PROGRAMACIÓN I

TEORÍA – CECILIA SANZ

Temas

- ✓ Repaso
- ✓ Concepto de Estructura de Datos
- ✓ Clasificación
- ✓ Ejemplos

REPASO

1

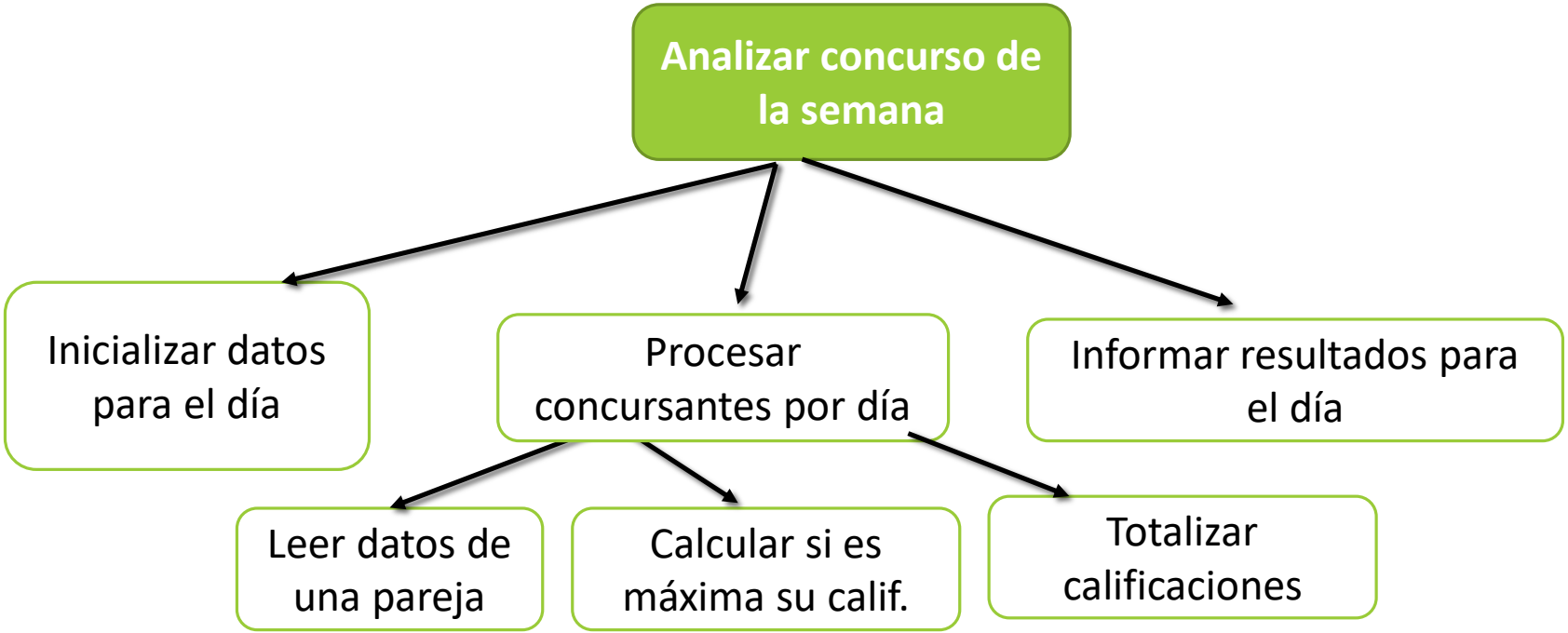
EJERCICIO de REPASO

En un concurso de baile se califican las actuaciones de cada pareja inscripta (20 en total). Las calificaciones van de 1 a 10, siendo 10 la calificación más alta. Las parejas han realizado su actuación 5 días de la semana. Se pide:

a. Leer desde teclado **para cada día** las calificaciones dadas a cada pareja que ha bailado.
b. Informar el promedio de calificaciones de cada día
c. Informar la mayor calificación dada para cada día

1

EJERCICIO de REPASO



1

EJERCICIO de REPASO

```
Program Concurso;
Const
    maxConc=20;
    notaMax = 10;
    Cantdias= 5;
Type
    dias=1..Cantdias;
    calificaciones= 0..notaMax;
    concursantes=1..maxConc;
Var d: dias; max: calificaciones;
total: integer;
Procedure InformarDia(d: dias);
Procedure ProcesarConcursantes(var total: integer; var
max: calificaciones);
{Acá iría la implementación de los procedures}
```

```
{Programa Principal}
Begin
    For d:=1 to  Cantdias do
    begin
        total:=0; max:=0;
        ProcesarConcursantes(total, max);
        InformarDia (d);
        writeln('Promedio: ', total/maxConc);
        writeln('Maximo: ', max);
    End;
End.
```

1

EJERCICIO de REPASO

```
Procedure InformarDia(d: dias);
Begin
    case d of
        1: writeln ('Dia lunes');
        2: writeln ('Dia martes');
        3:  writeln ('Dia miercoles');
        4: writeln ('Dia jueves');
        5: writeln('Dia viernes')
    else writeln( 'Dia erróneo');
    end;
End;
```

```
Procedure ProcesarConcursantes(var total: integer;  var max: calificaciones);
Var cali: calificaciones; i: concursantes;
Begin
    for i: = 1 to MaxConc do
    begin
        readln(cali);
        if (cali > max) then max:= cali;
        total:= total + cali;
    end;
End;
```

2

EJERCICIO de REPASO

Se lee una secuencia de caracteres terminada en '.'. Determinar si la secuencia cumple con el patrón. En caso de no cumplir, informar las partes que no verificaron el patrón.

A@B. donde:

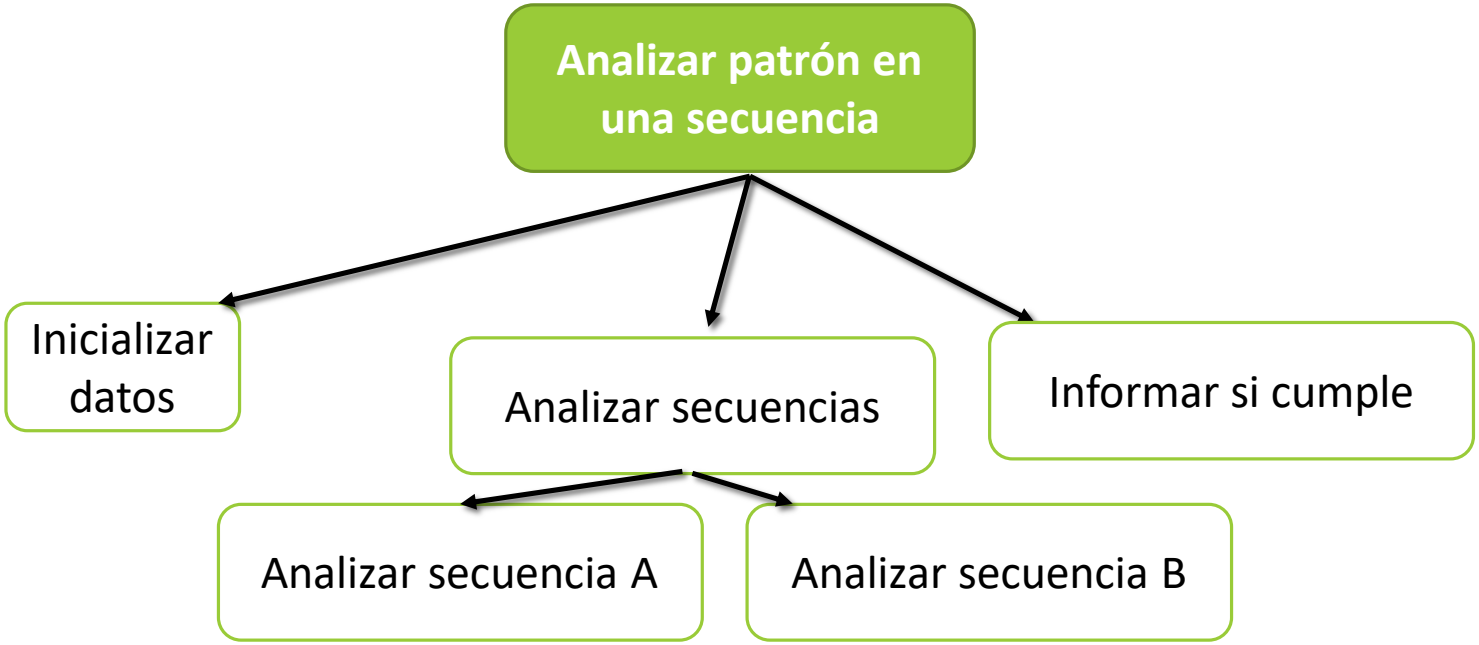
@ es el carácter '@' que seguro existe.

A debe ser una secuencia de letras mayúsculas.

B debe ser una secuencia de <u>caracteres letras mayúsculas que no aparecieron en A</u>.

2

EJERCICIO de REPASO



2

EJERCICIO de REPASO

```
Program AnalizarPatron;
Const
    finSecA='@';  finSecB = '.';
Type
      Letras= set of char; Estados=1..3;
Var cumplePatron: Estados;
Begin
   cumplePatron:=3; {por defecto cumple el patrón}
   ProcesarSecuencias (cumplePatron);
   Case cumplePatron of
       1: writeln ('La secuencia no cumple el patrón
                    en la Secuencia A');
       2:  writeln ('La secuencia no cumple el patrón
           en la Secuencia B');
       3: writeln ('La secuencia Cumple el patrón');
     End;
End.
```

```
Procedure Procesar Secuencias(var
cumplePatron:estados);
Var LM, EstaEnA: Letras; car: char;
cumpleA, cumpleB: boolean;
   Procedure SEC_A (Var
    cumpleA:boolean; LM: letras; var EstaEnA:letras);
   var car: char;
    Begin
       readln(car);
       While (car <> finSecA) and (cumpleA) do
       Begin
          If (car in LM)
          then Begin
                  EstaEnA:= EstaEnA + [car];
                  readln(car);
               end
          Else cumpleA:= false;
       End;
    End;
```

2 EJERCICIO de REPASO

```
{Este proceso está anidado en
  ProcesarSecuencias}

Procedure SEC_B (Var
  cumpleB:boolean; LM, EstaEnA:letras);
var car: char;
  Begin
      readln(car);
      While (car <> finSecB) and (cumpleB) do
      Begin
          If (car IN LM) and not (car IN EstaEnA)
          then  readln (car)
          Else cumpleB:= false;
       End;
  End;
```

```
Begin {Cuerpo del Procedimiento ProcesarSecuencias}
        LM:= ['A'..'Z'];
        EstaEnA:=[];
        cumpleA:=true; cumpleB:=true;
        Sec_A(cumpleA, LM, EstaEnA);
        If (cumpleA)
        then begin
                Sec_B (cumpleB, LM, EstaEnA);
                If not (cumpleB) then cumplePatron:=2;
              end
         else cumplePatron:=1;
End;
```

ABRIR CÓDIGO .PAS

Motivación

Motivación

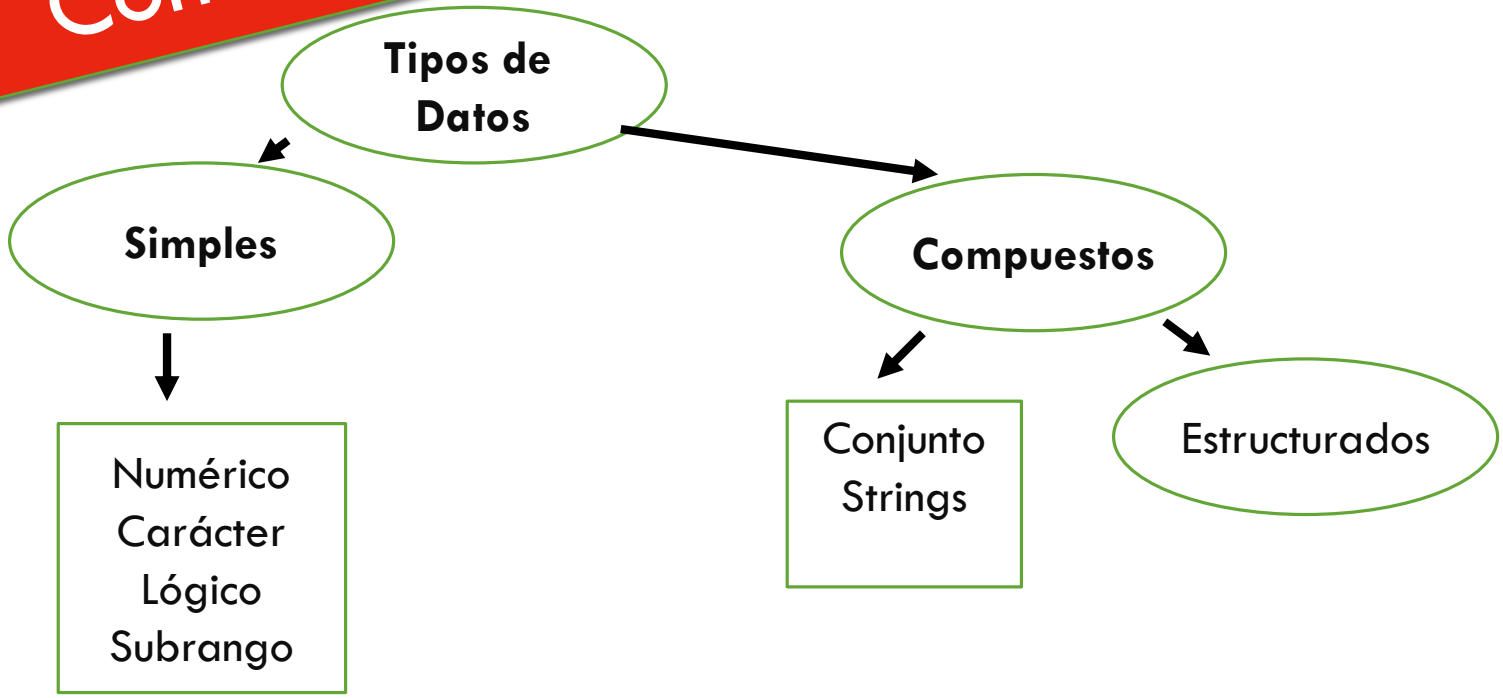
Los tipos de datos definidos por el usuario nos permiten dar representaciones más ajustadas a las datos de los problemas del mundo real

Motivación

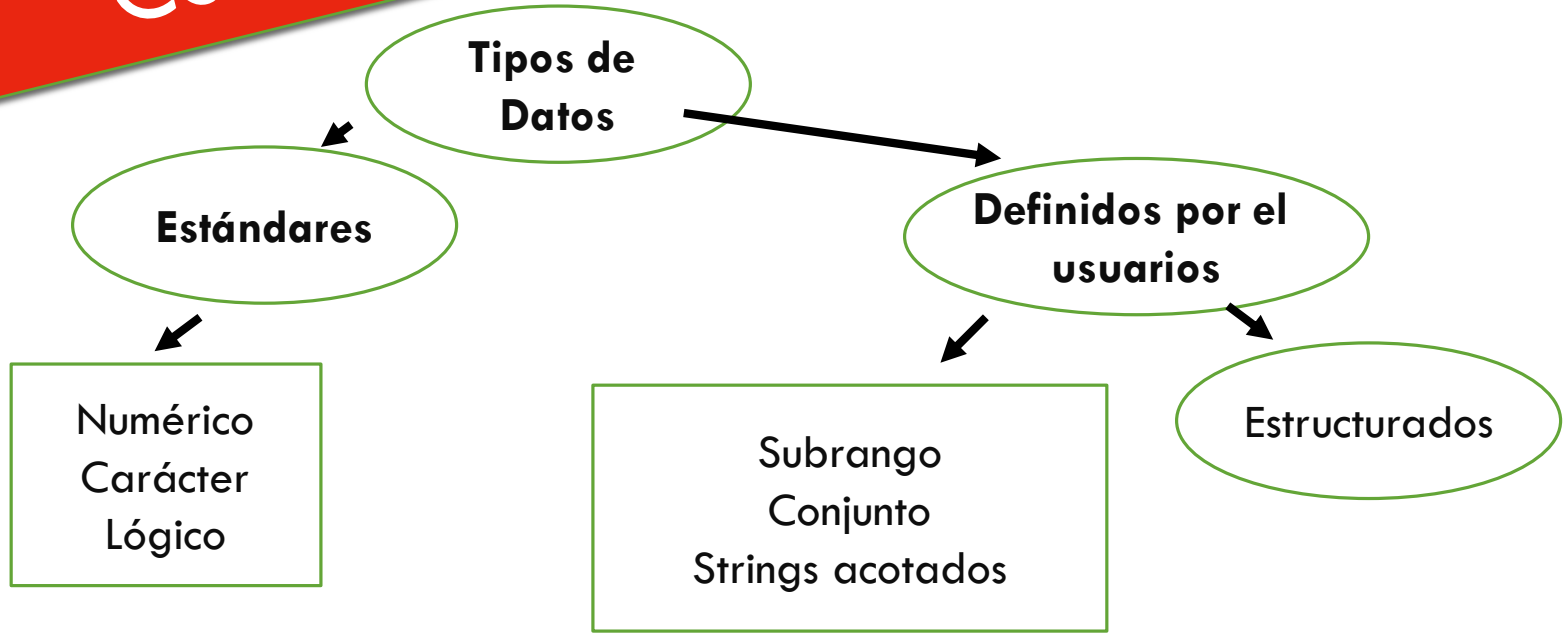
Separados son simples ladrillos, pero al agruparlos con algún sentido constituyen un nuevo objeto, representan algo nuevo…

Contexto

Contexto


Tipos de Datos
Simples
Numérico
Carácter
Lógico
Subrango
Compuestos
Conjunto
Strings
Estructurados

Contexto


Tipos de Datos
Estándares
Definidos por el usuarios
Numérico
Carácter
Lógico
Subrango
Conjunto
Strings acotados
Estructurados

Concepto


business
vision
plan
ideas
management
marketing
workshop
research
motivation
teaching
diagram
innovation
imagination
strategy
leadership
teamwork

Estructura de Datos

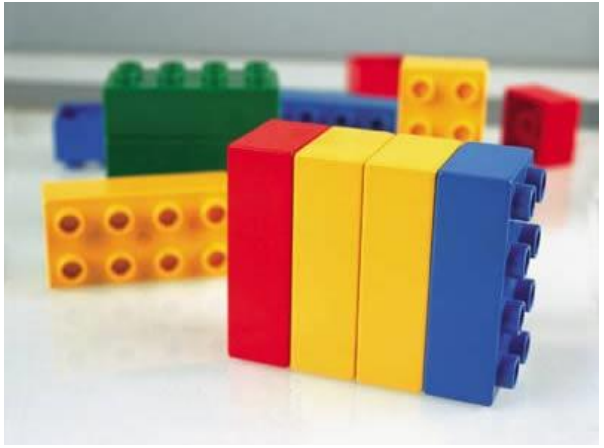
Vamos a empezar a trabajar con la noción de estructura de datos.

TIPO DE DATO ESTRUCTURADO

permite al programador definir un tipo al que se asocian diferentes datos que están lógicamente relacionados y asociados bajo un nombre único.

Definición

Una **estructura de datos** es un conjunto de variables (que podrían ser de distinto tipo) relacionadas entre sí y que se puede operar como un todo, bajo un nombre único.



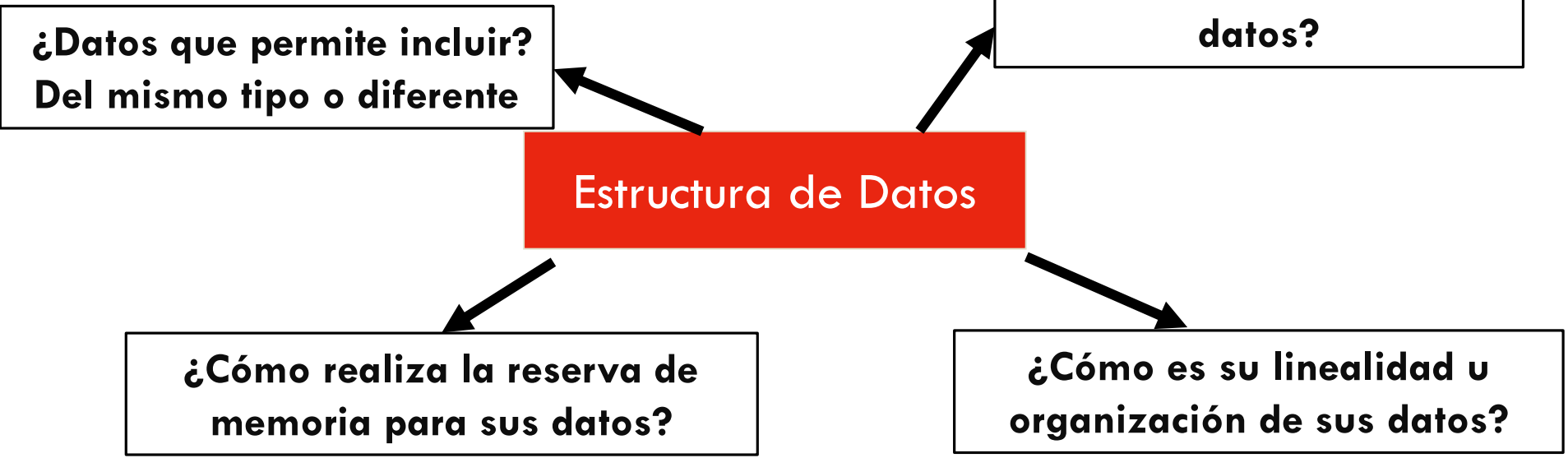
Ejemplos

1. Representar los datos de empleados de una empresa. Se identifica a través del nombre, el número de documento, la fecha de nacimiento, el número de legajo, el sexo, el sueldo, la antigüedad, etc.

2. Representar una pila de libros.

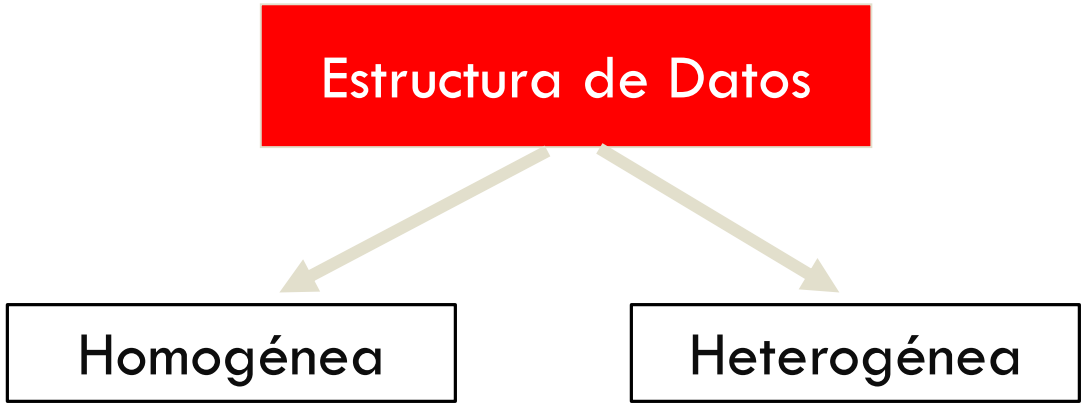
Clasificación

Clasificación



Clasificación

De acuerdo a los tipos de datos que se pueden almacenar en la estructura:

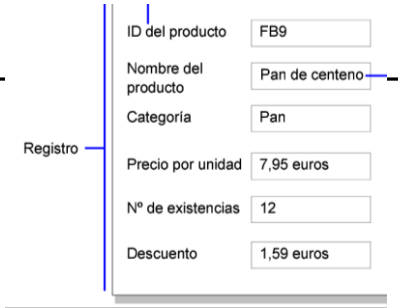


Clasificación

Las estructuras de datos pueden clasificarse, de acuerdo al tipo de datos que la componen en homogéneas y heterogéneas.

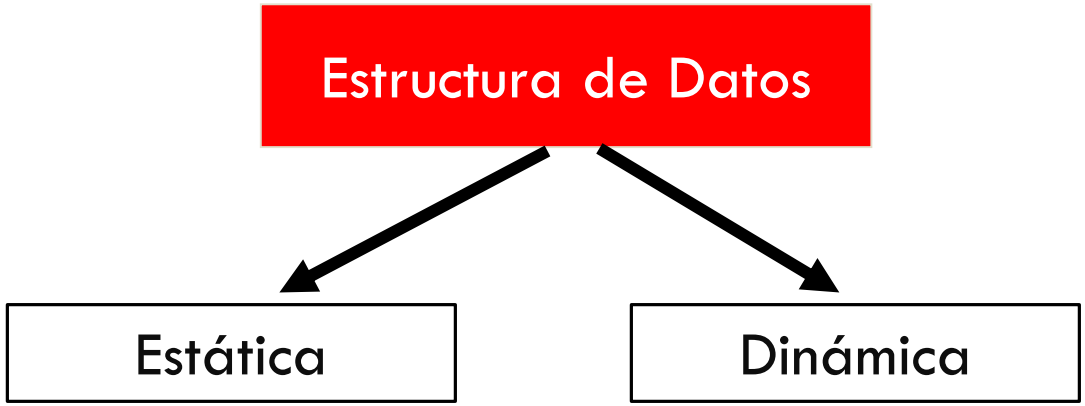
Una estructura de datos se dice homogénea si los datos que la componen son todos del mismo tipo.

Una estructura de datos se dice heterogénea si los datos que la componen son de distinto tipo.



Clasificación

De acuerdo a la ocupación de memoria las estructuras pueden ser:



Clasificación

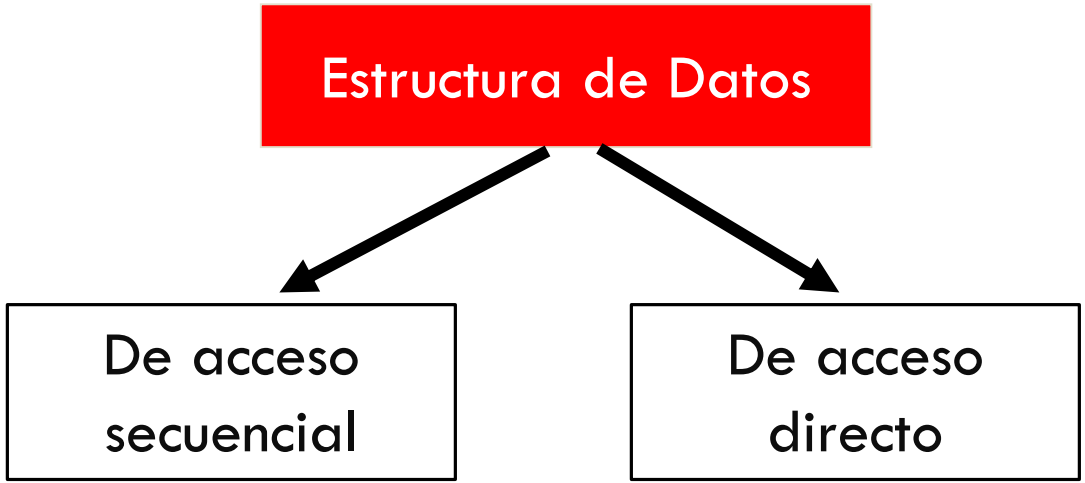
Las estructuras de datos pueden clasificarse, de acuerdo a la ocupación de memoria en estáticas y dinámicas.

Una estructura de datos se dice estática si la cantidad de elementos que contiene es fija, es decir que la cantidad de memoria que ocupa no varía durante la ejecución del programa.

Una estructura de datos se dice dinámica si la cantidad de elementos que contiene es variable, y por lo tanto la cantidad de memoria ocupada puede cambiar durante la ejecución de un programa.

Clasificación

De acuerdo al acceso a sus elementos, las estructuras pueden ser:



Clasificación

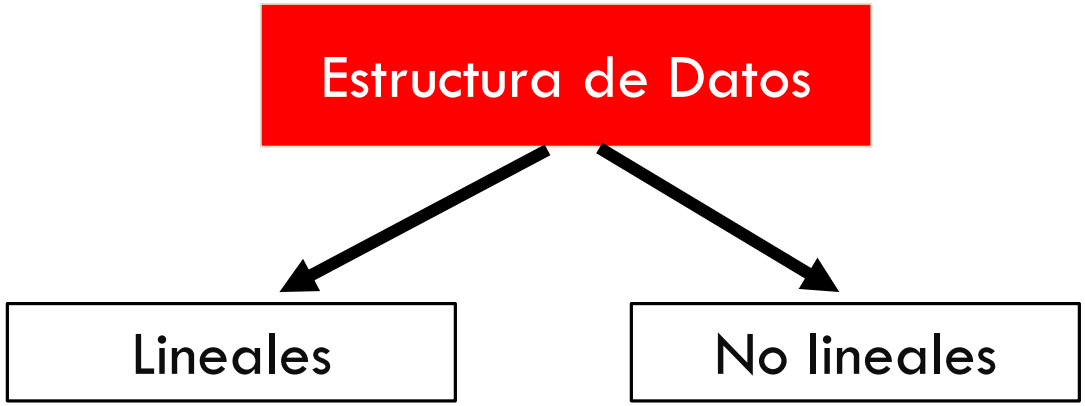
Las estructuras de datos pueden clasificarse, de acuerdo a como se accede a sus elementos, de acceso directo o secuencial.

Una estructura de datos se dice de acceso secuencial, si para acceder a un elemento particular se debe respetar un orden predeterminado, por ejemplo, pasando por todos los elementos que le preceden, por ese orden.

Una estructura de datos se dice de acceso directo, si se puede acceder a un elemento particular, directamente, sin necesidad de pasar por los anteriores a él, por ejemplo, referenciando una posición.

Clasificación

De acuerdo a la organización de sus datos, las estructuras pueden ser:



Clasificación

Las estructuras de datos pueden clasificarse, de acuerdo a su linealidad en Lineales y No Lineales.

Una estructura de datos se dice lineal cuando está formada por ninguno, uno o varios elementos que guardan una relación de adyacencia ordenada donde a cada elemento le sigue uno y le precede uno, solamente.

Una estructura de datos se dice No lineal si para un elemento dado pueden existir 0, 1 ó mas elementos que le suceden y 0, 1 ó mas elementos que le preceden.