (PROLOG)

- Permite representar reglas lógicas
- Prolog Evalúa esas reglas lógicas y proporciona un resultado.
- Usado en la realización de “programas inteligentes”

Ejemplo:

Comida (manzana)
Comida (pollo)
Come (Juan, X) :- Comida (X)

114

Programación Visual

- Inició en 1991 con la aparición del Visual Basic TM
- El modelo de programación visual incluye:
  - Entorno de desarrollo visual, editor de código.
  - Controles de alto nivel.
  - Acceso directo a las distintas partes del código
  - Creación de formularios y controles.
  - Personalización de Propiedades
  - Poderosas funciones (Acceso a datos, etc.)

115