



Conceptos de Algoritmos Datos y Programas

Conceptos de Algoritmos Datos y Programas



Lograr que el alumno cuando termine el curso, posea conocimientos, métodos y herramientas para resolver distintos problemas con la computadora logrando:

- Analizar problemas, poniendo énfasis en la **modelización, abstracción** y en la **modularización** de los mismos.
- Obtener una expresión sintética, precisa y **documentada** de los problemas y su solución.
- Analizar y expresar correctamente algoritmos, orientando los mismos a la **resolución de las partes** (módulos) en que se descomponen los problemas.
- Introducir las nociones de **estructuras de datos, tipos de datos y abstracción de datos**.

CADP – TEMAS



- Análisis de problemas
- Definiciones Fundamentales
- Modelos + Datos = programa

CADP – DEFINICIONES



Informática

Es la **ciencia** que estudia el análisis y **resolución de problemas** utilizando **computadoras**.

CADP – DEFINICIONES

Es la **ciencia** que estudia el análisis y **resolución de problemas** utilizando **computadoras**.



Ciencia Se relaciona con una metodología fundamentada y racional para el estudio y resolución de los problemas. En este sentido la Informática se vincula especialmente con la Matemática y la Ingeniería



Resolución

Se puede utilizar las herramientas informáticas en aplicaciones de áreas muy diferentes tales como biología, comercio, control industrial, administración, robótica, educación, arquitectura, etc.



Computadora

Máquina digital y sincrónica, con cierta capacidad de cálculo numérico y lógico controlado por un programa almacenado y con probabilidad de comunicación con el mundo exterior. Ayuda al hombre a realizar tareas repetitivas en menor tiempo y con mayor exactitud. No razona ni crea soluciones, sino que ejecuta una serie de órdenes que le proporciona el ser humano

CADP – DEFINICIONES



Informática - Objetivo

Resolver problemas del mundo real utilizando una computadora (utilizando un software)

CADP – PARADIGMAS DE PROGRAMACION



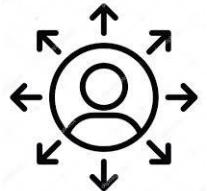
**Imperativo -
procedural**

En general, los lenguajes de programación pueden ser clasificados a partir del modelo que siguen para DEFINIR y OPERAR información. Este aspecto permite jerarquizarlos según el paradigma que siguen.

CADP – COMO VAMOS A TRABAJAR



Poseer un problema



Modelizar el problema



Modularizar la solución

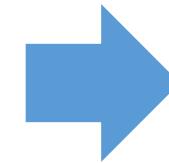


Realizar el programa



Utilizar la computadora

CADP – COMO VAMOS A TRABAJAR



Poseer un problema



En el laboratorio se compraron dos robots lego y ahora se quiere que los robots implementen los algoritmos que los alumnos desarrollan con el entorno CMRE.

Cómo es la comunicación?

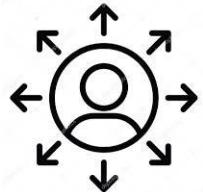
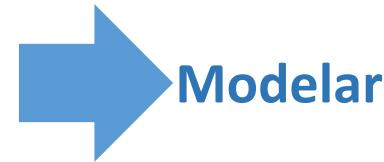
Cómo representamos la ciudad?

Qué consideraciones hay que tener?

Cuándo aparece la computadora?

Lenguaje?

CADP – COMO VAMOS A TRABAJAR



El modelo define los mecanismos de interacción y sus condiciones. Establece el efecto sobre la máquina y el usuario. Indica los Informes necesarios.

Pensar que acciones se van a permitir y que implica cada acción permitida

Acciones permitidas para el robot.

Condiciones para realizarlas.

Requerimientos de la máquina para cada acción.

Efecto de las acciones del robot en la máquina.

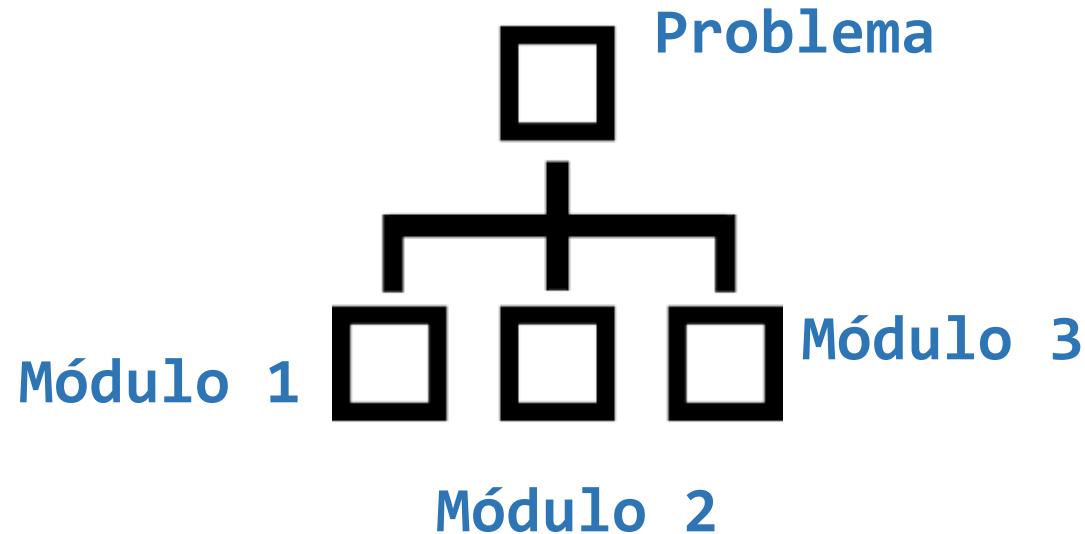
CADP – COMO VAMOS A TRABAJAR



Modularizar

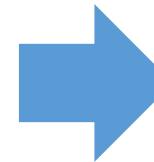


A partir del modelo es necesario encontrar la forma de descomponer en partes (módulos) para obtener una solución.



La descomposición funcional de todas las acciones que propone el modelo nos ayudará a reducir la complejidad, a distribuir el trabajo y en el futuro a reutilizar los módulos.

CADP – COMO VAMOS A TRABAJAR



Realizar el programa



Una vez que se tiene la descomposición en funciones / procesos o módulos, debemos diseñar su implementación: esto requiere escribir el programa y elegir los datos a representar.

PROGRAMA = Algoritmo

+

Datos

Las instrucciones (que también se han denominado acciones) representan las operaciones que ejecutará la computadora al interpretar el programa. Un conjunto de instrucciones forma un algoritmo.

Los datos son los valores de información de los que se necesita disponer y en ocasiones transformar para ejecutar la función del programa.



ALGORITMO

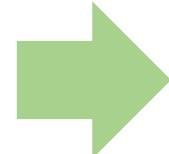
Especificación rigurosa de la secuencia de pasos (instrucciones) a realizar sobre un autómata para alcanzar un resultado deseado en un tiempo finito.



Alcanzar el resultado en tiempo finito: suponemos que un algoritmo comienza y termina. Está implícito que el número de instrucciones debe ser también finito.

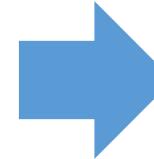


Especificación rigurosa: que debemos expresar un algoritmo en forma clara y unívoca.



Si el **autómata** es una computadora, tendremos que escribir el algoritmo en un lenguaje “entendible” y ejecutable por la máquina.

CADP – COMO VAMOS A TRABAJAR

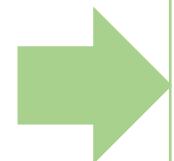


Realizar el programa



DATO

Es una representación de un objeto del mundo real mediante la cual podemos modelizar aspectos del problema que se quiere resolver con un programa sobre una computadora. Puede ser constante o variable.



Los pasos que realiza el robot en un recorrido

Las flores que hay en una esquina

Una imagen

El peso de una persona, el nombre, el dni, etc.

Qué características
tiene el programa?

CADP – COMO VAMOS A TRABAJAR

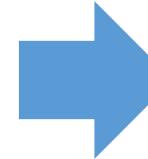


Para el Desarrollador

- **Operatividad:** El programa debe realizar la función para la que fue concebido.
- **Legibilidad :** El código fuente de un programa debe ser fácil de leer y entender. Esto obliga a acompañar a las instrucciones con comentarios adecuados.
- **Organización:** El código de un programa debe estar descompuesto en módulos que cumplan las subfunciones del sistema.
- **Documentados:** Todo el proceso de análisis y diseño del problema y su solución debe estar documentado mediante texto y/o gráficos para favorecer la comprensión, la modificación y la adaptación a nuevas funciones.

Para la Computadora

- Debe contener instrucciones válidas.
- Deben terminar.
- No deben utilizar recursos inexistentes.



COMPUTADORA

Máquina capaz de aceptar datos de entrada, ejecutar con ellos cálculos aritméticos y lógicos y dar información de salida (resultados), bajo control de un programa previamente almacenado en su memoria.

En cuál de todas las etapas apareció el lenguaje?



Poseer un problema



Modelizar el problema



Modularizar la solución



Realizar el programa



Utilizar la computadora



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS DE LA CLASE DE HOY



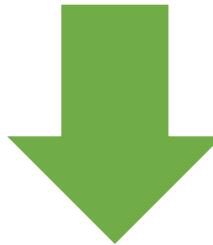
- Tipos de datos
- Tipos de datos: entero real y lógico
- Operaciones

CADP – TIPO DE DATOS

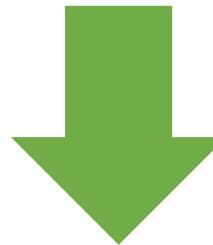


DATO

Es una clase de objetos de datos ligados a un conjunto de operaciones para crearlos y manipularlos.



Tienen un rango de
valores posibles

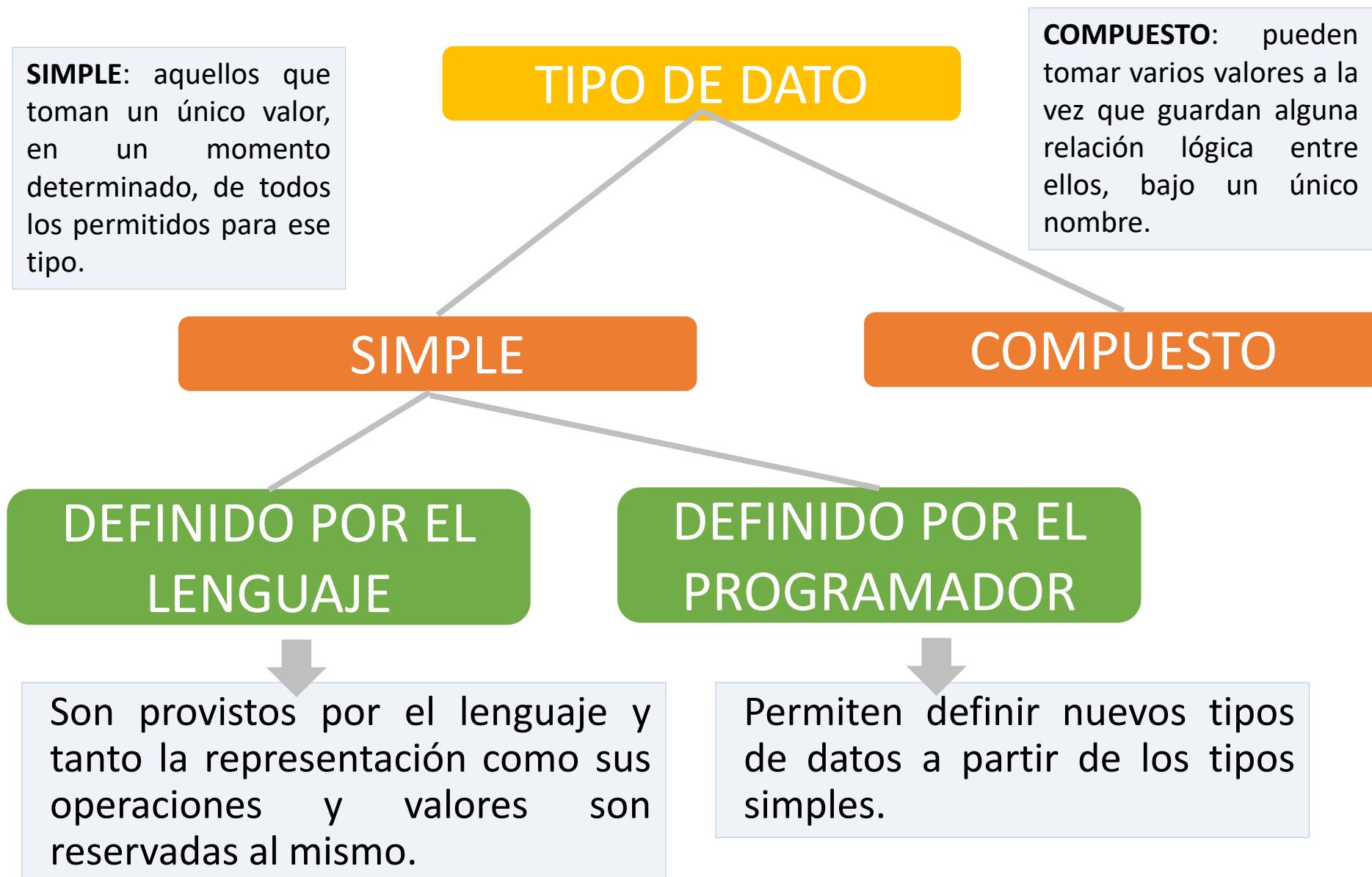


Tienen un conjunto de
operaciones permitidas



Tienen una
representación
interna

CADP – TIPO DE DATOS - Clasificación





DATO NUMERICO

Representa el conjunto de números que se pueden necesitar.
Estos números pueden ser enteros o reales.

Tipo de datos
entero

- Es un tipo de dato simple, ordinal
- Los valores son de la forma
-10 , 200, -3000, 2560
- Al tener una representación interna, tienen un número mínimo y uno máximo



Operaciones

Operadores Matemáticos

- +
- -
- *
- /

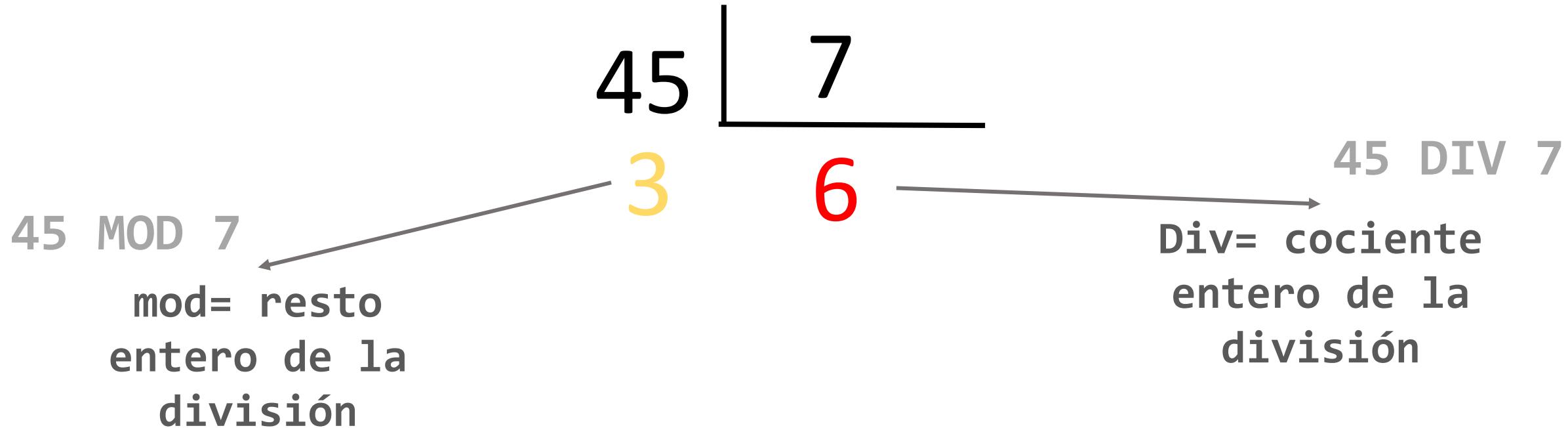
Operadores Lógicos

- <
- >
- =
- <=
- =>

Operadores Enteros

- Mod
- Div

Qué es div y mod?



Supongamos a,b,c,d variables enteras

a:= 22

b:= 6

c:= a DIV b; 3

d:= a MOD c; 4





DATO NUMERICO

**Representa el conjunto de números que se pueden necesitar.
Estos números pueden ser enteros o reales.**

Tipo de datos
real

- Es un tipo de dato simple, permiten representar números con decimales
- Los valores son de la forma
-10 , 200, -3000, 2560, 11.5, -22.89
- Al tener una representación interna, tienen un número mínimo y uno máximo



Operaciones

Operadores Matemáticos

- +
- -
- *
- /

Operadores Lógicos

- <
- >
- =
- <=
- =>

CADP – TIPO DE DATOS EJERCICIO



Pensar tres casos en los cuales para representar los datos utilizaría un número real y no un entero.

Pensar un caso en el cual para representar el dato utilizaría un número entero y no un real.

CADP – Tipos de Datos DATO NUMERICO - PARENTESIS



Las expresiones que tienen dos o más operandos requieren reglas matemáticas que permitan determinar el orden de las operaciones.

El orden de precedencia para la resolución, ya conocido, es:

1. operadores *, /, div y mod
2. operadores +, -

En caso que el orden de precedencia natural deba ser alterado, es posible la utilización de paréntesis dentro de la expresión.

Supongamos a,b,c,d variables enteras



```
a:= 22  b:= 6  
c := a DIV b +3 * 2;  
d:= a DIV (b + 3 * 2);
```



DATO LOGICO

Permite representar datos que pueden tomar dos valores verdadero o falso.

Tipo de datos
lógico

Es un tipo de dato simple, ordinal
Los valores son de la forma
 true = verdadero
 false = falso



Operaciones

Operadores Lógicos

- and (conjunción)
- or (disyunción)
- not (negación)



Operaciones

V	V	V
F	F	F
V	F	F
F	V	F

Conjunción

V	V	V
F	F	F
V	F	V
F	V	V

Disyunción

V	F
F	V

Negación



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



- Tipos de datos
- Tipo de dato char, string
- Operaciones
- Variables y constantes



DATO CARACTER

Representa un conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato de tipo carácter contiene solo un carácter.

Tipo de datos
caracter

Es un tipo de dato simple, ordinal

Los valores son de la forma
a B ! \$ L 4

CADP – TIPOS DE DATOS DATO CARACTER



Operaciones

Operadores Lógicos

- <, <=
- >, =>
- =
- <>

! "#\$%& ' ()*+, - ./
0123456789: ;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
'`abcdefghijklmn_o
pqrsuvwxyz{|}~
Hay 95 caracteres ASCII imprimibles,
numerados del 32 al 126.

La Tabla ASCII contiene todos los caracteres y el orden entre los mismos.
<http://ascii.cl/es/>



DATO STRING

Representa un conjunto finito de caracteres. Como máximo representa 256 caracteres. En general se utilizan para representar nombres.

Tipo de datos
string

Es un tipo de dato compuesto

Los valores son de la forma
casa /erML

CADP – TIPOS DE DATOS DATO STRING



Operaciones

Operadores Lógicos

- <, <=
- >, =>
- =
- <>
- Existen otras pero no las veremos

CADP – TIPOS DE DATOS DATO STRING



Supongamos que se tiene x,y variables string

$x := 'ABC'$ $y := 'aBC'$

$x = "abc"$?

$x = y$?

$x > y$?

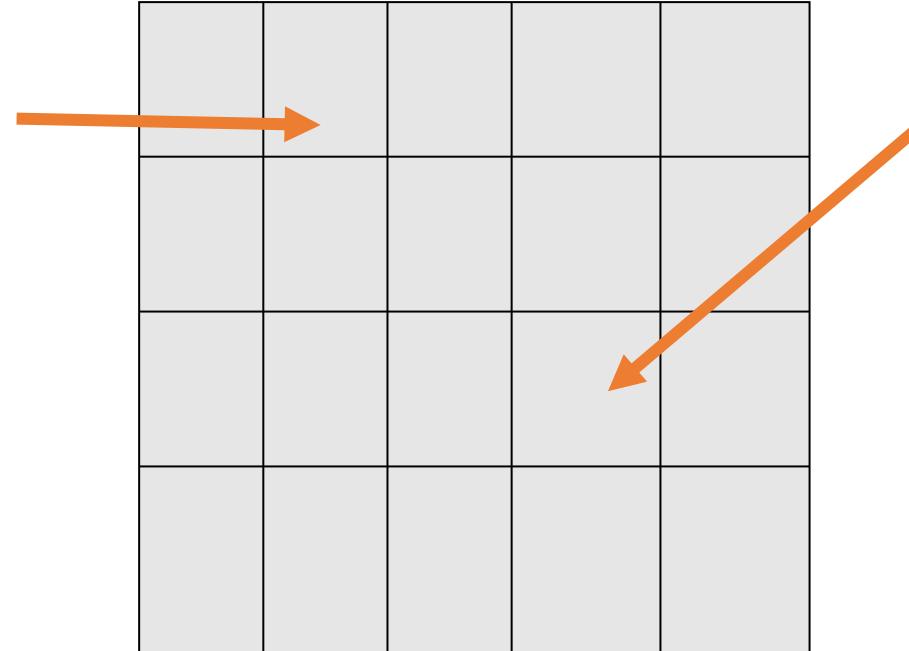
qué resultado
dan estas
operaciones?

CADP – VARIABLES - CONSTANTES



**Variable
NOMBRE**

Referencia una
zona de
memoria



**Constante
NOMBRE**

Referencia una
zona de
memoria

En qué se
diferencian?

CADP – VARIABLES - CONSTANTES



Variables

Es una zona de memoria cuyo contenido va a ser alguno de los tipos mencionados anteriormente. La dirección inicial de esta zona se asocia con el nombre de la variable.

Puede cambiar su valor durante el programa.



Constantes

Es una zona de memoria cuyo contenido va a ser alguno de los tipos mencionados anteriormente. La dirección inicial de esta zona se asocia con el nombre de la variable.

NO puede cambiar su valor durante el programa.

CADP – VARIABLES - CONSTANTES



Qué información para resolver los ejercicios del curso hubiera sido útil declararla como constante?.

CADP – TIPOS DE DATOS RECORDAR



Los diferentes tipos de datos deben especificarse y a esta especificación dentro de un programa se la conoce como declaración.

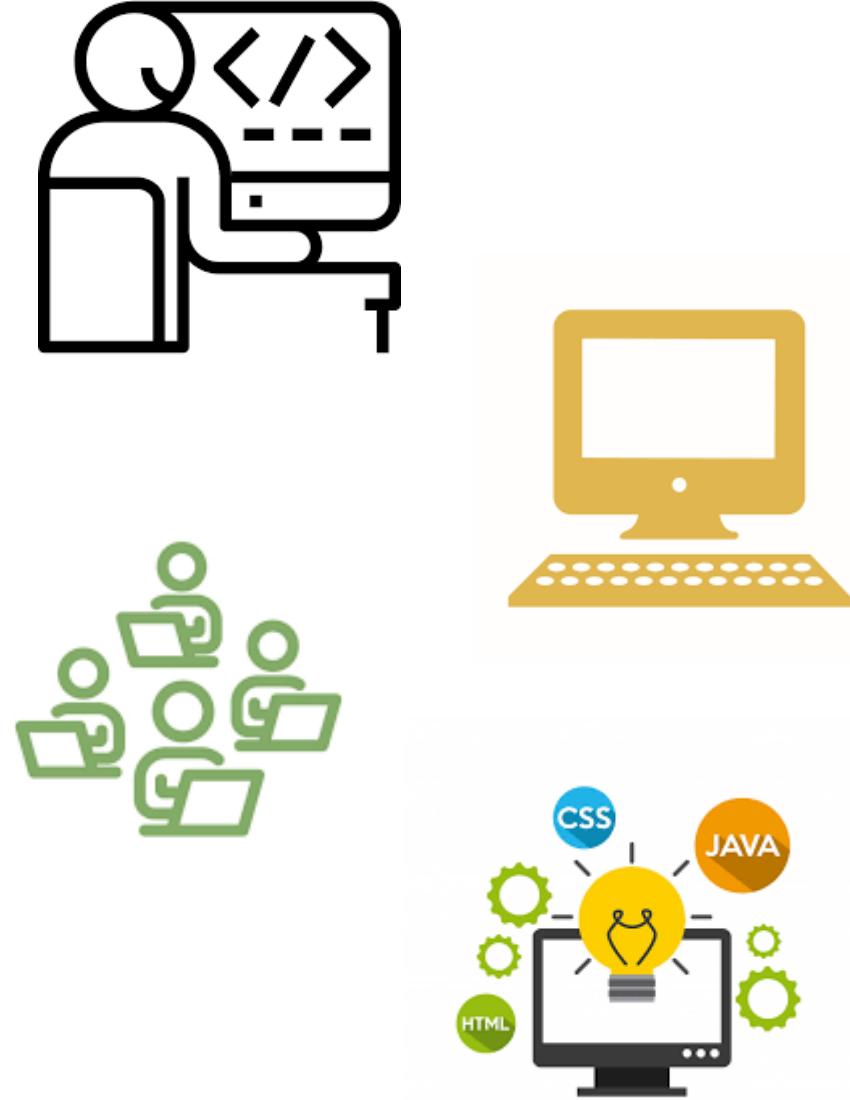
Una vez declarado un tipo podemos asociar al mismo variables, es decir nombres simbólicos que pueden tomar los valores característicos del tipo.

Algunos lenguajes exigen que se especifique a qué tipo pertenece cada una de las variables. Verifican que el tipo de los datos asignados a esa variable se correspondan con su definición. Esta clase de lenguajes se denomina fuertemente tipados (**strongly typed**).

Otra clase de lenguajes, que verifica el tipo de las variables según su nombre, se denomina auto tipados (**self typed**).

Otra clase de lenguajes, que verifica el tipo de las variables según su nombre, se denomina auto tipados (**self typed**).

Existe una tercera clase de lenguajes que permiten que una variable tome valores de distinto tipo durante la ejecución de un programa. Esta se denomina dinámicamente tipados (**dynamically typed**).



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



- Estructura de un programa
- Pre y post condiciones
- Operaciones de read y write

CADP – HASTA AHORA LOS PROGRAMAS



Programa nombre

areas

Procesos

 proceso nombre

 variables

 comenzar

 fin

 variables

 comenzar

 fin

Y ahora cómo
escribimos un
programa?

CADP –PROGRAMAS AHORA



Program nombre;

Const

Constantes del programa

... . . .

módulos {luego veremos como se declaran}

Módulos del programa

Var

Variables del programa

begin

...

Cuerpo del programa

end.

CADP –PROGRAMAS AHORA



Constantes
del programa

Variables
del
programa

Cuerpo
del
programa

Program nombre;

Const

N = 25;

pi = 3.14;

módulos {luego veremos como se declaran}

var

edad: integer;

peso: real;

letra: char;

resultado: boolean;

begin

edad:= 5;

peso:= -63.5;

edad:= edad + N;

letra:= 'A';

resultado:= letra = 'a';

end.

CADP – PRE y POST CONDICIONES

PRE CONDICON



Es la información que se conoce como verdadera antes de iniciar el programa (ó módulo).

POST CONDICON

es la información que debería ser verdadera al concluir el programa (ó módulo), si se cumplen adecuadamente los pasos especificados.

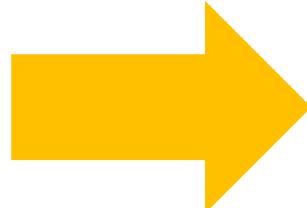
CADP – LECTURA / ESCRITURA

READ



HASTA AHORA

```
Program uno;  
  
var  
    edad: integer;  
    valor: integer;  
  
begin  
    edad:= 5;  
  
    valor:= edad + 15;  
end.
```



Program uno; *Cómo funciona el read?*
var
 edad: integer;
 valor: integer;
 suma:integer;
begin
 read (edad);
 valor:= 9;
 suma:= edad + valor;
end.



READ

Es una operación que contienen la mayoría de los lenguajes de programación. Se usa para tomar datos desde un dispositivo de entrada (por defecto desde teclado) y asignarlos a las variables correspondientes.



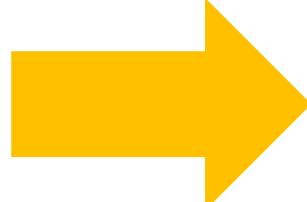
```
Program uno;  
var  
    cant: integer;  
Begin  
    read (cant);  
End.
```

El usuario ingresa un valor, y ese valor se guarda en la variable asociada a la operación read.



HASTA AHORA

```
Program uno;  
  
var  
    edad: integer;  
    valor: integer;  
  
begin  
    edad:= 5;  
  
    Informar(edad);  
end.
```



Program uno; *cómo funciona el write?*
var
 edad: integer;
 valor: integer;

begin
 read (edad);
 valor:= edad + 15;
 write (valor);
end.



WRITE

Es una operación que contienen la mayoría de los lenguajes de programación. Se usa para mostrar el contenido de una variable, por defecto en pantalla.

```
Program uno;  
var  
    cant: integer;  
Begin  
    read (cant);  
    cant:= cant + 1;  
    write (cant);  
End.
```

Variantes del
write?

El valor almacenado en la variable asociada a la operación write, se muestra en pantalla.

CADP – LECTURA / ESCRITURA

WRITE



```
Program uno;  
var  
    ...  
Begin  
    ...  
    write ("texto");      Write ("Los valores ingresados son 0")  
  
    write (variable);   Write (num);  
  
    write ("texto",variable);  Write ("El resultado es:",num);  
  
    write ("texto", resultado de una operación);  Write ("El resultado  
es:",num+4);  
End.
```



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS

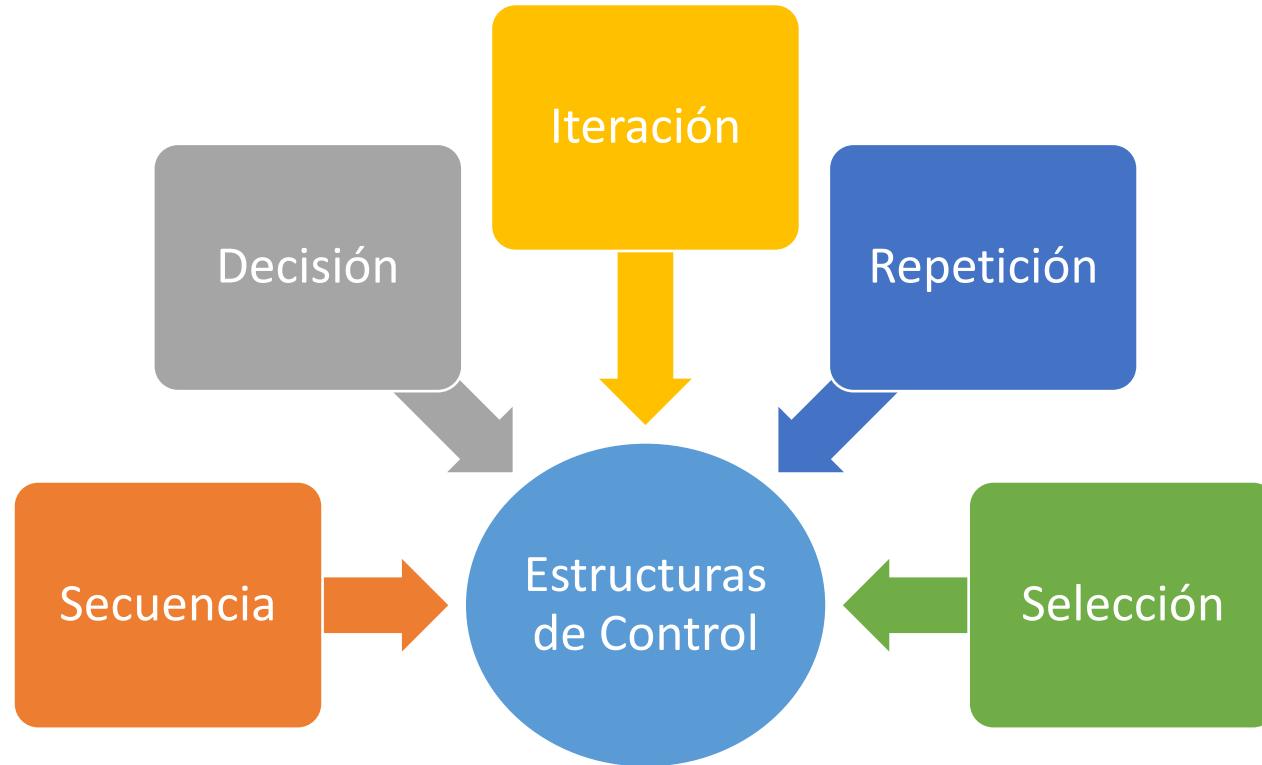
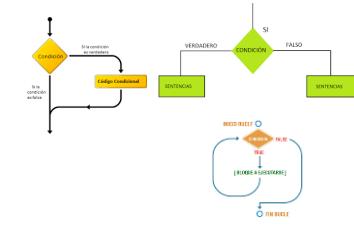


- Estructura de control
- Estructura de secuencia
- Estructura de control de decisión IF
- Estructura de control de selección CASE

CADP – ESTRUCTURAS DE CONTROL



Todos los lenguajes de programación tienen un conjunto mínimo de instrucciones que permiten especificar el control del algoritmo que se quiere implementar. Como mínimo deben contener: secuencia, decisión e iteración.



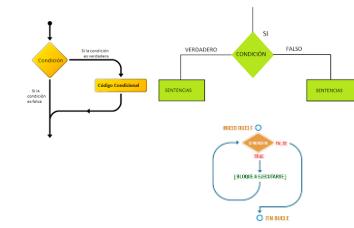
CADP – ESTRUCTURAS DE CONTROL



SECUENCIA

La estructura de control más simple, está representada por una sucesión de operaciones (por ej. asignaciones), en la que el orden de ejecución coincide con el orden físico de aparición de las instrucciones.

```
Program uno;  
...  
var  
    num:integer;  
begin  
    read (num);  
    write (num);  
end.
```

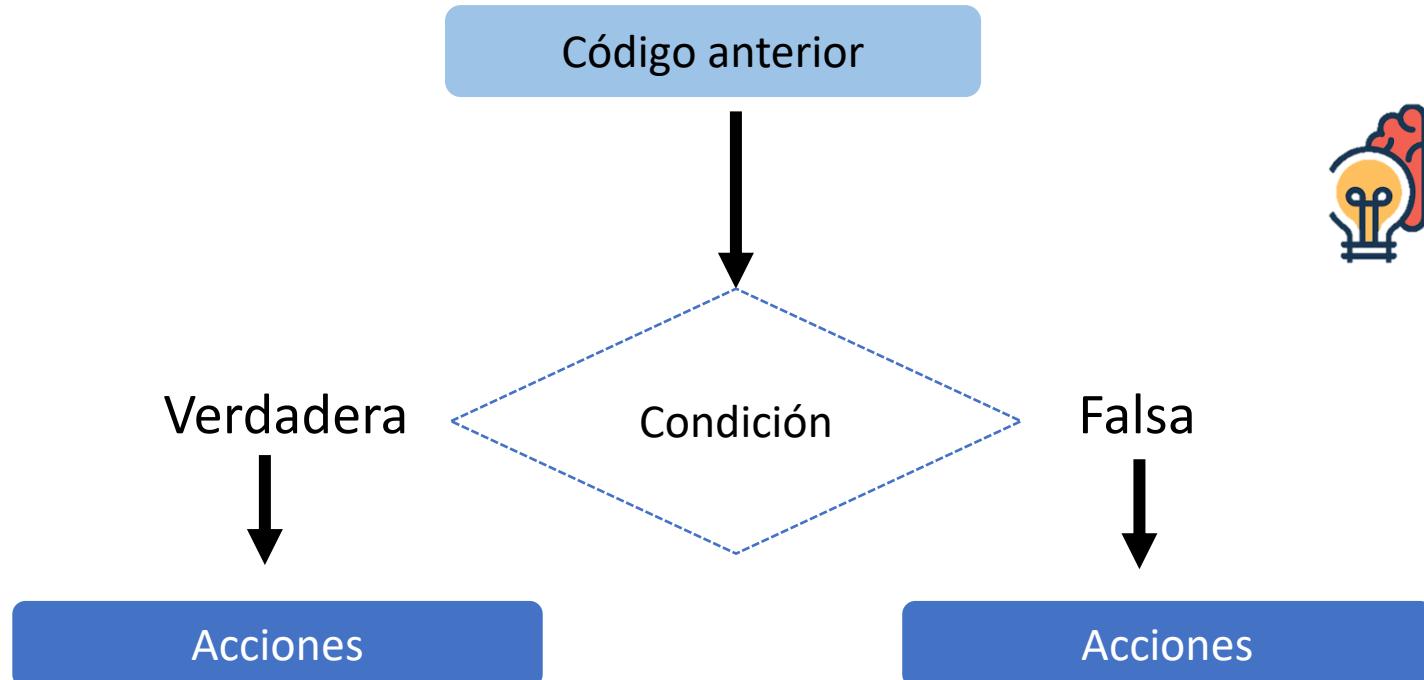


CADP – ESTRUCTURAS DE CONTROL



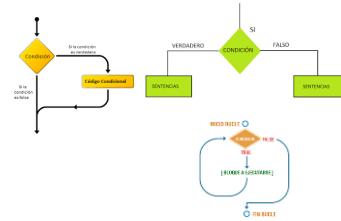
DECISION

En un algoritmo representativo de un problema real es necesario tomar decisiones en función de los datos del problema. La estructura básica de decisión entre dos alternativas es la que se representa simbólicamente:

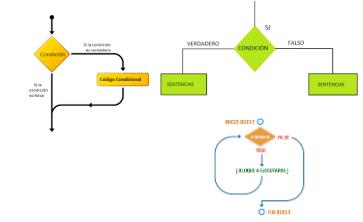


A qué estructura de control vista en el entorno del robot se parece?.

Cómo es la sintaxis?



CADP – ESTRUCTURAS DE CONTROL DECISION



```
if (condición) then  
accion;
```

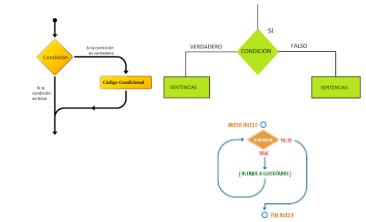
→
más de
una acción

```
if (condición) then  
acción 1  
else  
acción 2;
```

```
if (condición) then  
begin  
acción 1;  
acción 2;  
end  
else  
acción 3;
```

```
if (condición) then  
begin  
acción 1;  
acción 2;  
end;  
  
if (condición) then  
begin  
acción 1;  
acción 2;  
end  
else  
begin  
acción 3;  
acción 4;  
end;
```

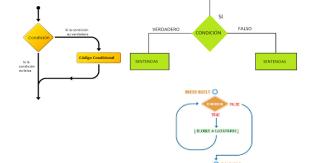
CADP – ESTRUCTURAS DE CONTROL DECISION



Realice un programa que lea dos números enteros e informe si la suma de los mismos es mayor a 20.

- Cómo leo un número
- Cómo veo si la suma es > 20
- Cómo muestro el resultado

CADP – ESTRUCTURAS DE CONTROL DECISION

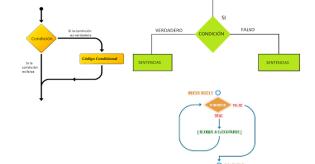


Realice un programa que lea dos números enteros e informe si la suma de los mismos es mayor a 20.

```
Program uno;  
var  
    num1,num2,suma:integer;  
  
begin  
    read (num1);  
    read (num2);  
    suma:= num1 + num2;  
    if (suma > 20)  
    then  
        write ("La suma supera 20")  
    else  
        write ("La suma NO supera 20");  
end.
```

otra
forma?

CADP – ESTRUCTURAS DE CONTROL DECISION

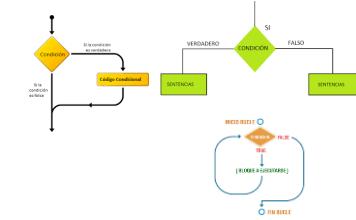


Realice un programa que lea dos números enteros e informe si la suma de los mismos es mayor a 20.

```
Program uno;
var
    num1,num2:integer;

begin
    read (num1);
    read (num2);
    if ((num1+num2) > 20)
    then
        write ("La suma supera 20")
    else
        write ("La suma NO supera 20");
end.
```

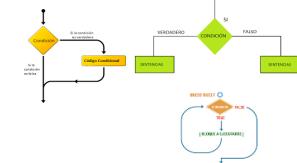
CADP – ESTRUCTURAS DE CONTROL DECISION



Realice un programa que lea un número (suponga > 0) y asigne el valor 10 a una variable si el número es menor a 10; asigne 50 a la misma variable si el número es mayor a 10 pero menor que 50; y 100 si el número es mayor a 50.

- Cómo leo un número
- Cómo verifico en qué rango está
- Cómo muestro el resultado

CADP – ESTRUCTURAS DE CONTROL DECISION



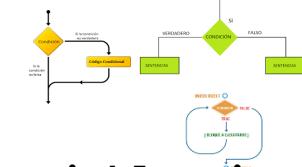
Realice un programa que lea un número (suponga > 0) y asigne el valor 10 a una variable si el número es menor a 10; asigne 50 a la misma variable si el número es mayor a 10 pero menor que 50; y 100 si el número es mayor a 50.

```
Program uno;
var
    num1,resultado:integer;

begin
    read (num1);
    if (num1 <= 10) then resultado:= 10;
    if (num1 > 10) and (num1 <= 50) then resultado:= 50;
    if (num1 > 50) then resultado:= 100;
    write (resultado);
end.
```

Es la mejor solución?

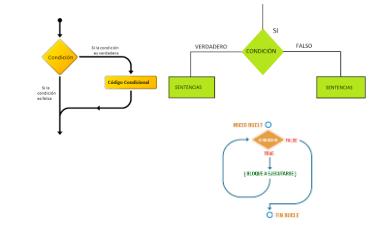
CADP – ESTRUCTURAS DE CONTROL DECISION



Realice un programa que lea un número (suponga > 0) y asigne el valor 10 a una variable si el número es menor a 10; asigne 50 a la misma variable si el número es mayor a 10 pero menor que 50; y 100 si el número es mayor a 50.

```
Program uno;  
var  
    num1,resultado:integer;  
  
begin  
    read (num1);  
    if (num1 <= 10) then resultado:= 10  
    else  
        if (num1 > 10) and (num1 <= 50) then resultado:= 50  
        else  
            resultado:= 100;  
    write (resultado);  
end.
```

CADP – ESTRUCTURAS DE CONTROL SELECCION



Realizar un programa que lea un carácter y al finalizar informe si se leyó un carácter mayúscula, minúscula, dígito, y ó especiales ha leído.



Qué tipo de datos leo?

'A'

'0'

'B'

'x'

'!'

'\$'

'='



informa

Mayúscula

Dígito

Mayúscula

Minúscula

Especial

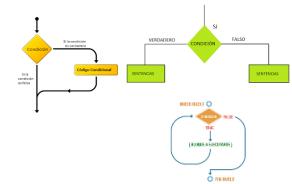
Especial

Especial



Cómo identifico que carácter es?

CADP – ESTRUCTURAS DE CONTROL SELECCION



```
Program uno;  
var  
    car:char;  
  
begin  
    read (car);
```

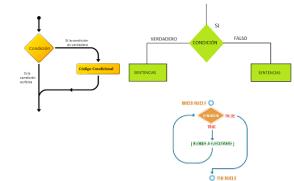
Si sólo existe el if

```
if (car = 'a') or (car = 'b')... or (car = 'z') then  
    write ("minúscula")  
if (car = 'A') or (car = 'B')... or (car = 'Z') then  
    write ("mayúscula");  
if (car = '0') or (car = '1')... or (car = '9') then  
    write ("numero");  
if (car = '@') or (car = '!')... or (car = '*') then  
    write ("especial");
```

End.

Se puede mejorar?

CADP – ESTRUCTURAS DE CONTROL SELECCION



Program uno;

var

 car:char;

begin

 read (car);

if (car = 'a') or (car = 'b')... or (car = 'z') **then**

 write ("minúscula");

else

if (car = 'A') or (car = 'B')... or (car = 'Z') **then**

 write ("mayúscula");

else

if (car = '0') or (car = '1')... or (car = '9') **then**

 write ("digito");

else if (car = '@') or (car = '!')... or (car = '*')

then

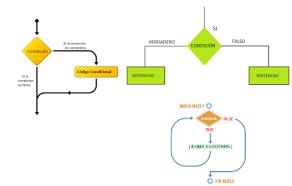
 write ("especial");

End.

Si sólo existe el if

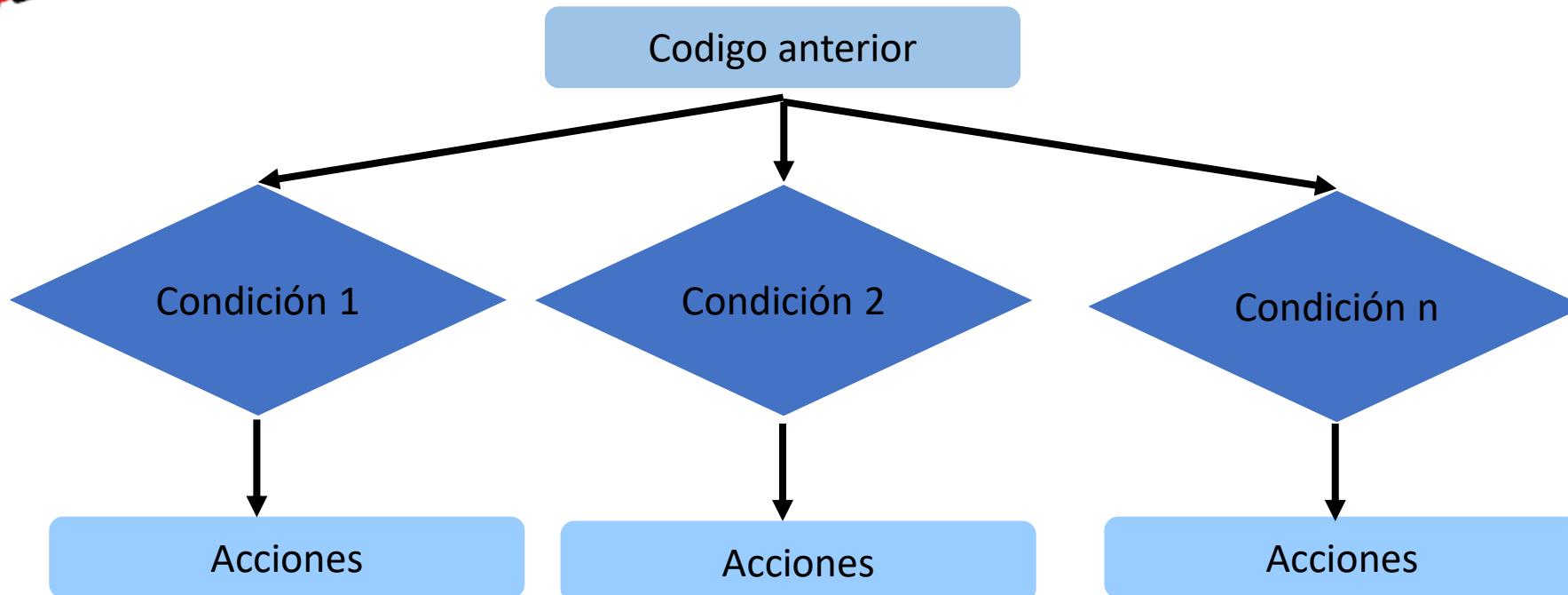
Se puede mejorar?

CADP – ESTRUCTURAS DE CONTROL



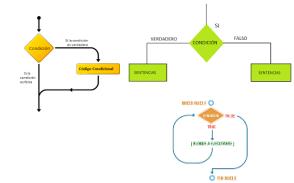
SELECCION

Permite realizar distintas acciones dependiendo del valor de una variable de tipo ordinal.



Cómo es la
sintaxis?

CADP – ESTRUCTURAS DE CONTROL SELECCION



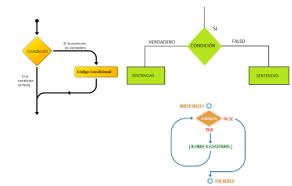
```
case (variable) of  
    condicion1: accion1;  
    condición 2: acción2;  
    ....  
    condición n: acción n;  
end;
```



más de una acción

```
case (variable) of  
    condicion1: accion1;  
    condición 2: begin  
        acción2;  
        accion3;  
    end;  
    ....  
    condición n: accion;  
end;
```

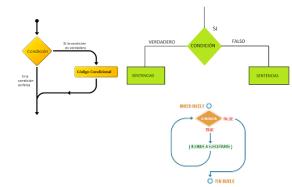
CADP – ESTRUCTURAS DE CONTROL SELECCION



```
Program uno;  
var  
    car:char;  
begin  
    read (car);  
    case car of  
        car = 'a': write("minúscula");  
        ...  
        car = 'z': write("minúscula");  
  
        car = 'A': write("mayúscula");  
        ...  
        car = 'Z': write("mayúscula");  
  
        car = '0': write("dígito");  
        ...  
        car = '9': write("dígito");  
        else write("especial");  
    end;  
End.
```

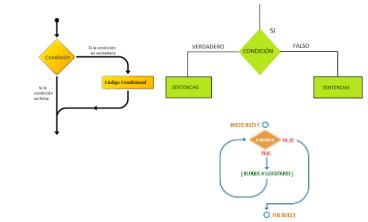
Se puede mejorar?

CADP – ESTRUCTURAS DE CONTROL SELECCION



```
Program uno;  
var  
    car:char;  
begin  
    read (car);  
    case car of  
        'a'.. 'z': write ("minúscula");  
        'A'.. 'Z': ("mayúscula");  
        '0'.. '9': ("dígito");  
        else ("especial");  
    end;  
End.
```

CADP – ESTRUCTURAS DE CONTROL SELECCION



- La variable del case debe ser de tipo ordinal
- Las opciones deben ser disjuntas

CADP – Estructuras de control



Problema: se leen valores de alturas de personas, hasta leer la altura 1.59. Informar la cantidad de personas que miden entre 1.00 y 1.30; la cantidad de personas que miden entre 1.31 y 1.50; la cantidad de personas que miden entre 1.51 y 1.89 y las que miden más de 1.89

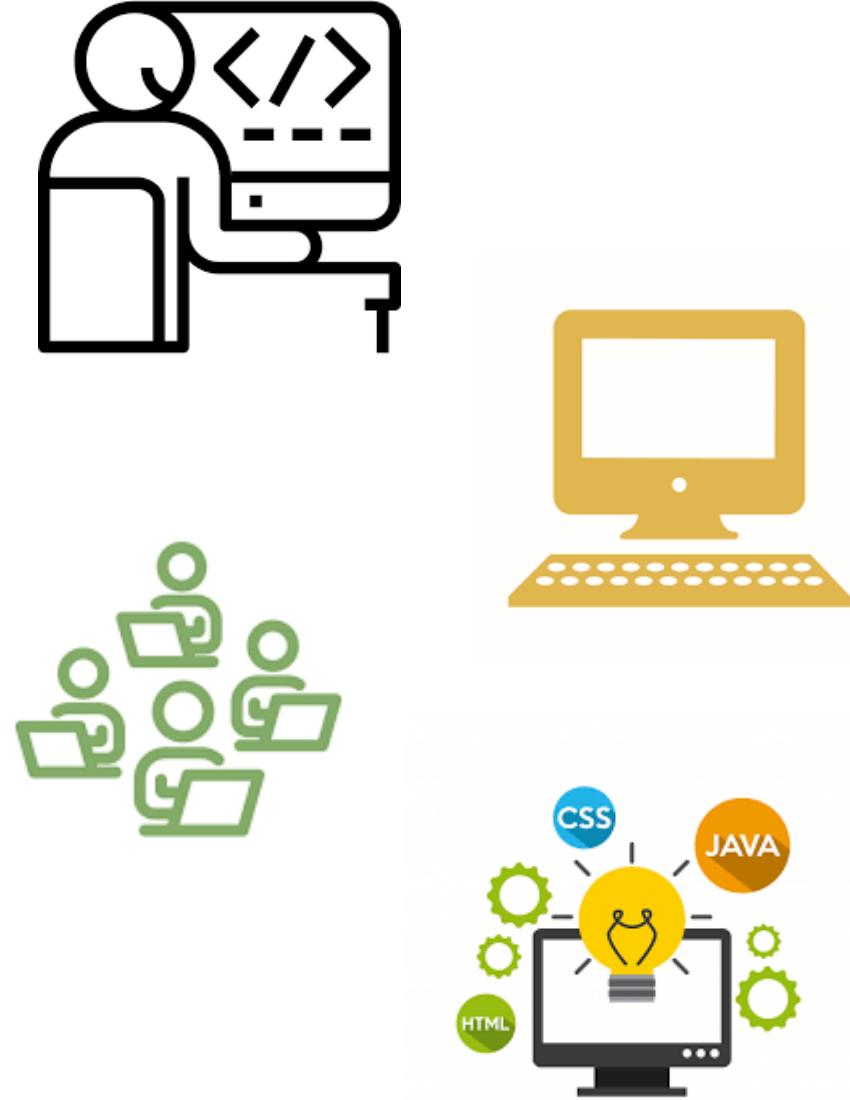
El alumno 1: utiliza una variable real para leer las alturas y cuatro contadores para contar la cantidad de personas en cada rango.

Además utiliza un while como estructura de control principal y adentro utiliza un case que incluye los rangos de alturas para saber cual contador sumar. Al final informa los valores de los contadores.

El alumno 2: utiliza una variable real para leer las alturas y cuatro contadores para contar la cantidad de personas en cada rango.

Además utiliza un while como estructura de control principal y adentro utiliza un if con else para saber cual contador sumar. Al final informa los valores de los contadores.

Ambas soluciones son
correctas?



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS

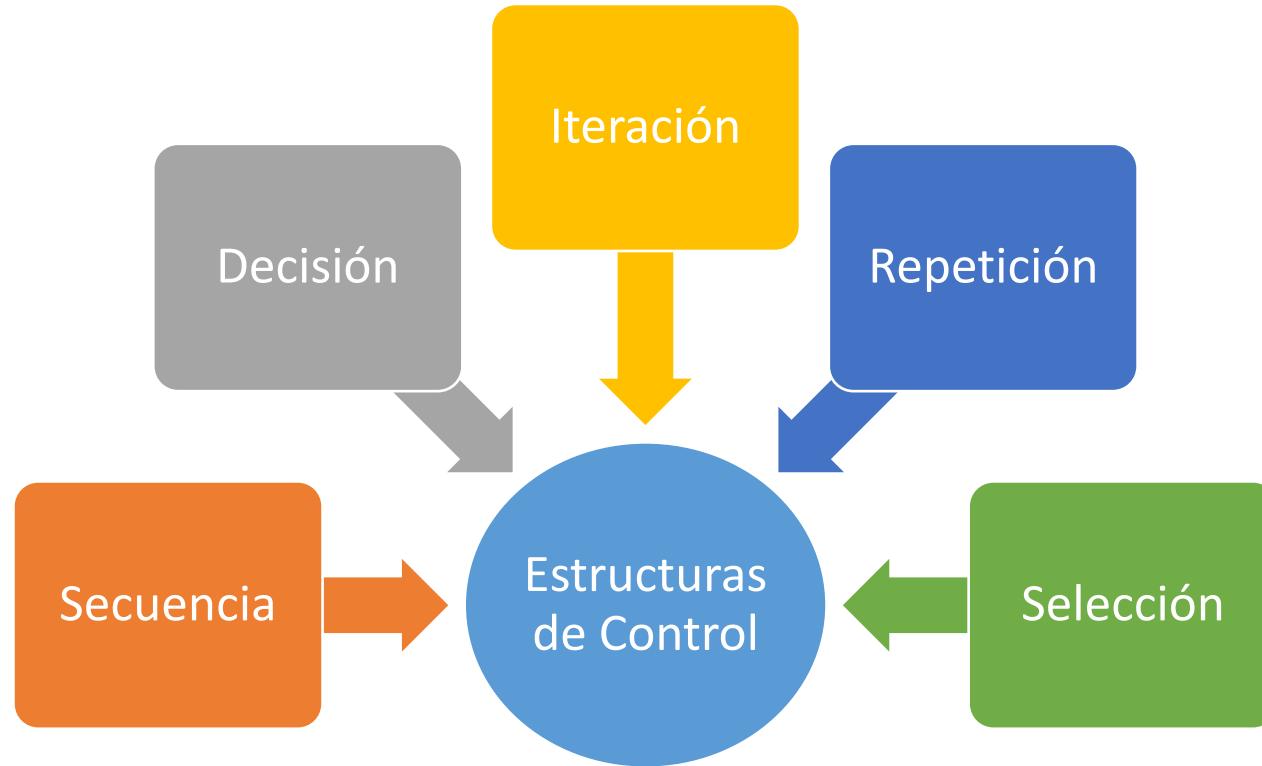
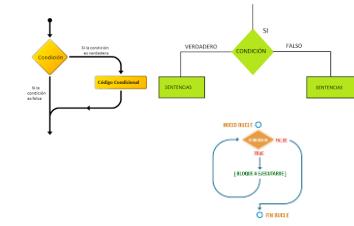


- Estructura de control
- Estructura de iteración
- Estructura de control WHILE y REPEAT UNTIL

CADP – ESTRUCTURAS DE CONTROL



Todos los lenguajes de programación tienen un conjunto mínimo de instrucciones que permiten especificar el control del algoritmo que se quiere implementar. Como mínimo deben contener: secuencia, decisión e iteración.



CADP – ESTRUCTURAS DE CONTROL



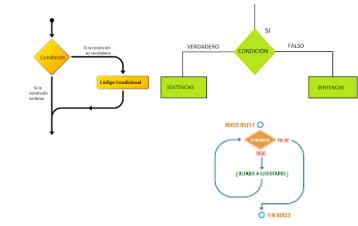
ITERACION

Puede ocurrir que se desee ejecutar un bloque de instrucciones desconociendo el número exacto de veces que se ejecutan.

Para estos casos existen en la mayoría de los lenguajes de programación estructurada las estructuras de control iterativas condicionales.

Como su nombre lo indica las acciones se ejecutan dependiendo de la evaluación de la condición.

Estas estructuras se clasifican en **pre-condicionales** y **post-condicionales**.



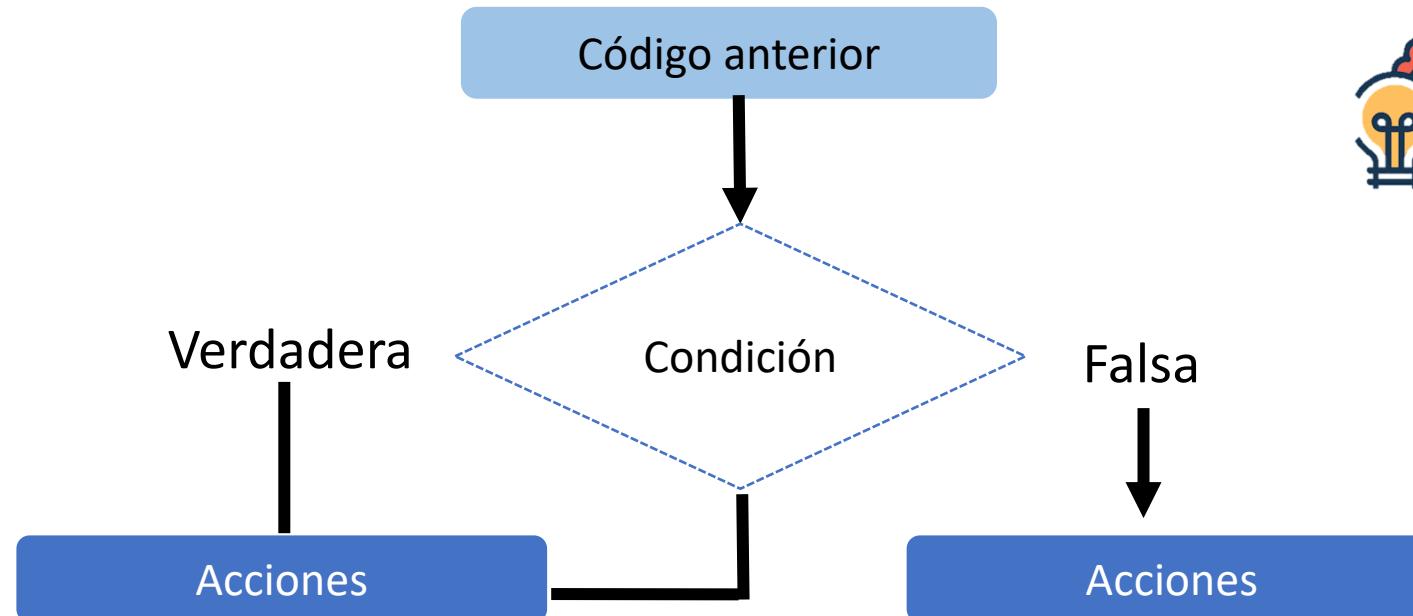
CADP – ESTRUCTURAS DE CONTROL



ITERACION - PRECONDICIONAL

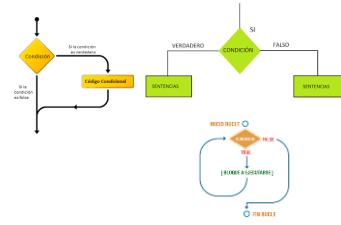
Evalúan la condición y si es verdadera se ejecuta el bloque de acciones. Dicho bloque se pueda ejecutar 0, 1 ó más veces.

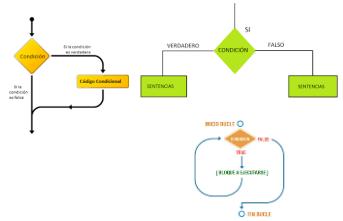
Importante: el valor inicial de la condición debe ser conocido o evaluable antes de la evaluación de la condición.



A qué estructura de control vista en el entorno del robot se parece?.

Cómo es la sintaxis?





ITERACION - PRECONDICIONAL

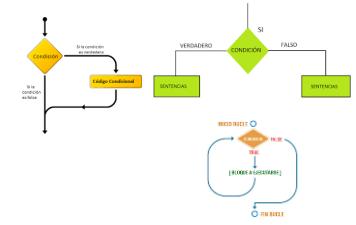
```
while (condición) do  
accion;
```

más de una acción



```
while(condición) do  
begin  
acción 1;  
acción 2;  
end;
```

CADP – ESTRUCTURAS DE CONTROL ITERACION



Realizar un programa que lea códigos de productos hasta leer un código igual a 30. Al finalizar informe la cantidad de productos con código par.

- Cómo leo un código
- Cómo veo si es par
- Cuál es la condición de fin
- Cómo muestro el resultado

35
70
32
5
30



2
informa

CADP – ESTRUCTURAS DE CONTROL ITERACION

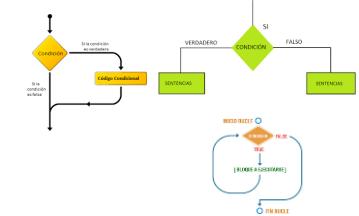


```
Program uno;  
var  
    resto,prod:integer;  
    total:integer;  
  
begin  
    total:=0;  
    while (prod <> 30)do  
        begin  
            read(prod);  
            resto:= prod MOD 2;  
            if (resto = 0)then  
                total:= total + 1;  
            end;  
            write (total);  
end.
```

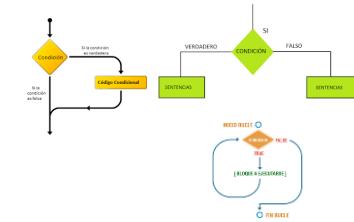
Cuál es el error?

```
Program dos;  
var  
    prod,resto:integer;  
    total:integer;  
  
begin  
    total:=0;  
    read (prod);  
    while (prod <> 30)do  
        begin  
            resto:= prod MOD 2;  
            if (resto = 0)then  
                total:= total + 1;  
            read (prod);  
        end;  
        write (total);  
end.
```

Otra alternativa?



CADP – ESTRUCTURAS DE CONTROL ITERACION



```
Program dos;
var
    prod:integer;
    total:integer;

begin
    total:=0;
    read (prod);
    while (prod <> 30)do
        begin
            if (prod MOD 2 = 0)then
                total:= total + 1;
            read (prod);
        end;
        write (total);
    end.
```

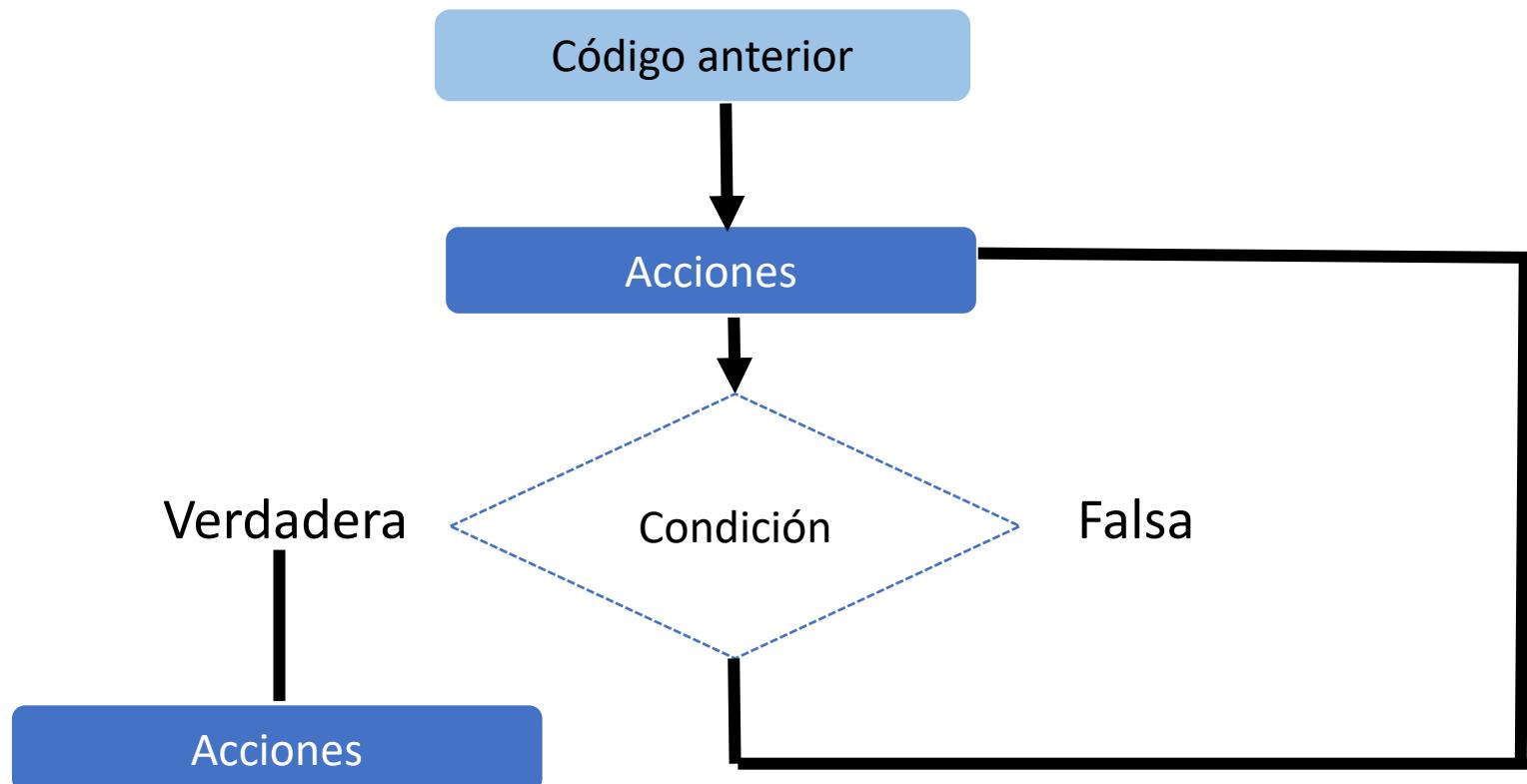
No se utiliza
la variable
resto

CADP – ESTRUCTURAS DE CONTROL



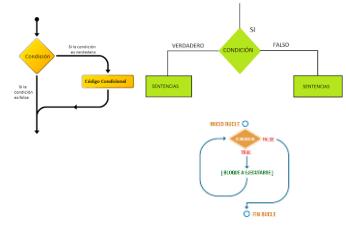
ITERACION - POSTCONDICIONAL

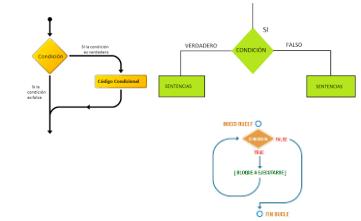
Ejecutan las acciones luego evalúan la condición y ejecutan las acciones mientras la condición es falsa. Dicho bloque se pueda ejecutar 1 ó más veces.



A qué estructura de control vista en el entorno del robot se parece?.

Cómo es la sintaxis?





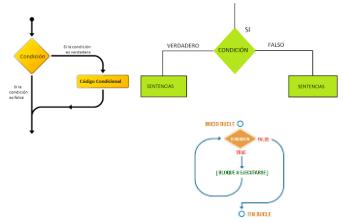
ITERACION - POSTCONDICIONAL

```
repeat  
    acción;  
until (condición);
```

→
más de una acción

```
repeat  
    acción 1;  
    acción 2;  
until (condición);
```

CADP – ESTRUCTURAS DE CONTROL ITERACION



Realizar un programa que lea códigos de productos hasta leer un código igual a 30. Al finalizar informe la cantidad de productos con código par. El último producto debe procesarse.

- Cómo leo un producto
- Cómo veo si es par
- Cuál es la condición de fin
- Cómo muestro el resultado

35
70
32
5 3
30



informa

CADP – ESTRUCTURAS DE CONTROL ITERACION

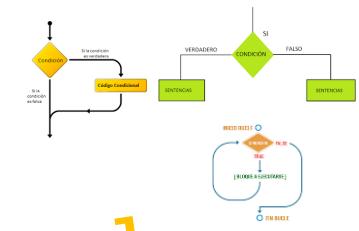


Sino existiera el
repeat until

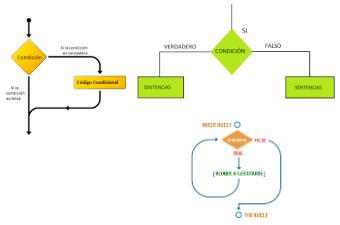
```
Program uno;
var
    prod:integer;
    total:integer;
begin
    total:=0;
    read(prod);
    while (prod <> 30)do
        begin
            if (prod MOD 2 = 0)then
                total:= total + 1;
            read(prod);
        end;
        if (prod MOD 2 = 0)then
            total:= total + 1;
        write (total);
    end.
```

Cuál es el
problema?

Todo el procesamiento
sobre la variable total se
debe repetir dentro y
fuera del while



CADP – ESTRUCTURAS DE CONTROL ITERACION



Program correcto;

var

prod:integer;

total:integer;

begin

 total:=0;

repeat

 read (prod);

 if (prod MOD 2 = 0)then

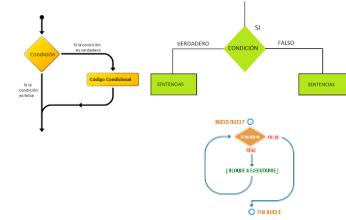
 total:= total + 1;

 until (prod = 30)

 write (total);

end.

Se ejecuta cuando
la condición es
falsa



PRE CONDICIONALES

Evalúa la condición y en caso de ser verdadera, ejecuta las acciones.

Se repite mientras la condición es verdadera.

Puede ejecutarse 0, 1 o más veces.



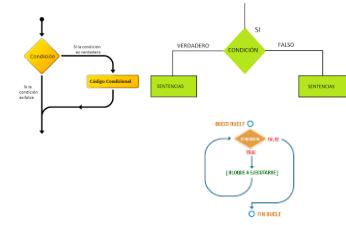
POST CONDICIONALES

Ejecuta las acciones y luego evalúa la condición.

Se repite mientras la condición es falsa.

Puede ejecutarse 1 o más veces.

CADP – ESTRUCTURAS DE CONTROL



Mirando estos enunciados que estructuras de control usarías?

- Realizar un programa que lea un número e informe si el número es par o impar
- Realizar un programa que lea un letras hasta leer la letra “@” la cual debe procesarse e informe la cantidad de letras ‘á’ leídas.
- Realizar un programa que lea un letras hasta leer la letra “@” e informe la cantidad de letras ‘á’ leídas.



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS

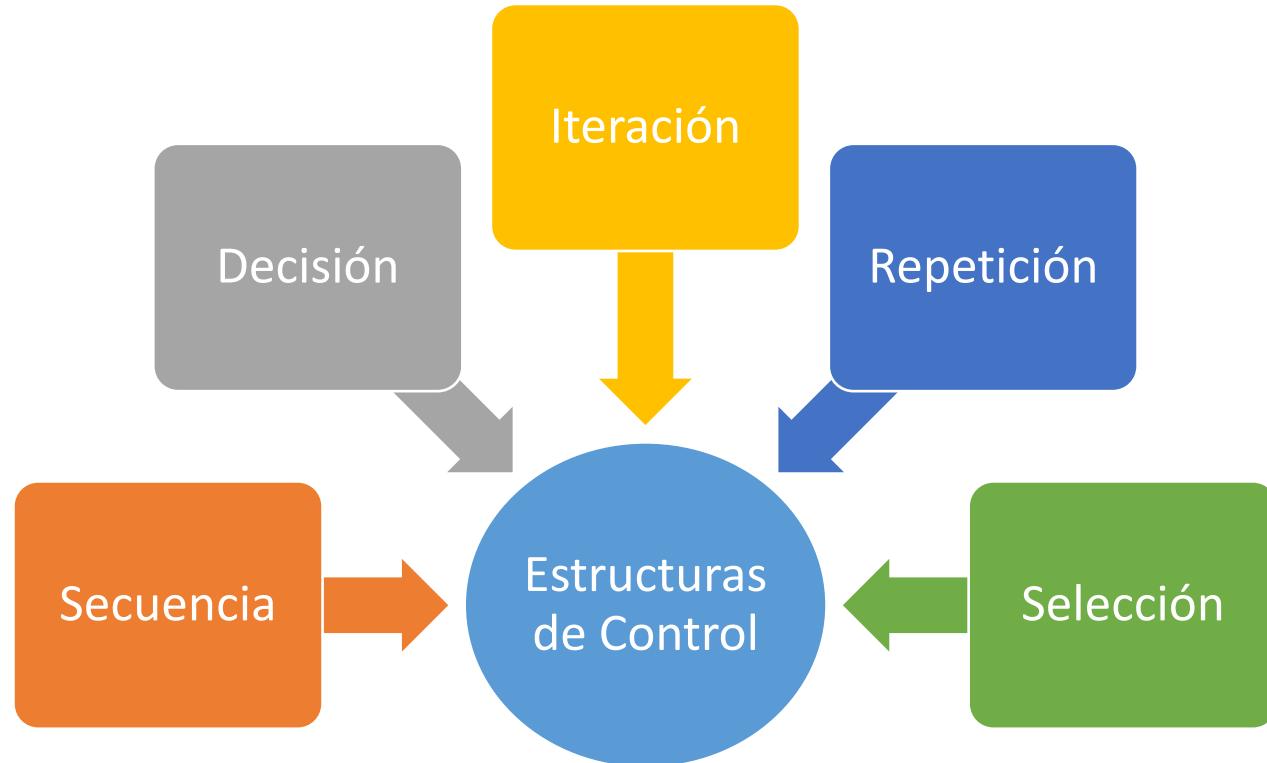
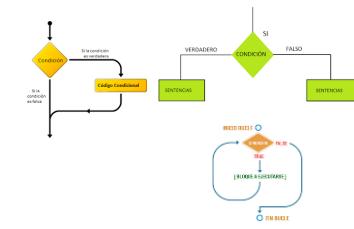


- Estructura de control
- Estructuras de control repetitivas
- Estructura de control FOR

CADP – ESTRUCTURAS DE CONTROL



Todos los lenguajes de programación tienen un conjunto mínimo de instrucciones que permiten especificar el control del algoritmo que se quiere implementar. Como mínimo deben contener: secuencia, decisión e iteración.



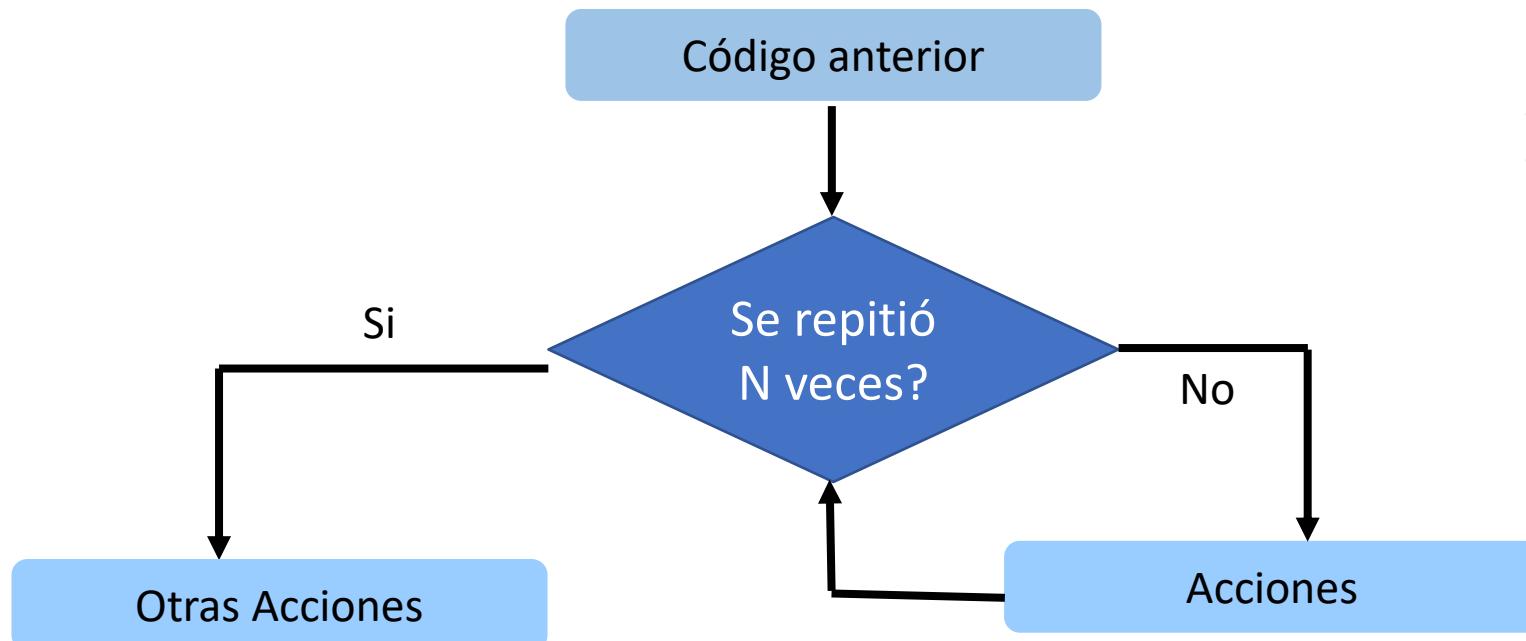
CADP – ESTRUCTURAS DE CONTROL



REPETICION

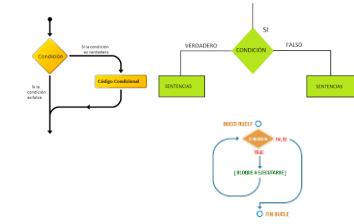
Es una extensión natural de la secuencia. Consiste en repetir N veces un bloque de acciones.

Este número de veces que se deben ejecutar las acciones es fijo y conocido de antemano

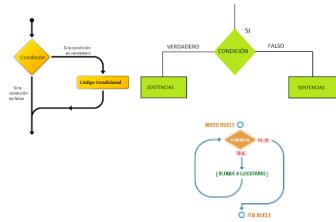


A qué estructura de control vista en el entorno del robot se parece?.

Cómo es la sintaxis?



CADP – ESTRUCTURAS DE CONTROL REPETICION



```
for indice := valor_inicial to valor_final do  
    accion 1;
```



más de una
acción

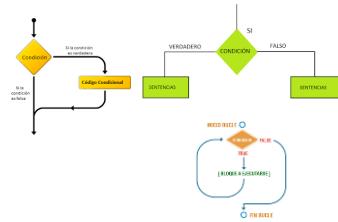
```
for indice := valor_inicial to valor_final do  
begin  
    accion 1;  
    accion 2;  
end;
```

Qué es el
índice?

Dónde se
declara?

Qué son valor
final e inicial?

CADP – ESTRUCTURAS DE CONTROL REPETICION



Ejemplo 0:

For i := 1 **to** 10 **do**
 accion;

¿De qué tipo es el índice i ?
¿qué valores toma i ?

Ejemplo 1:

For i := 'A' to 'H' do
 accion;

¿De qué tipo es el índice i ?
¿qué valores toma i ?

Ejemplo 2:

For i:= False **to** True **do**
 accion;

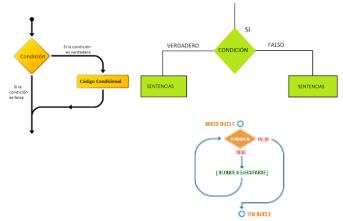
¿De qué tipo es el índice i ?
¿qué valores toma i ?

Ejemplo 3:

```
For i := 20 to 18 do  
    accion;
```

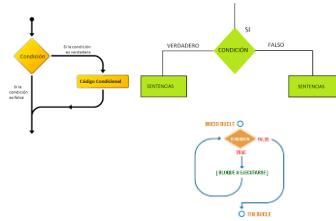
```
For índice := 20 downto 18 do
    begin
        accion;
        accion;
    end;
```

CADP – ESTRUCTURAS DE CONTROL REPETICION



- La variable índice debe ser de tipo ordinal
- La variable índice no puede modificarse dentro del lazo
- La variable índice se incrementa y decrementa automáticamente
- Cuando el for termina la variable índice no tiene valor definido.

CADP – ESTRUCTURAS DE CONTROL REPETICION



Realizar un programa que lea precios de 10 productos que vende un almacén. Al finalizar informe la suma de todos los precios leídos.

● Qué valor es el precio?

100,5

56,5

15

10

12,5

14

7,5

150,00

25,40

78,50

● Cuál es la condición de fin?

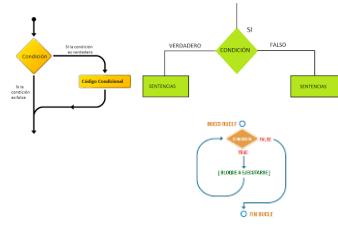


474,4

informa

● Cómo calculo la suma

CADP – ESTRUCTURAS DE CONTROL REPETICION



Program uno;

var

 precio, total:real;
 i:integer;

begin

 total := 0;

 for i:= 1 to 10 do

 begin

 read (precio);

 total:= total + precio;

 end;

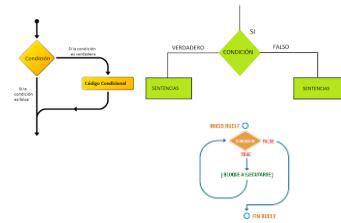
 write (“La suma de los precios de los
 productos del almacén son: ”,total);

 end.

Qué modificaría si
quiere informar al final,
también el precio del
5to producto?



CADP – ESTRUCTURAS DE CONTROL REPETICION



```
Program uno;
var
    quinto,precio,total:real;
    i:integer;
begin
    total := 0;
    for i:= 1 to 10 do
        begin
            read (precio);
            if (i=5) then
                quinto:= precio;
            total:= total + precio;
        end;
    write ("La suma de los precios de los
            productos del almacén son: ",total);
    write ("El precio del quinto producto es: ",quinto);
end.
```

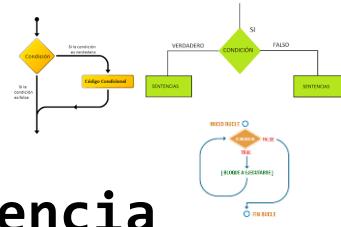
CADP – ESTRUCTURAS DE CONTROL



Qué crees que imprime el programa, si se leyera esta secuencia de números:

```
Program uno;
var
    i,num1,num2:integer;
Begin
    num2:= 0;
    for i:= 1 to 5 do
        begin
            read (num1);
            while (num1 mod 2 = 0) do
                begin
                    num2:= num2 + 1;
                    read (num1);
                end;
        end;
    write (num2);
end.
```

4
8
126
5
3
6
1
1568
6
10
7
19
22
24
3

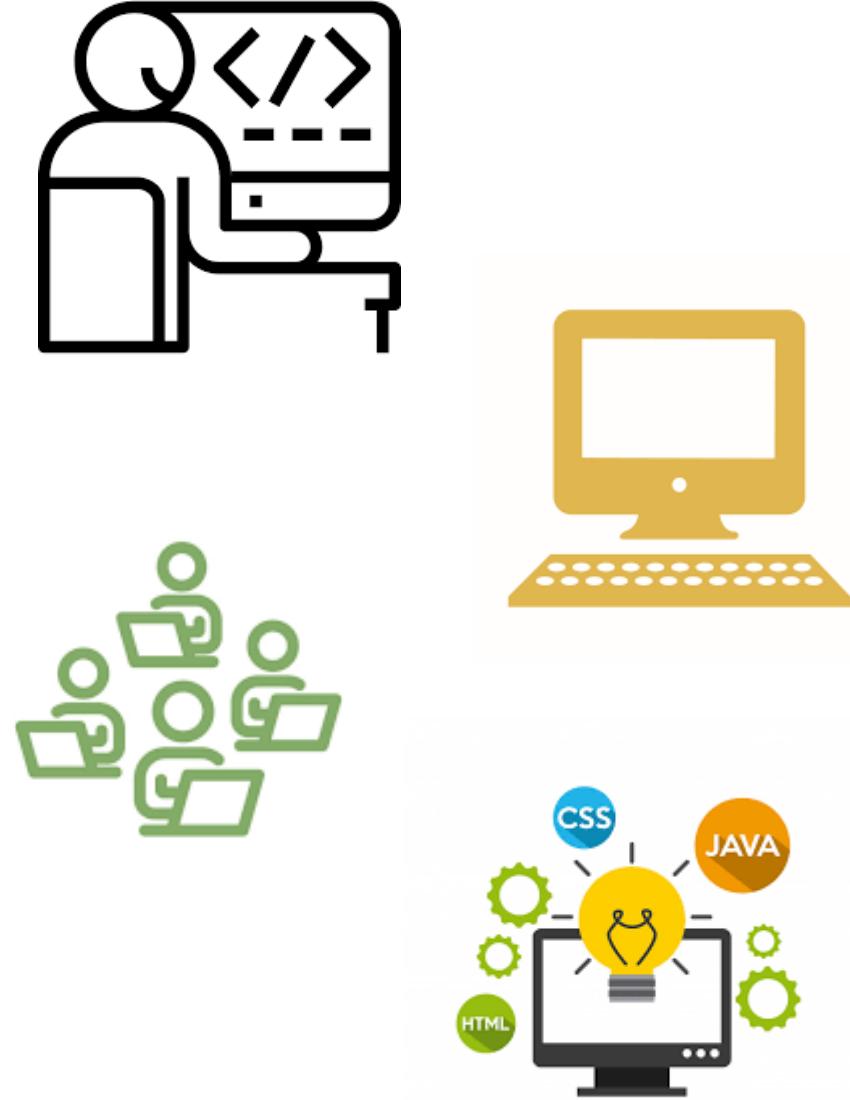


CADP – ESTRUCTURAS DE CONTROL



Qué crees que imprime el programa, si se leyera esta secuencia de números:

```
Program uno;                                4
var
    i,j,num1,num2:integer;                  7
Begin                                         126
    num2:= 0;                               5
    for i:= 1 to 3 do                      3
        begin                                 6
            read (num1);                   1
            for j:= 1 to 2 do              1568
                begin                         6
                    if (num1 mod 2 = 1) then   10
                        num2:= num2+1;
                    read (num1);           7
                end;                      19
            read (num1);                 22
        end;                      24
    end;                                  3
    write (num2);
end.
```



Conceptos de Algoritmos Datos y Programas

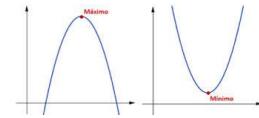
CADP – TEMAS



- Máximos y mínimos
- EJERCICIO – PREGUNTAS FINALES

CADP – MAXIMOS y MINIMOS

MAXIMOS



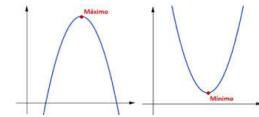
Realizar un programa que lea número de alumno y promedio hasta leer un promedio igual a 0. Al finalizar informar el promedio más alto.

- Qué tipo de valor es el promedio?
- Cuál es la condición de fin?
- Cómo verifico que es el mejor promedio?

La forma de trabajar es teniendo en cuenta la manera natural de pensarla, por cada número (o valor) a tener en cuenta se verifica si es máximo y en caso de serlo se actualiza el máximo que se tiene hasta el momento.

CADP – MAXIMOS y MINIMOS

MAXIMOS



Realizar un programa que lea número de alumno y promedio hasta leer un promedio igual a 0. Al finalizar informar el promedio más alto.

4

7.25

Es máximo?

→ Máximo
7.25

2

6.50

Es máximo?

→ Máximo
7.25

5

8.50

Es máximo?

→ Máximo
8.50

1

4.50

Es máximo?

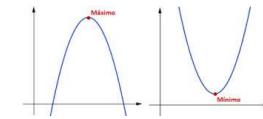
→ Máximo
8.50

0

informa
8.50

CADP – MAXIMOS y MINIMOS

MAXIMOS



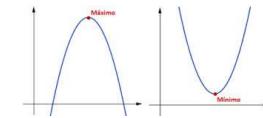
Realizar un programa que lea número de alumno y promedio hasta leer un promedio igual a 0. Al finalizar informar el promedio más alto.

Cómo verifico si es máximo?
Cómo actualizo el máximo?

```
Program uno;  
var  
    prom:real;  
    alu:integer;  
begin  
    Leo un promedio (prom y un alumno alu);  
    while (no sea la condición de fin) do  
        begin  
            verificar si es máximo  
            si (es máximo) entonces  
                actualizar máximo  
            Leo un promedio (prom y un alumno alu);  
        end;  
    write (“El mejor promedio es:”, );  
end.
```

CADP – MAXIMOS y MINIMOS

MAXIMOS



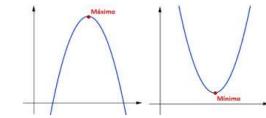
Realizar un programa que lea número de alumno y promedio hasta leer un promedio igual a 0. Al finalizar informar el promedio más alto.

Cuál es el
error?

```
Program uno;  
var  
    prom:real;alu:integer; max:real;  
begin  
    read(prom);  
    read(alu);  
    while (prom <> 0) do  
        begin  
            If (prom >= max) then  
                max:= prom;  
            read (prom);  
            read(alu);  
        end;  
    write (“El mejor promedio es:”, max );  
end.
```

CADP – MAXIMOS y MINIMOS

MAXIMOS



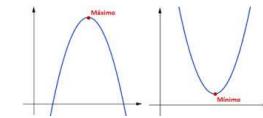
Realizar un programa que lea número de alumno y promedio hasta leer un promedio igual a 0. Al finalizar informar el promedio más alto.

Qué modiflico si
quiero saber el
número del alumno con
mejor promedio?

```
Program uno;  
var  
    prom:real;alu:integer; max:real;  
begin  
    read(prom);  
    read(alu);  
max:=-1;  
    while (prom <> 0) do  
        begin  
            If (prom >= max) then  
                max:= prom;  
            read(prom);  
            read(alu);  
        end;  
        write (“El mejor promedio es:”, max );  
    end.
```

CADP – MAXIMOS y MINIMOS

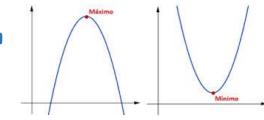
MAXIMOS



Realizar un programa que lea número de alumno y promedio hasta leer un promedio igual a 0. Al finalizar informar el promedio más alto.

```
Program uno;
var
    prom:real;alu:integer; max:real;maxalu:integer;
begin
    read(prom);
    read(alu);
    max:= -1;
    while (prom <> 0) do
        begin
            If (prom >= max) then begin
                max:= prom;
                maxalu:= alu;
            end;
            read(prom);
            read(alu);
        end;
    write (“El mejor alumno es:”, maxalu );
end.
```

CADP – MAXIMOS y MINIMOS MAXIMOS -Recordar



Utilizar una variable que representará al máximo.

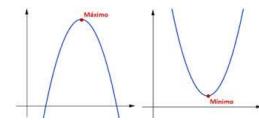


Inicializar la variable antes de comenzar la lectura de los datos. El máximo en un valor bajo.

Actualizar la variable máximo cuando corresponda

CADP – MAXIMOS y MINIMOS

MAXIMOS



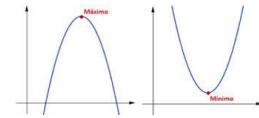
```
Program uno;
var
    cantidad,num,max_cant,max_num:integer;
begin
    max_cant:= 0;
    read(num); read(cantidad);
    while (num <> 80) do
        begin
            if (cantidad > max_cant) then begin
                max_num:= num;
                max_cant:= cantidad;
            end;
            read(num); read(cantidad);
        end;
    write ("El número con mas cantidad es :",max_num);
end.
```

Qué imprime si se lee?

15	0
23	0
8	0
80	16

CADP – MAXIMOS y MINIMOS

MINIMOS



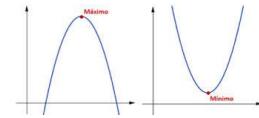
Realizar un programa que lea número de alumno y promedio hasta leer un promedio igual a 0. Al finalizar informar el promedio más bajo.

- Qué tipo de valor es el promedio?
- Cuál es la condición de fin?
- Cómo verifico que es el peor promedio?

La forma de trabajar es teniendo en cuenta la manera natural de pensarla, por cada número (o valor) a tener en cuenta se verifica si es mínimo y en caso de serlo se actualiza el mínimo que se tiene hasta el momento.

CADP – MAXIMOS y MINIMOS

MINIMOS



Realizar un programa que lea número de alumno y promedio hasta leer un promedio igual a 0. Al finalizar informar el promedio más bajo.

4

7.25

Es mínimo?

→ Mínimo
7.25

2

6.50

Es mínimo?

→ Mínimo
6.50

5

8.50

Es mínimo?

→ Mínimo
6.50

1

4.50

Es mínimo?

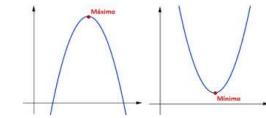
→ Mínimo
4.50

0

informa
4.50

CADP – MAXIMOS y MINIMOS

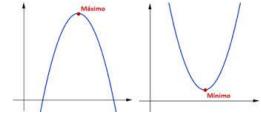
MAXIMOS



Realizar un programa que lea número de alumno y promedio hasta leer un promedio igual a 0. Al finalizar informar el promedio más bajo.

```
Program uno;  
var  
    prom:real;alu:integer; min:real;  
begin  
    read(prom);  
    read(alu);  
    min:=11;  
    while (prom <> 0) do  
        begin  
            If (prom <= min) then  
                min:= prom;  
            read(prom);  
            read(alu);  
        end;  
    write (“El mejor promedio es:”, min );  
end.
```

CADP – MAXIMOS y MINIMOS RECORDAR



Utilizar una variable que representará al mínimo.

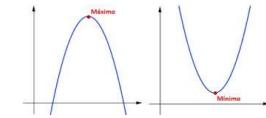


Inicializar la variable antes de comenzar la lectura de los datos. El mínimo en un valor alto.

Actualizar la variable mínimo cuando corresponda

Qué modifco si quiero saber
el promedio máximo y el
promedio mínimo?

CADP – MAXIMOS y MINIMOS AMBOS



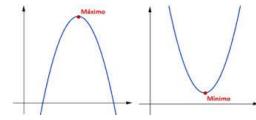
```
Program opcion1;
var
prom,max,min:integer;
begin
max:= -1;
min= 11;
read(prom);
while (prom <> 0) do
begin
  if (prom >= max) then
    max:= prom
  else
    if (prom <= min) then
      min:= prom;
  read(prom)
end;
write (max,min);
end.
```



Ambas soluciones
son correctas?

```
Program opcion2;
var
prom,max,min:integer;
begin
max:= -1;
min= 11;
read(prom);
while (prom <> 0) do
begin
  if (prom >= max) then
    max:= prom
  if (prom <= min) then
    min:= prom;
  read(prom)
end;
write (max,min);
end.
```

CADP – MAXIMOS y MINIMOS



```
Program uno;
Var
  max,valor:integer;
begin
  max:=-999;
  read(valor);
  while (valor <> 0) do
    begin
      if (valor >= max) then
        max:= valor
      read (valor);
    end;
    write (max);
end.
```

```
Program dos;
Var
  min,max,valor:integer;
begin
  min:= 9999; max:=-999;
  read(valor);
  while (valor <> 0) do
    begin
      if (valor >= max) then
        max:= valor
      else
        if (valor <= min) then
          min:= valor;
      read (valor);
    end;
    write (min,max);
end.
```

```
Program tres;
Var
  min,max,valor:integer;
begin
  min:= 9999; max:=-999;
  read(valor);
  while (valor <> 0) do
    begin
      if (valor >= max) then
        max:= valor
      if (valor <= min) then
        min:= valor;
      read (valor);
    end;
    write (min,max);
end.
```



Qué imprime cada programa si se leen los siguientes valores?

-6, 5, 31, 5, 50, -6, 50, 20, 0



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



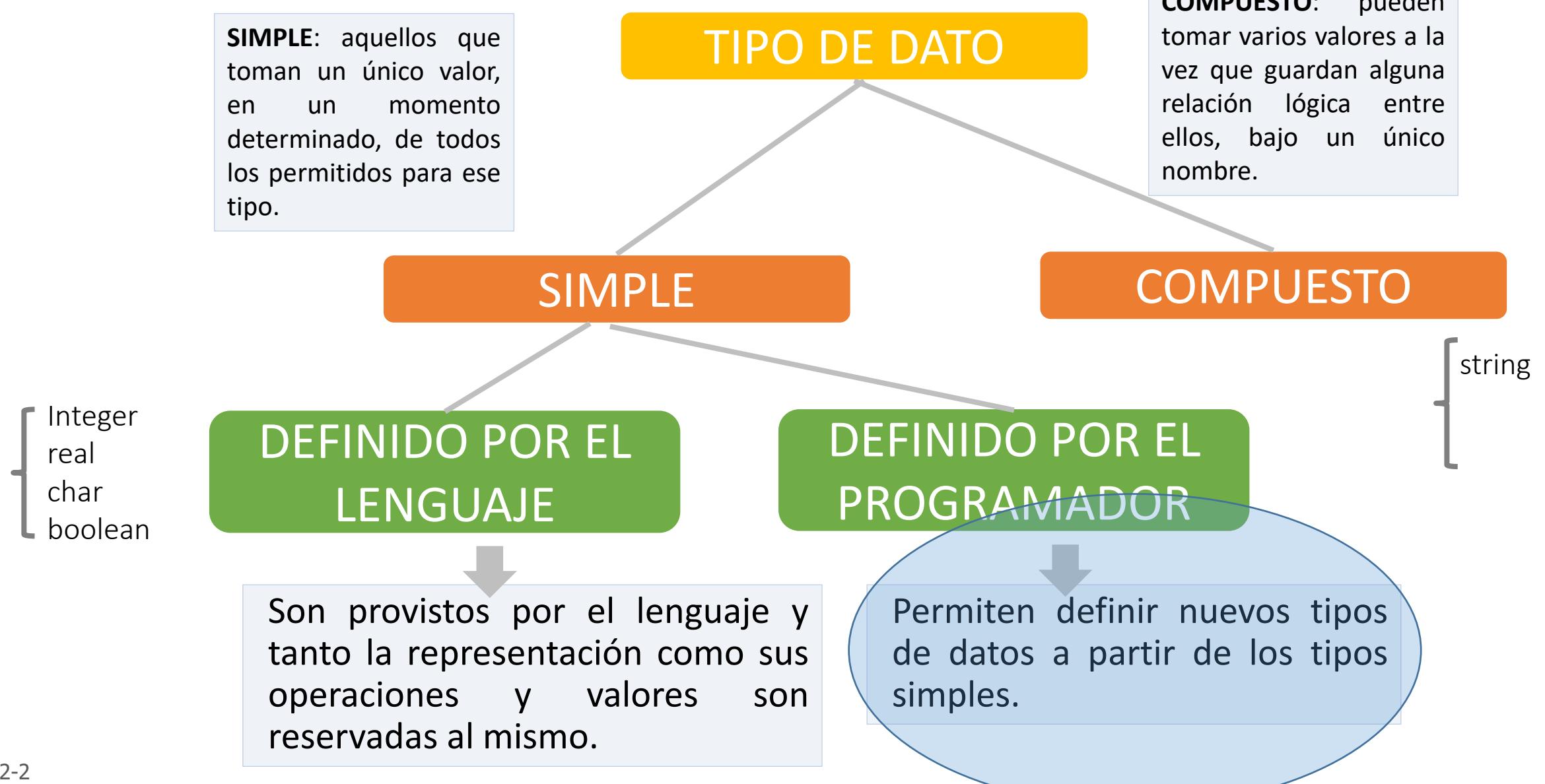
Tipos de Datos definidos por el usuario



EJERCICIO – PREGUNTAS FINALES



CADP – TIPOS DE DATOS- CLASIFICACION



CADP – TIPOS DE DATOS DEFINIDOS POR EL USUARIO



HASTA AHORA

Hemos trabajado los tipos de datos simples o compuestos que se pueden considerar estándar en la mayoría de los lenguajes de programación.



Esto significa que el conjunto de valores de ese tipo, las operaciones que se pueden efectuar y su representación están definidas y acotadas por el lenguaje.

Esta definición nos indica que podemos requerir la representación de elementos “NO de tipo estándar” que se asocien con el fenómeno real a tratar.

CADP – TIPOS DE DATOS DEFINIDOS POR EL USUARIO



Un aspecto muy importante en los lenguajes de programación es la capacidad de especificar y manejar datos no estándar, indicando valores permitidos, operaciones válidas y su representación interna.

Aumento de la riqueza expresiva del lenguaje, con mejores posibilidades de abstracción de datos.

Mayor seguridad respecto de las operaciones que se realizan sobre cada clase de datos.

**Tipos
Definidos
por el
Usuario**

Límites preestablecidos sobre los valores posibles que pueden tomar las variables que corresponden al tipo de dato.



TDU (Tipos de Datos definidos por el usuario)

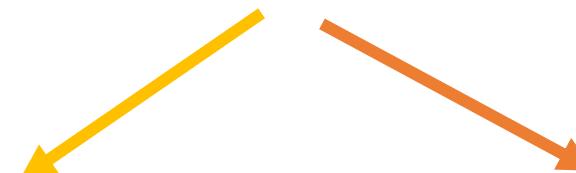
Un tipo de dato definido por el usuario es aquel que no existe en la definición del lenguaje, y el programador es el encargado de su especificación.

Type

```
identificador = tipo;
```

Un tipo estandar

Un tipo definido por el
lenguaje





CADP – TIPOS DE DATOS DEFINIDOS POR EL USUARIO

program uno;

Const

...

Type

 nuevotipo1 = tipo;

Módulos

Var

 x: identificador;

...

Begin

...

End.

CADP – TIPOS DE DATOS DEFINIDOS POR EL USUARIO



```
Program nombre;  
Const  
  N = 25;  
  pi = 3.14;
```

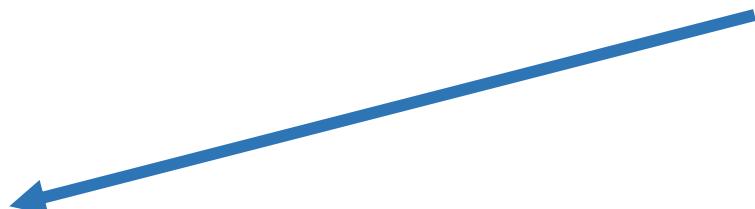
Type

```
  nuevotipo1 = integer;
```

módulos {luego veremos como se declaran}

```
var  
edad: integer;  
peso: real;  
valor: nuevotipo1;
```

```
begin
```



```
end.
```

La variable valor:

- puede tener asignado cualquier valor permitido para los enteros.
- Puede utilizar cualquier operación permitida para los enteros.
- NO puede relacionarse con ninguna otra variable que no sea de su mismo tipo

CADP – TIPOS DE DATOS DEFINIDOS POR EL USUARIO



```
Program nombre;
```

```
Const
```

```
    N = 25;
```

```
    pi = 3.14;
```

```
Type
```

```
    nuevotipo1 = integer;
```

```
var
```

```
    edad: integer;
```

```
    peso: real;
```

```
    valor: nuevotipo1;
```

```
begin
```

```
    valor:= 8;  read (valor); if (valor MOD 3 = ...) then ...
```

```
        edad:= valor;  edad:= edad + valor
```



```
end.
```



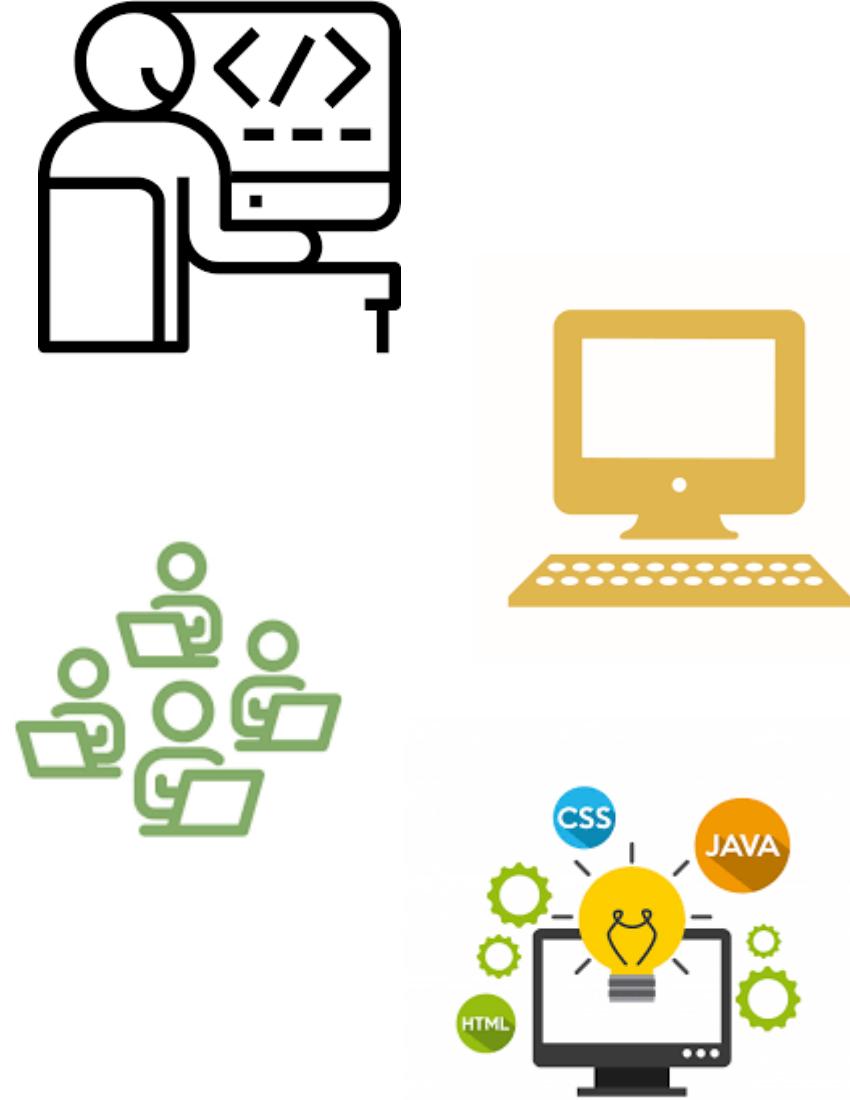
VENTAJAS



Flexibilidad: en el caso de ser necesario modificar la forma en que se representa el dato, sólo se debe modificar una declaración en lugar de un conjunto de declaraciones de variables.

Documentación: se pueden usar como identificador de los tipos, nombres autoexplicativos, facilitando de esta manera el entendimiento y lectura del programa.

Seguridad: se reducen los errores por uso de operaciones inadecuadas del dato a manejar, y se pueden obtener programas más confiables..



Conceptos de Algoritmos Datos y Programas

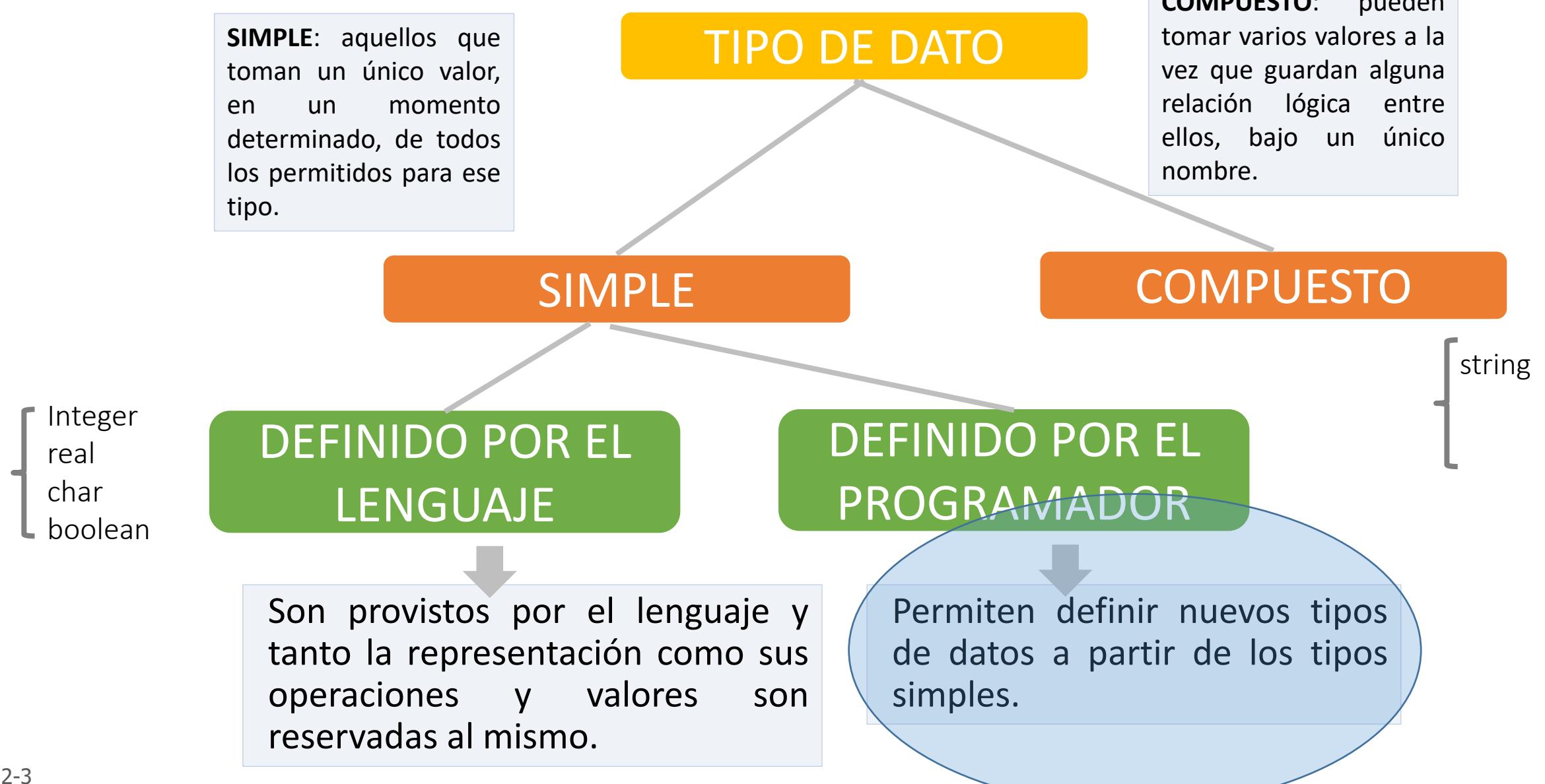
CADP – TEMAS



- Tipos de Datos definidos por el usuario
- Tipos de Datos subrango
- EJERCICIO – PREGUNTAS FINALES



CADP – TIPOS DE DATOS- CLASIFICACION





SUBRANGO

Es un tipo ordinal que consiste de una sucesión de valores de un tipo ordinal (predefinido o definido por el usuario) tomado como base.

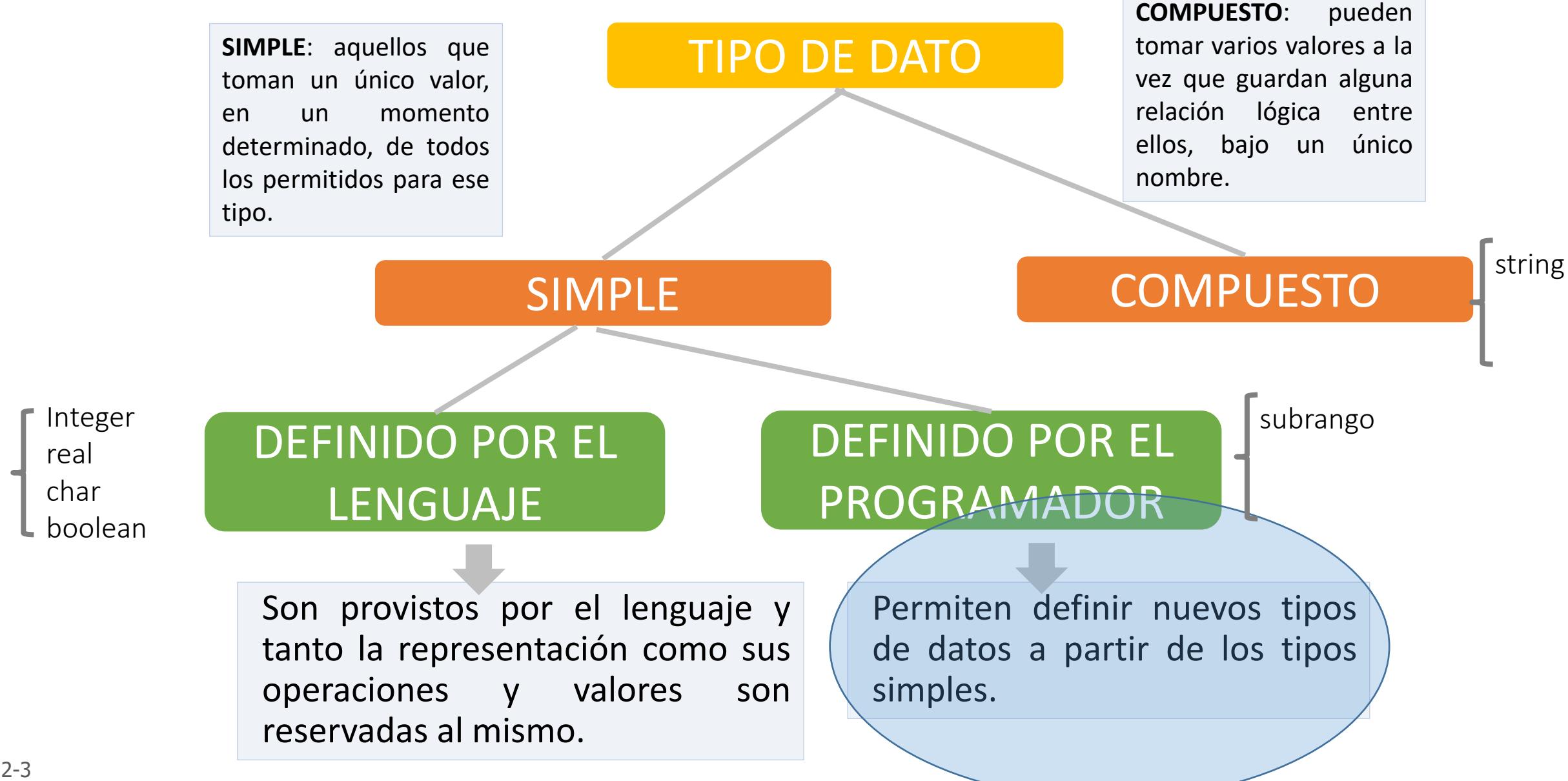
- Es simple.
- Es ordinal
- Existe en la mayoría de los lenguajes

Cómo se
declara?

Cómo se
usa?



CADP – TIPOS DE DATOS- CLASIFICACION



CADP – TIPOS DE DATOS DEFINIDOS POR EL USUARIO



SUBRANGO

program uno;

Const

...

Type

tipo1 = valor1..valor2;

var

x,y: tipo1;

Begin

...

End.

CADP – TIPOS DE DATOS DEFINIDOS POR EL USUARIO



SUBRANGO

Qué operaciones
puedo hacer con
cada variable?

program uno;

Const

...

Type

notas = 0..10;

letras = 'a'..'l';

var

not1,not2: notas;

let:letras;

Begin

...

End.

Se puede hacer un
subrango de
reales?



SUBRANGO – OPERACIONES

Operaciones Permitidas

- Asignación
- Comparación
- Todas las operaciones permitidas para el tipo base

Operaciones NO permitidas

- Depende del tipo base.

CADP – EJERCITACION



Programa 1: Se pide realizar que lea caracteres (suponga sólo minúsculas) hasta leer el carácter ‘z’. Al finalizar informar cuantos caracteres leídos estaban entre ‘a’ y ‘h’; cuántos entre ‘i’ y ‘n’ y cuántos entre ‘ñ’ y ‘y’.

- Estructura de control principal del programa?
- Necesito definir algún tipo de datos?
- Cómo cuento según el valor?

CADP – EJERCITACION



Qué tiene de
malo esta
solución?

Qué puede
pasar?

```
Program uno;
var
  primer,segunda,tercer :integer;
  letra:char;
Begin
  primer:= 0;  segunda:=0;  tercer:=0;
  read (letra);
  while (letra <> 'z') do
    begin
      if ((letra = 'a') or (letra = 'b') or ...) then
        primer:= primer + 1
      else
        if ((letra = 'i') or (letra = 'j') or ...) then
          segunda:= segunda + 1
        else
          tercer:= tercer + 1;
      read (letra);
    end;
  write (primer,segunda,tercer);
end.
```

CADP – EJERCITACION



```
Program uno;
Type
  letras = 'a'..'z';
var
  primer,segunda,tercer:integer;
  letra:letras;
Begin
  primer:= 0;  segunda:=0;  tercer:=0;
  read (letra);
  while (letra <> 'z') do
    begin
      if ((letra = 'a') or (letra = 'b') or ...) then
        primer:= primer + 1
      else
        if ((letra = 'i') or (letra = 'j') or ...) then
          segunda:= segunda + 1
        else
          tercer:= tercer + 1;
      read (letra);
    end;
    write (primer,segunda,tercer);
end.
```

Se puede
mejorar?

CADP – EJERCITACION



```
Program uno;
Type
    letras = 'a'..'z';
var
    primer,segundo,tercer:integer;
    letra:letras;
Begin
    primer:= 0;  segunda:=0;  tercer:=0;
    read (letra);
    while (letra <> 'z') do
        begin
            case letra of
                'a'..'h': primer:= primer + 1;
                'i'..'n': segunda:= segunda + 1;
                'ñ'..'y': tercer:= tercer + 1;
            end;
            read (letra);
        end;
        write (primer,segunda,tercer);
    end.
```



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS

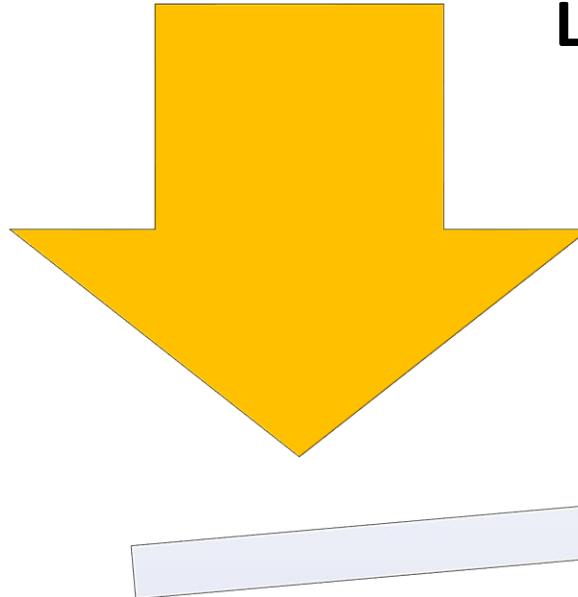
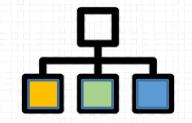


Modularización



EJERCICIO – PREGUNTAS FINALES

CADP – MODULARIZACION

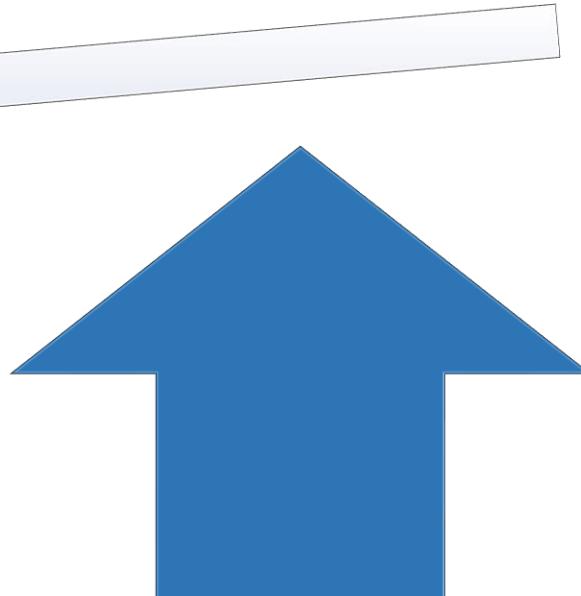


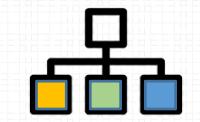
Los problemas del mundo real implican:

- Complejidad**
- Extensión**
- Modificaciones**

Los tratamos de resolver con:

- Abstracción**
- Descomposición**
- Independencia Funcional**





MODULARIZAR

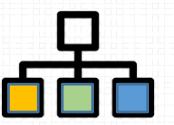
Significa dividir un problema en partes funcionalmente independientes, que encapsulen operaciones y datos.



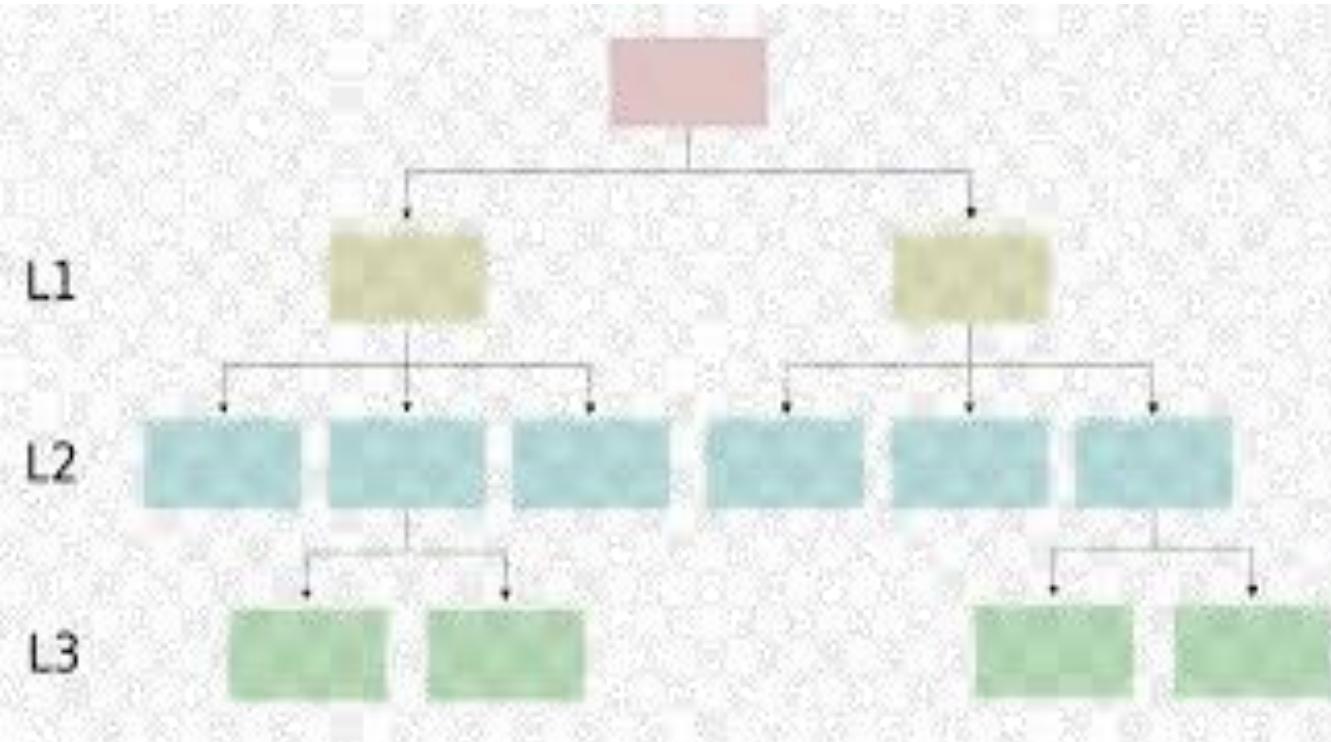
No se trata simplemente de subdividir el código de un sistema de software en bloques con un número de instrucciones dado.



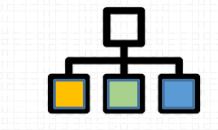
Separar en funciones lógicas con datos propios y datos de comunicación perfectamente especificados.



CADP – MODULARIZACION



- Cada subproblema está en un mismo nivel de detalle.
- Cada subproblema puede resolverse independientemente.
- Las soluciones de los subproblemas pueden combinarse para resolver el problema original.



MODULO

Tarea específica bien definida se comunican entre sí adecuadamente y cooperan para conseguir un objetivo común.

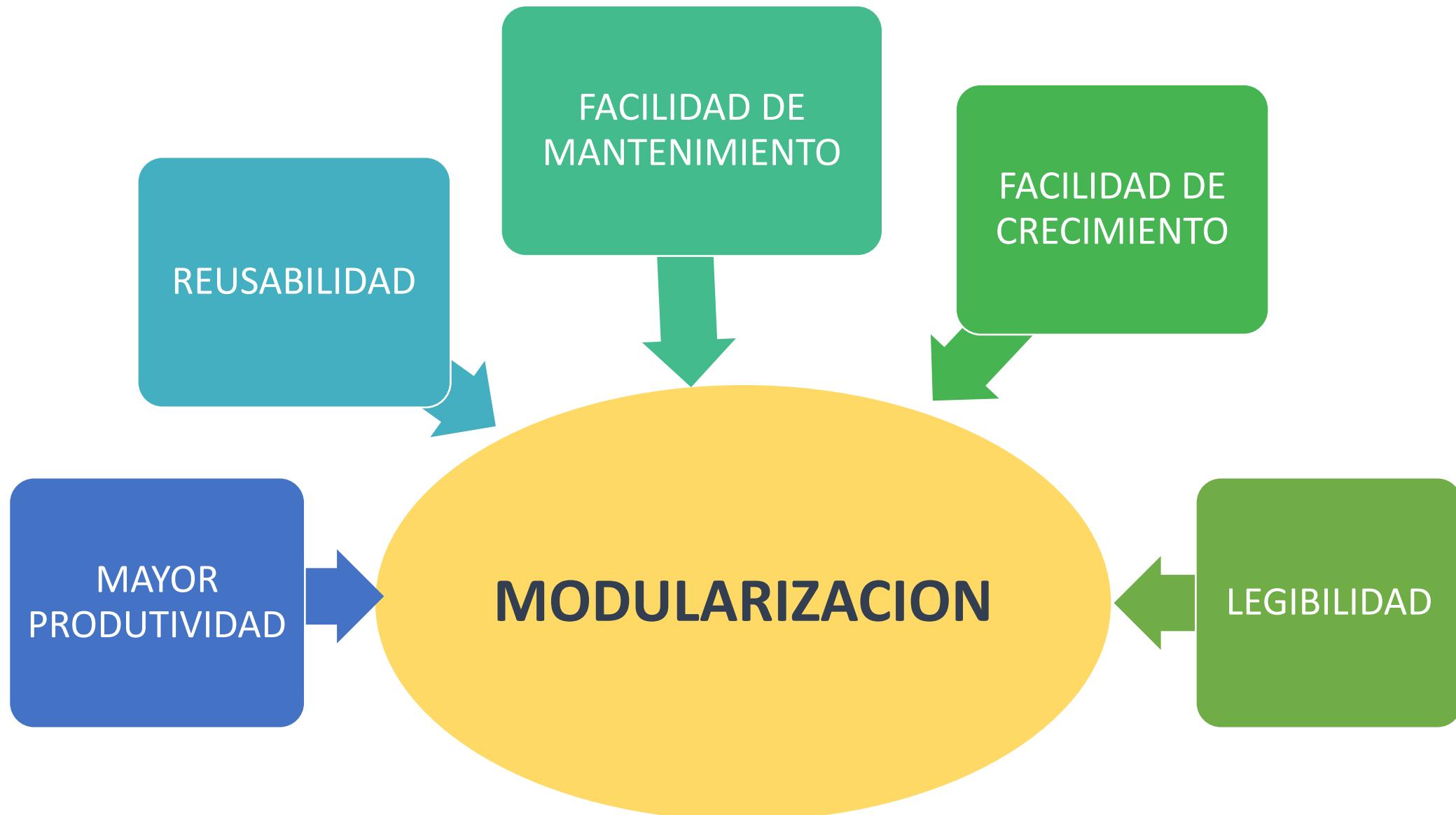
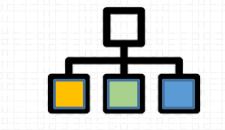
Encapsula acciones tareas o funciones.

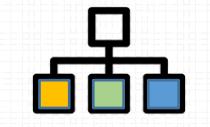
En ellos se pueden representar los objetivos relevantes del problema a resolver.

Existen diferentes metodologías para usarlos en los programas en particular nosotros usaremos la **METODOLOGIA TOP-DOWN**

CADP – MODULARIZACION

VENTAJAS

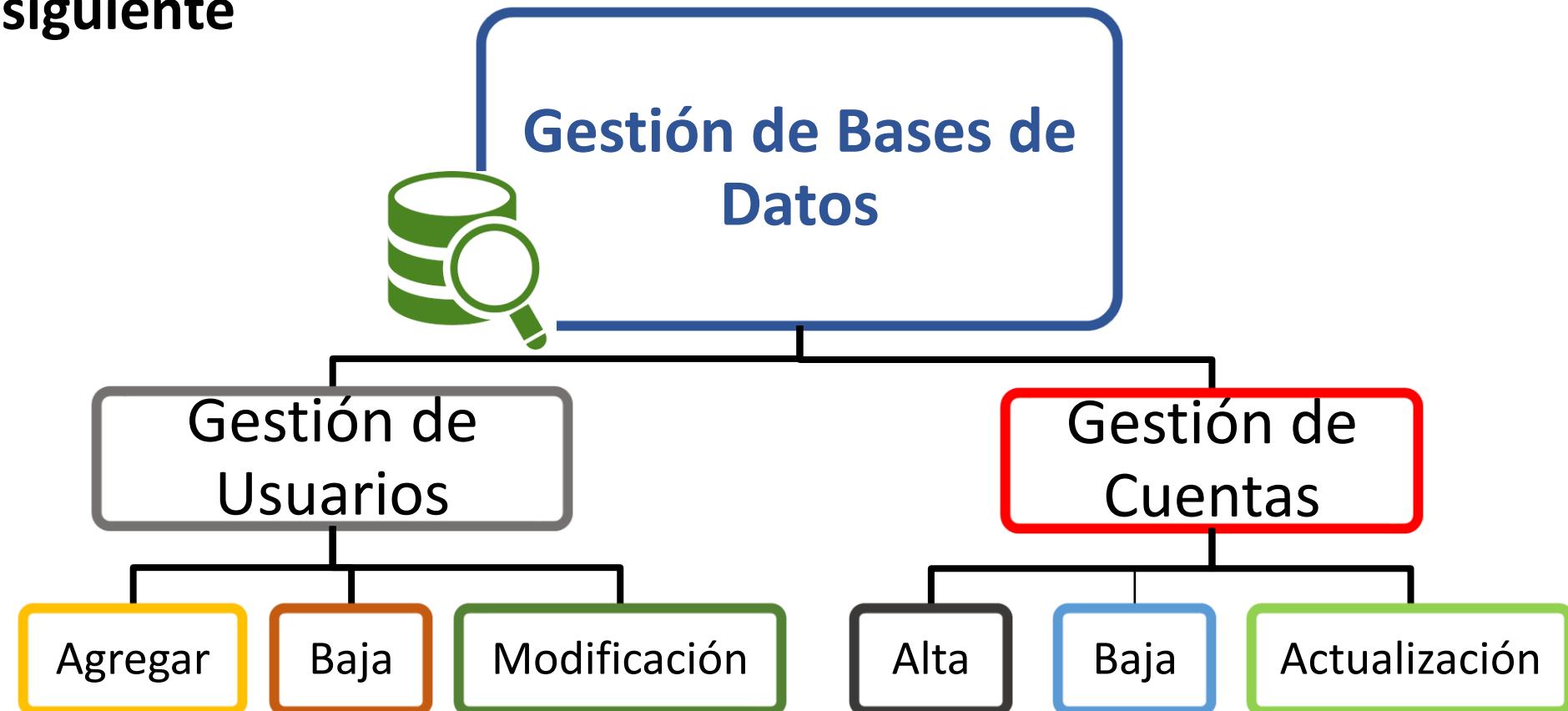


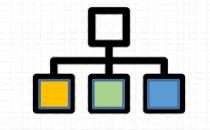


CADP – MODULARIZACION

VENTAJAS

Supongamos que tenemos que resolver el siguiente proyecto



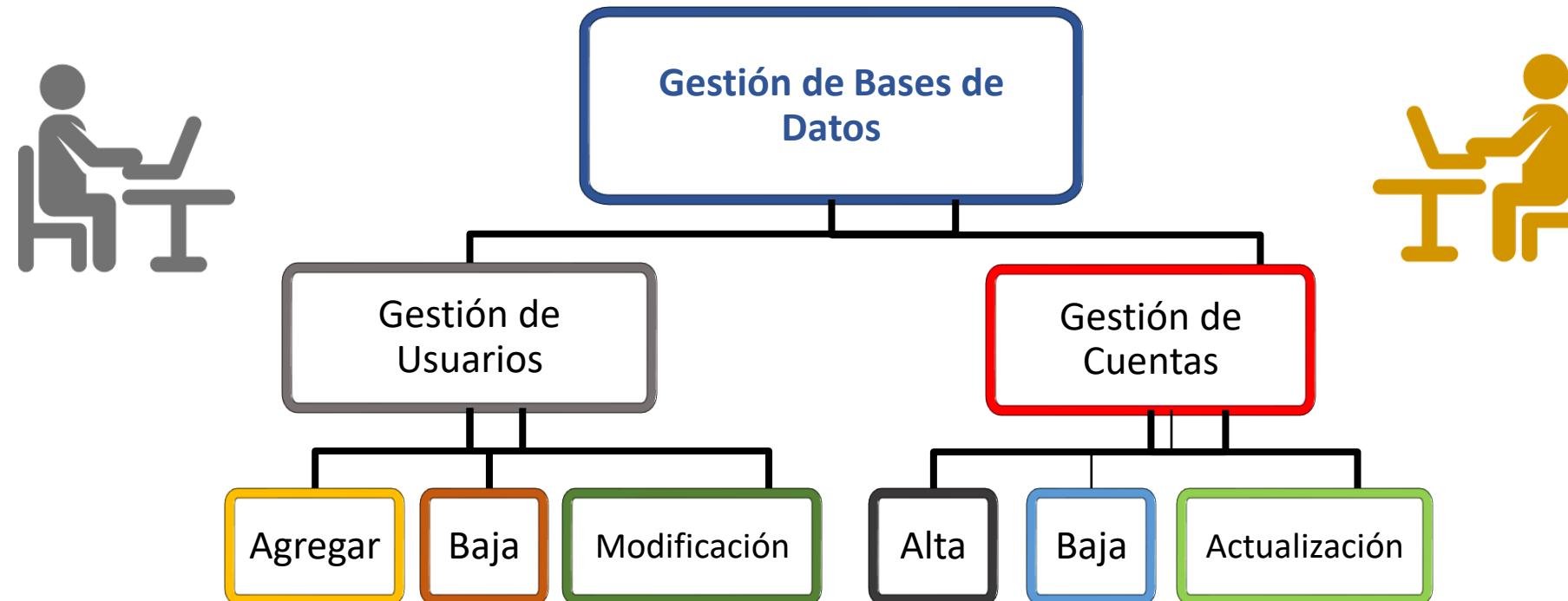


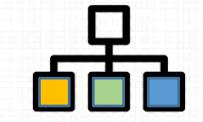
CADP – MODULARIZACION

VENTAJAS

Al dividir un sistema de software en módulos funcionalmente independientes, un equipo de desarrollo puede trabajar simultáneamente en varios módulos, incrementando la productividad (es decir reduciendo el tiempo de desarrollo global del sistema).

**MAYOR
PRODUCTIVIDAD**



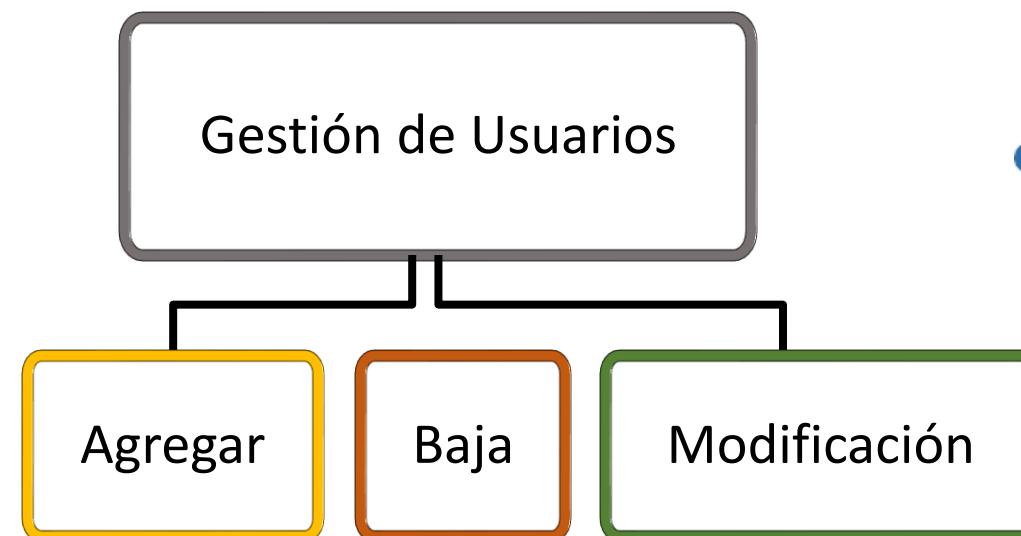


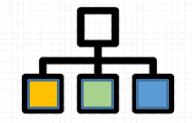
CADP – MODULARIZACION

VENTAJAS

Un objetivo fundamental de la Ingeniería de Software es la reusabilidad, es decir la posibilidad de utilizar repetidamente el producto de software desarrollado. Naturalmente la descomposición funcional que ofrece la modularización favorece el reuso.

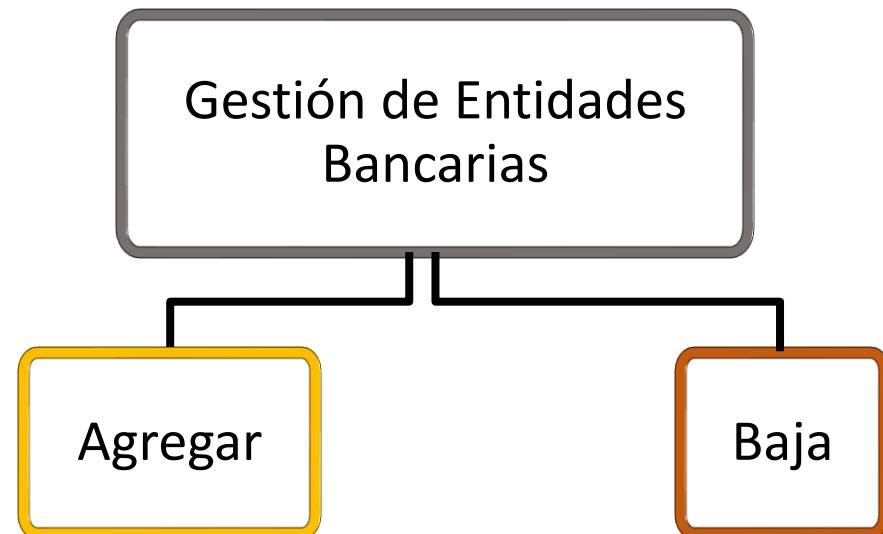
REUSABILIDAD





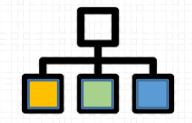
Los sistemas de software reales crecen (es decir aparecen con el tiempo nuevos requerimientos del usuario). La modularización permite disminuir los riesgos y costos de incorporar nuevas prestaciones a un sistema en funcionamiento.

FACILIDAD DE CRECIMIENTO



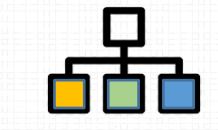
CADP – MODULARIZACION

VENTAJAS



Un efecto de la modularización es una mayor claridad para leer y comprender el código fuente. El ser humano maneja y comprende con mayor facilidad un número limitado de instrucciones directamente relacionadas.

LEGIBILIDAD



Fortran

- Subroutine

Modula

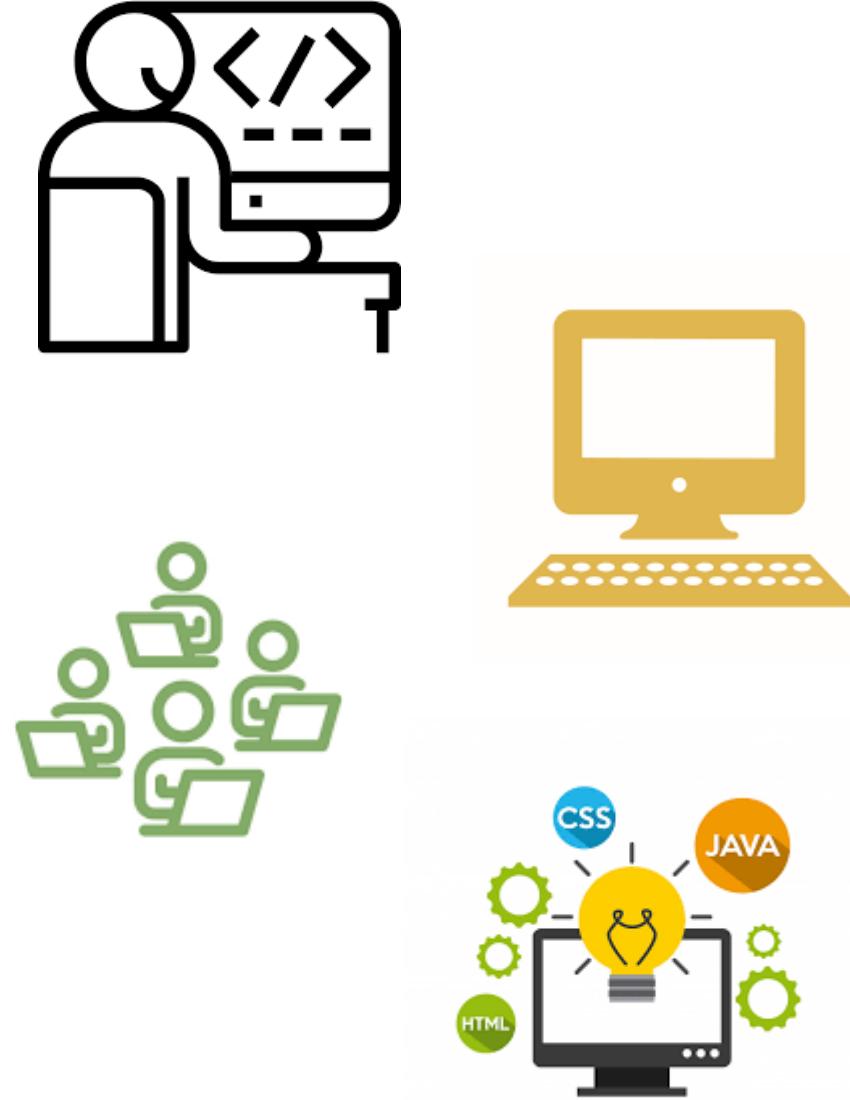
- Module

Ada, Pascal, C

- Procedure/Function

Orientado a objetos

- Class



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS

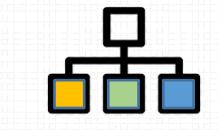


Modularización - PROCEDIMIENTOS

EJERCICIO – PREGUNTAS FINALES

CADP – MODULARIZACION

PROCEDIMIENTOS



Programa nombre

areas

...

Procesos

proceso nombre (parámetros)

variables

comenzar

fin

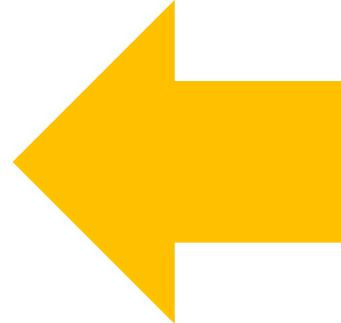
Variables

...

comenzar

...

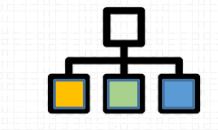
fin



Cómo son?

Cómo se declaran?

Cómo se usan?



PROCEDIMIENTO

Conjunto de instrucciones que realizan una tarea específica y retorna 0, 1 ó más valores.

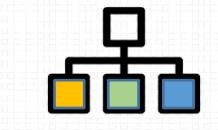
```
procedure nombre (parámetros);  
var  
    ....  
begin  
    ....  
end;
```

}

Variables locales

}

Código del procedimiento



CADP – MODULARIZACION PROCEDIMIENTO

```
Program uno;  
Const  
....  
Type  
....  
procedure auxiliar;  
Var  
  x:integer;  
begin  
  x:=8;  
end;  
  
Var  
....  
Begin  
...  
End.
```

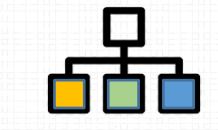
El procedimiento auxiliar no tiene parámetros

El procedimiento auxiliar tiene una variable x local declarada



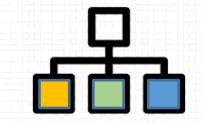
Qué modulo utilizaron en el curso de nivelación que no necesitaba recibir parámetros?

Cómo se invocan?



CADP – MODULARIZACION PROCEDIMIENTO

```
Program uno;  
Const  
    ....  
Type  
    ....  
procedure auxiliar;  
Var  
    x:integer;  
begin  
    x:=8;  
end;  
  
Var  
    ....  
Begin  
    auxiliar; ← Por su nombre  
End.
```



CADP – MODULARIZACION PROCEDIMIENTO

Program uno;

Const

....

Type

....

procedure auxiliar;

Var

 x:integer;

begin

 x:=8;

end;

Var

....



Begin //programa principal

...

 if (auxiliar) then

...

End.

Begin //programa principal

...

 while (auxiliar) do

...

End.

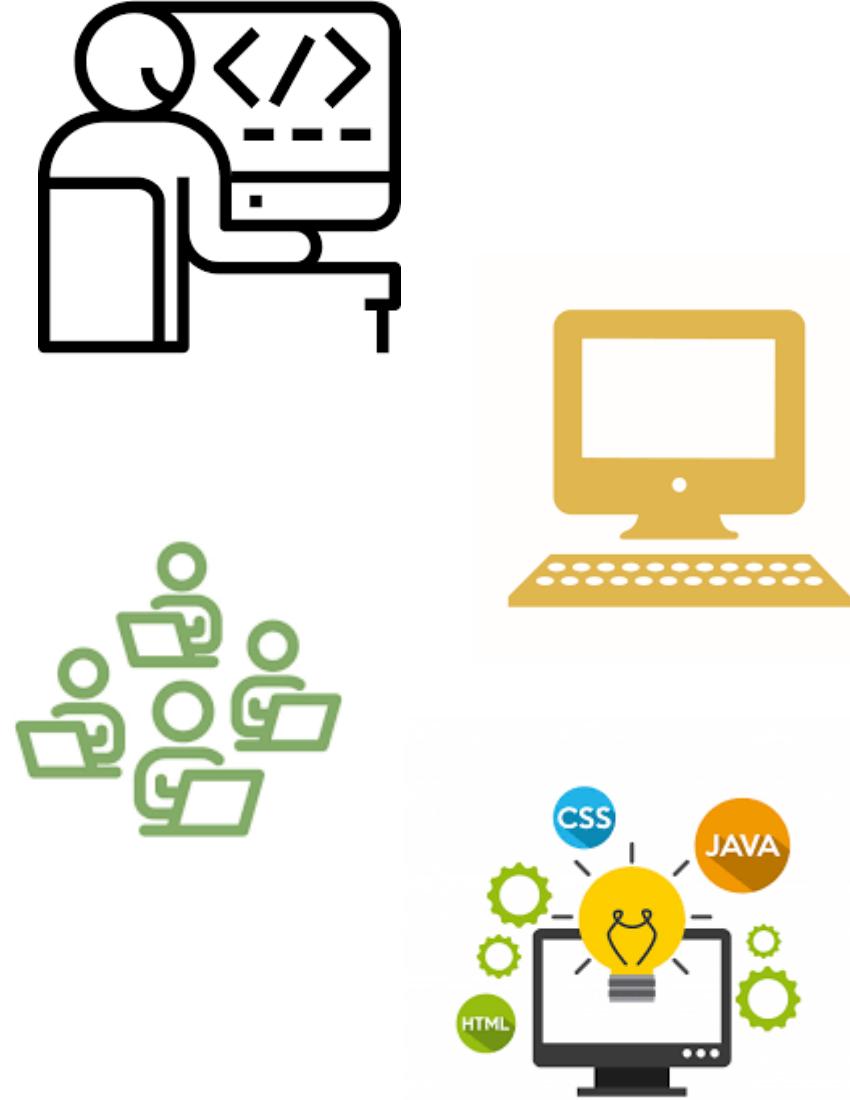
Begin //programa principal

...

 write (auxiliar);

...

End.



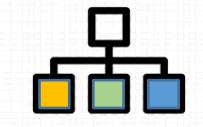
Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



Modularización - FUNCIONES

EJERCICIO – PREGUNTAS FINALES



CADP – MODULARIZACION

FUNCIONES

Programa nombre

areas

...

Procesos

proceso nombre (parámetros)

variables

comenzar

fin

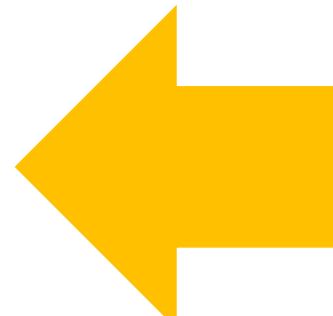
Variables

...

comenzar

...

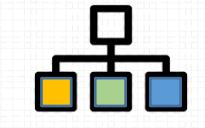
fin



Cómo son?

Cómo se declaran?

Cómo se usan?



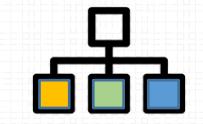
FUNCION

Conjunto de instrucciones que realizan una tarea específica y retorna 1 valor de tipo simple.

```
function nombre (parámetros): tipo;  
var  
    ....  
begin  
    ....  
end;
```

} Variables locales

} Código de la función



CADP – MODULARIZACION FUNCIONES

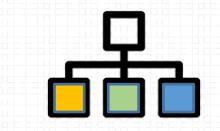
```
Program uno;
Const
  ....
Type
  ....
function auxiliar (parametros): tipo;
Var
  x:integer;
begin
  x:=8;
  ...
  auxiliar:= valor que se quiere retornar;
end;
Var
  ....
Begin
  ...
End.
```

La función auxiliar no tiene parámetros

La función auxiliar tiene una variable x local declarada

Tipo debe ser un tipo de datos simple

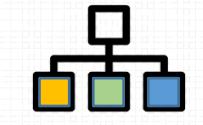
Para retornar el valor la última instrucción de la función debe ser asignarle a su nombre el valor que se quiere retornar



CADP – MODULARIZACION FUNCIONES

```
Program uno;
Const
  ....
Type
  ....
function auxiliar:integer;
Var
  x,resto:integer;
begin
  x:=8;
  resto:= x MOD 5;
  ...
  auxiliar:= resto;
end;
Var
  ....
Begin
  ...
End.
```

**Cómo se
invocan?**



CADP – MODULARIZACION

FUNCIONES

INVOCACION POR SU NOMBRE

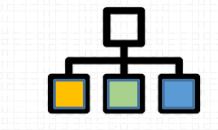
Invocación usando variable

El resultado se asigna
a una variable del
mismo tipo que
devuelve la función.

```
program uno;
Function auxiliar: real;
Var
  x, y, cociente:real;

begin
  x:= 10;
  y:= 4;
  cociente:= x/y;
  auxiliar:= cociente;
end;
Var
  aux:real;
begin
  aux:= auxiliar;
  write (aux);
end.
```

El retorno de la
función es a la
misma línea de
invocación



CADP – MODULARIZACION

FUNCIONES

INVOCACION POR SU NOMBRE

Invocación en un while
o
en un if

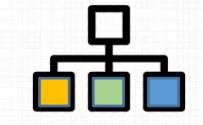
El resultado se utiliza
para evaluar la
condición.

```
program uno;
Function auxiliar: real;
Var
  x, y, cociente:real;

begin
  x:= 10;
  y:= 4;
  cociente:= x/y;
  auxiliar:= cociente;
end;
```

```
begin
  while (auxiliar = 5.5) do
    if (auxiliar = 5.5) then
end.
```

El retorno de la
función es a la misma
línea de invocación



CADP – MODULARIZACION

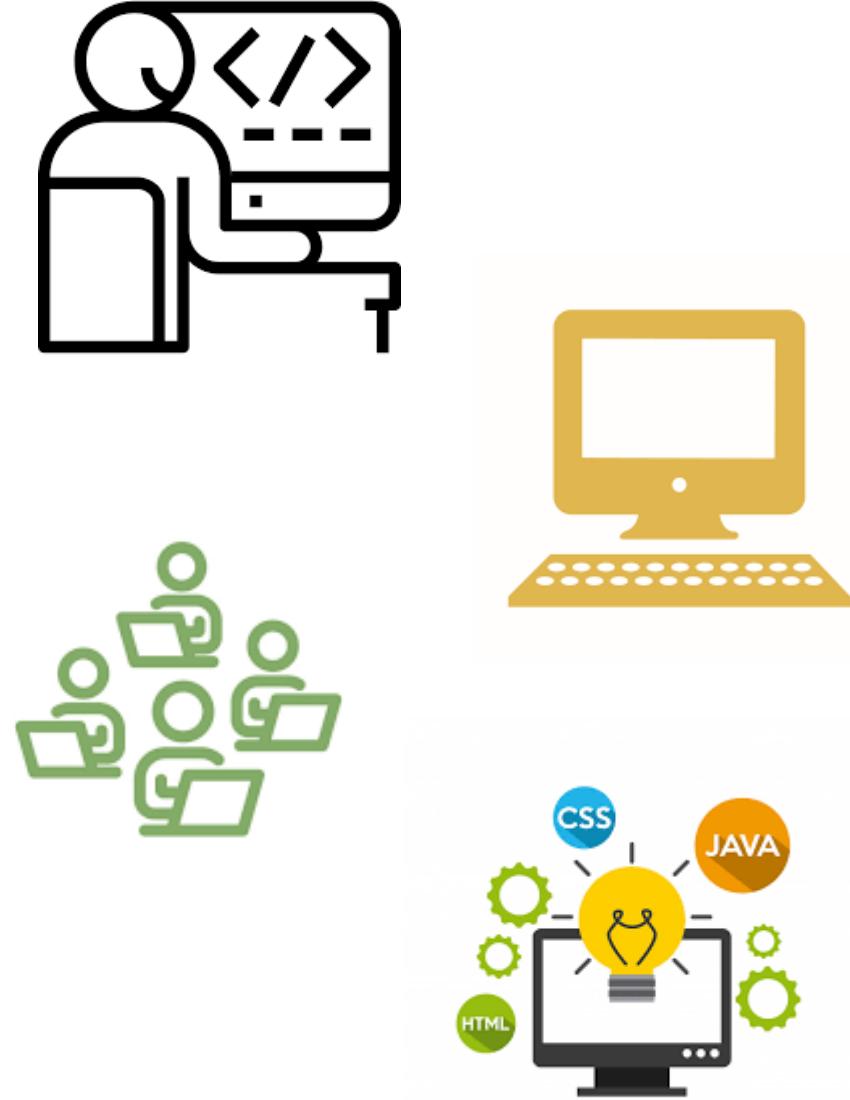
FUNCIONES

INVOCACION POR SU NOMBRE

Invocación en un write

El resultado se utiliza para informar en la sentencia write.

```
program uno;  
  
Function auxiliar: real;  
Var  
  x, y, cociente:real;  
  
begin  
  x:= 10;  
  y:= 4;  
  cociente:= x/y;  
  auxiliar:= cociente;  
end;  
  
begin  
  write ('El resultados es,'auxiliar);  
end.
```



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS

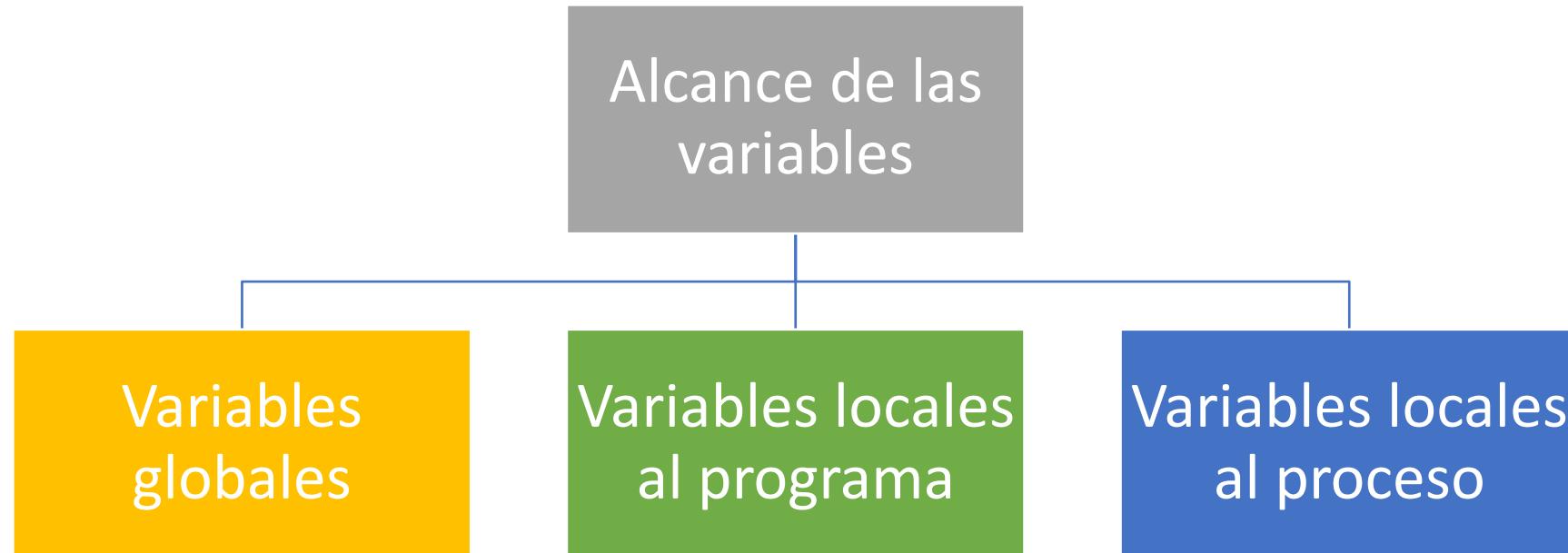
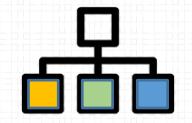


ALCANCE DE VARIABLES

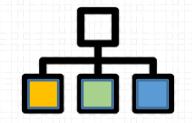


EJERCICIO – PREGUNTAS FINALES

CADP – MODULARIZACION ALCANCE DE LAS VARIABLES



CADP – MODULARIZACION ALCANCE DE LAS VARIABLES



```
Program alcance;
```

```
Var
```

```
  a,b: integer;
```

a,b, son **Variables globales** del programa



Pueden ser usadas en todo el programa (incluyendo módulos)

```
procedure prueba;
```

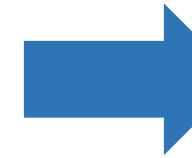
```
Var
```

```
  c: integer;
```

```
Begin
```

```
End.
```

c, es una **variable local** del proceso



Pueden ser usadas sólo en el proceso que están declaradas

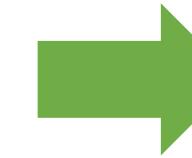
```
Var
```

```
  d:integer;
```

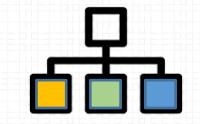
```
Begin
```

```
End.
```

d, es una **variable local** del programa



Pueden ser usadas sólo en el cuerpo del programa



CADP – MODULARIZACION ALCANCE DE LAS VARIABLES

```
Program alcance;  
Const  
...  
Type  
...  
Var
```

```
a,b: integer;
```

```
Procedure prueba;  
Var  
c: integer;  
Begin  
End.
```

```
Var  
d:integer;  
Begin  
End.
```

```
Program alcance;  
Const  
...  
Type  
...  
Var
```

```
a,b: integer;
```

```
Procedure prueba;  
Var  
c: integer;  
Begin  
End.
```

```
Var  
d:integer;  
Begin  
End.
```

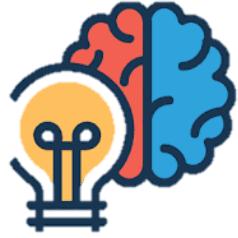
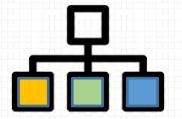
```
Program alcance;  
Const  
...  
Type  
...  
Var
```

```
a,b: integer;
```

```
Procedure prueba;  
Var  
c: integer;  
Begin  
End.
```

```
Var  
d:integer;  
Begin  
End.
```

CADP – MODULARIZACION ALCANCE DE LAS VARIABLES



Qué imprime?

```
Program alcance;  
Var  
    x,y: integer;  
  
Procedure prueba;  
Var  
    x:integer;  
Begin  
    x:= 34 DIV 3;  
    write (x);  
End;  
Var  
    x:integer;  
Begin  
    x:= 8; y:=9;  
    prueba;  
    write (x);  
    write (y);  
End.
```

Variables de programa (globales)

x:=
y:= 9

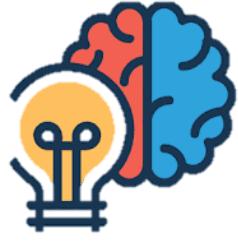
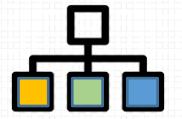
Variables del proceso prueba

x:= 11 Imprime 11

Variables del programa (locales)

x:= 8 Imprime 8
 Imprime 9

CADP – MODULARIZACION ALCANCE DE LAS VARIABLES



Qué imprime?

```
Program alcance;  
Var  
    x,y: integer;  
  
Procedure prueba;  
Var  
    x:integer;  
Begin  
    x:= 34 DIV 3;  
    write (x);  
End;  
Var  
    x:integer;  
Begin  
    x:= 8;  
    prueba;  
    write (x);  
    write (y);  
End.
```

Variables de programa (globales)

x:=
y:=

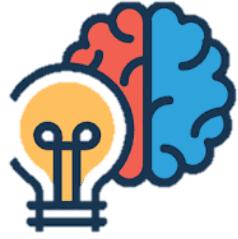
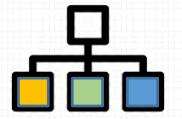
Variables del proceso prueba

x:= 11 Imprime 11

Variables del programa (locales)

x:= 8 Imprime 8
 Imprime basura

CADP – MODULARIZACION ALCANCE DE LAS VARIABLES



Qué imprime?

```
Program alcance;  
Var  
    x: integer;  
  
Procedure prueba;  
Var  
    x:integer;  
Begin  
    x:= 34 DIV 3;  
    write (x);  
End;  
Var  
    x:integer;  
Begin  
    x:= 8;  
    prueba;  
    write (x);  
    write (y);  
End.
```

Variables de programa (globales)

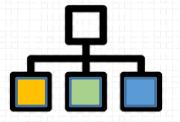
x:=

Variables del proceso prueba

x:= 11 Imprime 11

Variables del programa (locales)

x:= 8 Imprime 8
Da error



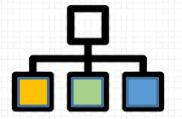
Si es una variable utilizada en un proceso

- Se busca si es variable local
- Se busca si es un parámetro
- Se busca si es variable global al programa

Si es una variable usada en un programa

- Se busca si es variable local al programa
- Se busca si es variable global al programa

CADP – MODULARIZACION ALCANCE DE LAS VARIABLES



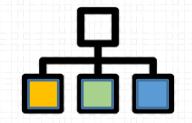
Qué
imprimen?

```
Program uno;
Var
  x,a,b: integer;
procedure prueba;
  var
    x: integer;
begin
  x:= 5;
  write (x);
end;
Begin
  x:=10;
  prueba;
  write (x);
End.
```

```
Program dos;
Var
  x,a,b: integer;
procedure prueba;
Begin
  write (x);
End;
Begin
  x:=5;
  prueba;
  write (x);
End.
```

```
Program tres;
Var
  x : char;
procedure prueba;
Begin
  x:=‘a’;
  prueba;
  write (x);
End.
```

CADP – MODULARIZACION ALCANCE DE LAS VARIABLES

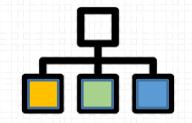


```
Program uno;
Var
  x,a,b: integer;
procedure prueba;
  type
    días = 1..7;
  var
    x: integer;
begin
  x:= 5;
end;
Begin
  x:=10;
  prueba;
  write (x);
End.
```

Se puede declarar un tipo nuevo dentro de un módulo?

Si se puede donde puedo declarar variables de ese tipo nuevo?

CADP – MODULARIZACION ALCANCE DE LAS VARIABLES

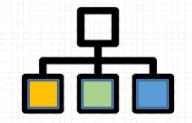


```
Program uno;
procedure prueba;
    procedure auxiliar
        var
            ...
        begin
        end;
    var
        x: integer;
    begin
        x:= 5;
    end;
Begin
    prueba;
End.
```

Se puede declarar un procedimiento dentro de otro?

Si se puede, desde donde se puede invocar a ese nuevo procedimiento?

CADP – MODULARIZACION ALCANCE DE LAS VARIABLES



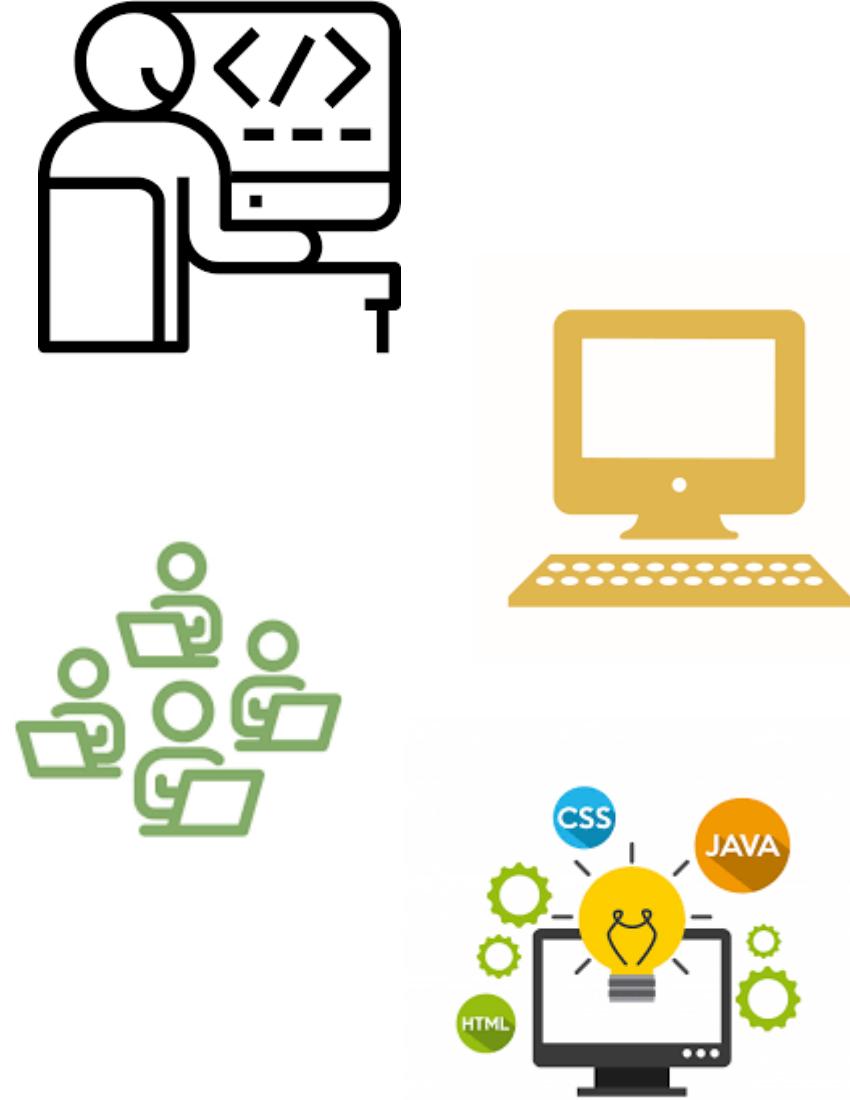
```
Program uno;
Var
  x:integer;
procedure prueba;
  procedure auxiliar;
  var
    ...
begin
  x:= 4;
end;
```

Se puede? A que x se hace referencia ?

```
var
  x: integer;
begin
  x:= 5;
end;
```

Se puede? A que x se hace referencia?

```
Begin
  prueba;
End.
```

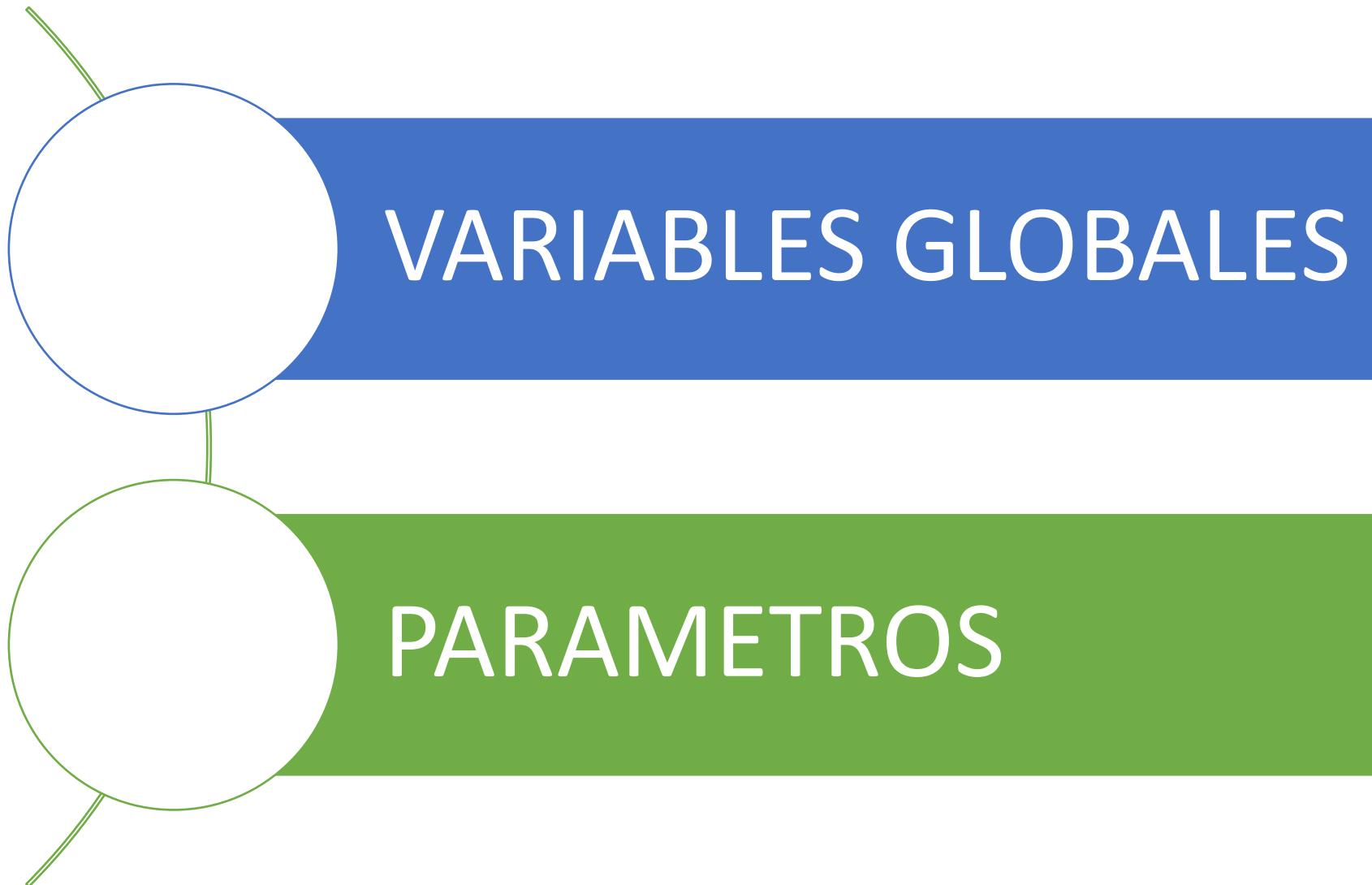
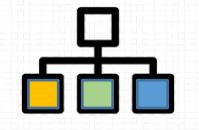


Conceptos de Algoritmos Datos y Programas

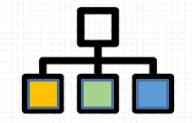
CADP – TEMAS



Comunicación entre módulos



CADP – COMUNICACION ENTRE MODULOS



VARIABLES GLOBALES

```
Program ejemplo1;
Var
  x:integer;

Procedure uno;
Begin
  x:= x+1;
  write (x);
End;
Procedure dos;
Begin
  x:= x MOD 10;
  write (x);
End;
var
  x: integer;

Begin
  x:=9;
  uno;
  write (x);
End.
```

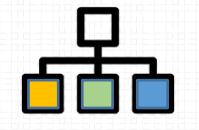


Demasiados identificadores

No se especifica la comunicación entre los módulos

Conflictos de nombres de identificadores utilizados por diferentes programadores.

Posibilidad de perder integridad de los datos, al modificar involuntariamente en un módulo datos de alguna variable que luego deberá utilizar otro módulo.

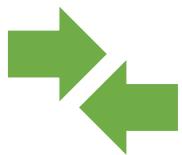


PARAMETROS

La solución a estos problemas ocasionados por el uso de variables globales es una combinación de **ocultamiento de datos (Data Hiding)** y **uso de parámetros**.

El ocultamiento de datos significa que los datos exclusivos de un módulo NO deben ser "visibles" o utilizables por los demás módulos.

El uso de parámetros significa que los datos compartidos se deben especificar como parámetros que se trasmitten entre módulos.



PARAMETROS – Cómo vamos a trabajar?

1

- Se analiza para cada módulo entonces: ¿cuáles son los datos propios? y ¿cuáles son los datos compartidos?

2

- Los datos propios se declararan locales al módulo.

3

- Los datos compartidos se declararán como parámetros.

Parámetros por
valor

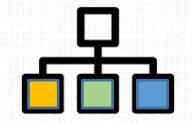
Parámetros por
referencia



PARAMETROS

VALOR	REFERENCIA
<ul style="list-style-type: none">• El módulo recibe un valor, puede realizar operaciones y/o cálculos, pero no producirá ningún cambio ni tampoco tendrá incidencia fuera del módulo.	<ul style="list-style-type: none">• El módulo recibe una dirección, puede realizar operaciones y/o cálculos, que producirán cambios y tendrán incidencia fuera del módulo.

CADP – COMUNICACION ENTRE MODULOS



PARAMETROS

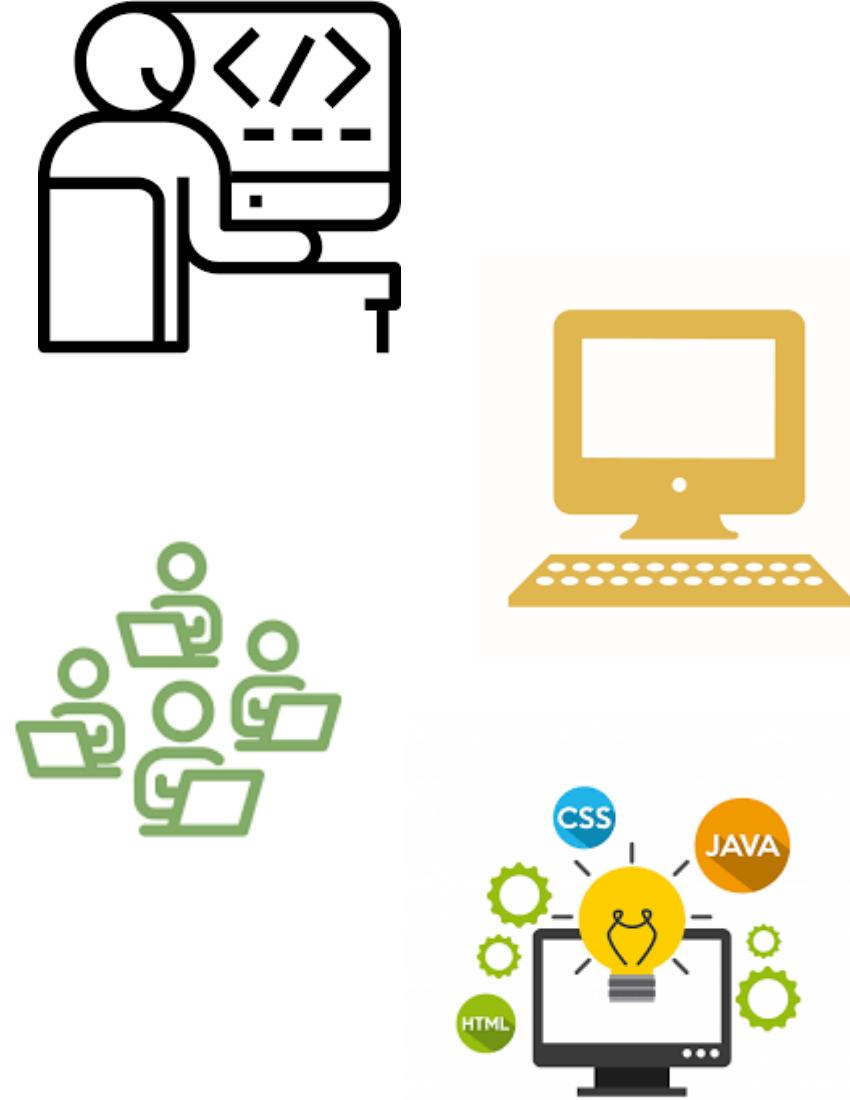
```
Program ejemplo2;  
  
Procedure uno (PARAMETO1; PARAMETO2);  
Begin  
  ...  
End;  
Procedure dos (PARAMETO);  
Begin  
  ...  
End;  
var  
  x,y,z: integer;  
  
Begin  
  ...  
  uno(x,y);  
  dos(z);  
  ...  
End.
```



Cada módulo indica que necesita recibir

Cada módulo indica que devuelve

No existe el problema donde un se pueda modificar el valor sin darse cuenta.

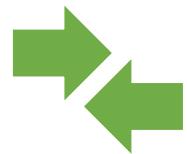


Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



- Parámetros por valor
- Parámetros por referencia
- Ejercicios – PREGUNTAS FINALES



PARAMETRO POR VALOR

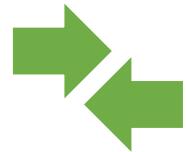
Un dato de entrada por valor es llamado parámetro IN y significa que el módulo recibe (sobre una variable local) un valor proveniente de otro módulo (o del programa principal).

Con él puede realizar operaciones y/o cálculos, pero no producirá ningún cambio ni tampoco tendrá incidencia fuera del módulo.

Con qué tipo de parámetro se relaciona?

Cómo se declaran?

Cómo se usan?



PARAMETRO POR VALOR

```
procedure uno (nombre1: tipo; nombre2: tipo);
```

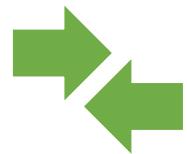
```
var
```

```
...
```

```
Begin
```

Uso de los parámetros con nombre1 y nombre2

```
End;
```



PARAMETRO POR VALOR

Program porValor;

```
procedure uno (num: integer);  
Begin
```

```
    if (num = 7)  then  
        num:= num + 1;  
    write (num);
```

```
end;
```

```
var
```

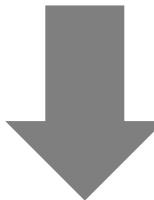
```
    x: integer;
```

```
begin
```

```
    x:= 7;
```

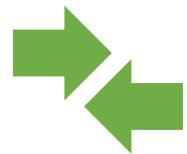
```
    uno (x);
```

```
end.
```



Dentro del procedimiento uno, el parámetro **num** copia el valor enviado por **x** (variable del programa)

Cómo
funciona?



```
Program porValor;  
  
procedure uno (num: integer);  
Begin  
    if (num = 7) then  
        num:= num + 1;  
    write (num);  
end;  
var  
    x: integer;  
begin  
    x:= 7;  
    uno (x);  
end.
```



Qué pasa si después de llamar al procedimiento uno en el programa imprimo num?

Procedimiento uno
Variables locales
Parámetros

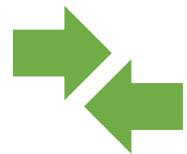
num = 8

Imprime 8

Programa ppal
Variables globales
Variables de prog

x = 7

Imprime 7



```
Program porValor;  
  
procedure uno (num: integer);  
Begin  
    if (num = 7) then  
        num:= num + 1;  
    write (num);  
end;  
var  
    num: integer;  
begin  
    num:= 7;  
    uno (num);  
end.
```



Qué pasa si después de llamar al procedimiento uno en el programa imprimo num?

Procedimiento uno
Variables locales
Parámetros

num = 8

Imprime 8

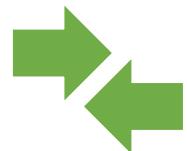
Programa ppal
Variables globales
Variables de prog

num = 7

Imprime 7

MEMORIA

CADP – MODULARIZACION COMUNICACION



```
Program porValor;
```

```
procedure uno (x: integer);
```

```
Begin
```

```
  if (x = 7) then
```

```
    x:= x + 1;
```

```
  write (x);
```

```
end;
```

```
var
```

```
  x: integer;
```

```
begin
```

```
  x:= 7;
```

```
  uno (x);
```

```
end.
```



Qué valores
imprimen?

```
Program porValor;
```

```
procedure uno (num: integer);
```

```
Var
```

```
  x:integer;
```

```
Begin
```

```
  if (num = 7) then
```

```
    num:= num + 1;
```

```
  x:= num;
```

```
  write (num); write (x);
```

```
end;
```

```
var
```

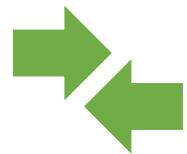
```
  x: integer;
```

```
begin
```

```
  x:= 7;
```

```
  uno (x);
```

```
end.
```



PARAMETRO POR REFERENCIA

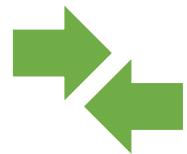
La comunicación por referencia (OUT, INOUT) significa que el módulo recibe el nombre de una variable (referencia a una dirección) conocida en otros módulos del sistema.

Puede operar con ella y su valor original dentro del módulo, y las modificaciones que se produzcan se reflejan en los demás módulos que conocen la variable.

Con qué tipo de parámetro se relaciona?

Cómo se declaran?

Cómo se usan?



PARAMETRO POR REFERENCIA

```
procedure uno (var nombre1: tipo; var nombre2: tipo);
```

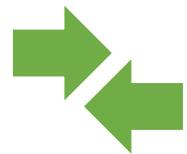
```
var
```

```
...
```

```
Begin
```

Uso de los parámetros con nombre1 y nombre2

```
End;
```



PARAMETRO POR REFERENCIA

Program porReferencia;

```
procedure uno (var num: integer);  
Begin
```

```
    if (num = ...) then  
        num:= num + 1;  
    write (num);
```

```
end;
```

```
var
```

```
    x: integer;
```

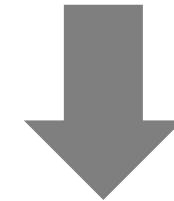
```
begin
```

```
    x:= 7;
```

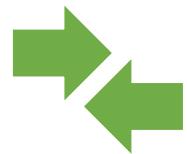
```
    uno (x);
```

```
end.
```

Dentro del procedimiento uno,
el parámetro **num** comparte la
dirección de memoria con **x**
(variable del programa)



Cómo
funciona?



Program porReferencia;

procedure uno (var num: integer);

Begin

 if (num = 7) then
 num:= num + 1;
 write (num);

end;

var

 x: integer;

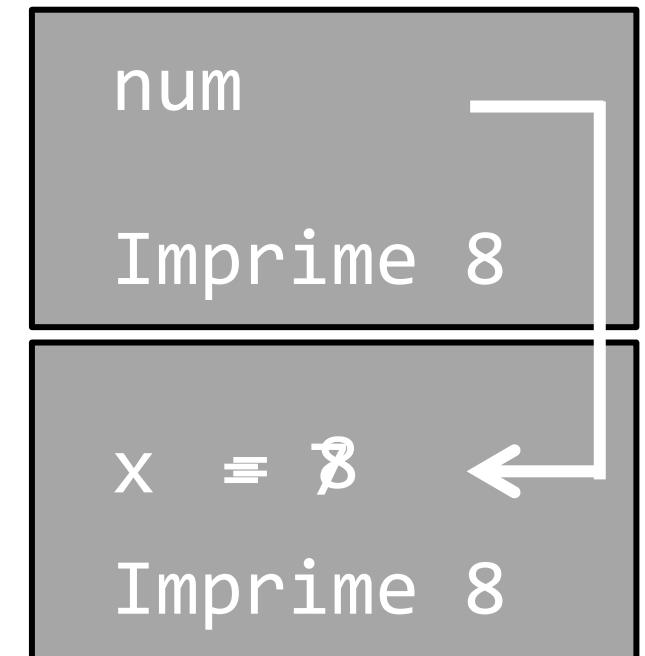
begin

 x:= 7;
 uno (x);

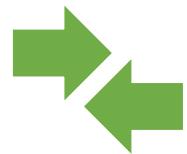
end.

Procedimiento uno
Variables locales
Parámetros

Programa ppal
Variables globales
Variables de prog



MEMORIA



Program porReferencia;

procedure uno (var num: integer);

Begin

 if (num = 7) then
 num:= num + 1;
 write (num);

end;

var

 num: integer;

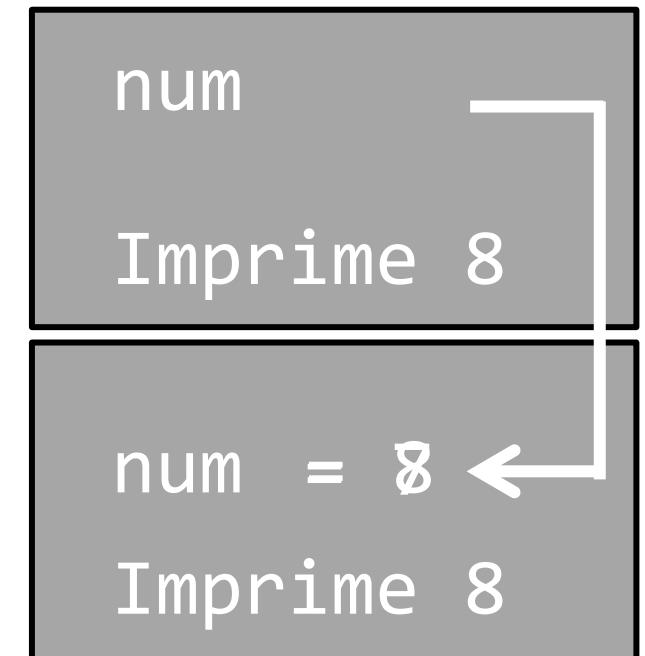
begin

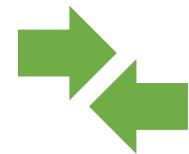
 num:= 7;
 uno (num);

end.

Procedimiento uno
Variables locales
Parámetros

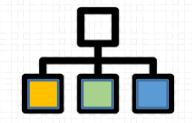
Programa ppal
Variables globales
Variables de prog





- El número y tipo de los argumentos utilizados en la invocación a un módulo deben coincidir con el número y tipo de parámetros del encabezamiento del módulo.
- Un parámetro por valor debiera ser tratado como una variable de la cuál el módulo hace una copia y la utiliza localmente. Algunos lenguajes permiten la modificación local de un parámetro por valor, pero toda modificación realizada queda en el módulo en el cual el parámetro es utilizado.
- El número y tipo de los argumentos utilizados en la invocación a un módulo deben coincidir con el número y tipo de parámetros del encabezamiento del módulo.

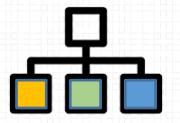
CADP – COMUNICACION ENTRE MODULOS



Implemente un programa que lea un valor entero e invoque a un módulo al cual le envía el valor leído y le devuelve si el valor recibido es múltiplo de 6.

- Cuántos y de qué tipo son los parámetros que recibe el módulo?
- Cuántos y de qué tipo son los parámetros que devuelve el módulo?
- Qué tipo de módulo utilizo?

CADP – COMUNICACION ENTRE MODULOS



Implemente un programa que lea un valor entero e invoque a un módulo al cual le envía el valor leído y le devuelve si el valor recibido es múltiplo de 6.

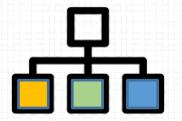
```
procedure multiplo (num:integer; var esmultiplo:boolean);
```

```
begin
  if (num MOD 6 = 0) then
    esmultiplo:= true
  else
    esmultiplo:= false;
end;
```

```
procedure multiplo (num:integer; var esmultiplo:boolean);
```

```
begin
  esmultiplo:= (num MOD 6 = 0);
end;
```

CADP – COMUNICACION ENTRE MODULOS



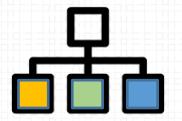
Implemente un programa que lea un valor entero e invoque a un módulo al cual le envía el valor leído y le devuelve si el valor recibido es múltiplo de 6.

```
function multiplo (num:integer):boolean;  
var  
  res:boolean;  
begin  
  if (num MOD 6 = 0) then  
    res:= true  
  else  
    res:= false;  
  multiplo:= res;  
end;
```

```
function multiplo (num:integer):boolean;  
begin  
  if (num MOD 6 = 0) then  
    multiplo:= true  
  else  
    multiplo:= false;  
end;
```

```
function multiplo (num:integer):boolean;  
begin  
  multiplo:= (num MOD 6 = 0);  
end;
```

CADP – COMUNICACION ENTRE MODULOS



Implemente un programa que lea un valor entero e invoque a un módulo al cual le envía el valor leído y le devuelve si el valor recibido es múltiplo de 6.

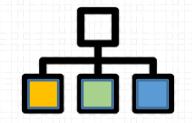
```
program uno;
procedure multiplo (num:integer;
                     var esmultiplo:boolean);
begin
  esmultiplo:= (num MOD 6 = 0);
end;

var
  valor:integer;
  res:boolean;
begin
  read(valor);
  multiplo (valor,res);
  write (res);
end.
```

```
program uno;
function multiplo (num:integer):boolean;
begin
  multiplo:= (num MOD 6 = 0);
end;

var
  valor:integer;
begin
  read(valor);
  write (multiplo (valor));
end.
```

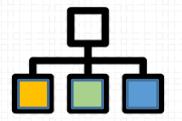
CADP – COMUNICACION ENTRE MODULOS



Implemente un programa que lea un valor entero e invoque a un módulo al cual le envía el valor leído y le devuelve cuántos dígitos pares y cuántos impares componen el numero recibido

- Cuántos y de qué tipo son los parámetros que recibe el módulo?
- Cuántos y de qué tipo son los parámetros que devuelve el módulo?
- Qué tipo de módulo utilizo?

CADP – COMUNICACION ENTRE MODULOS



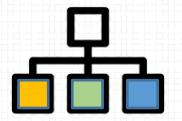
Implemente un programa que lea un valor entero e invoque a un módulo al cual le envía el valor leído y le devuelve cuántos dígitos pares y cuántos impares componen el numero recibido

```
procedure contador (num:integer; var par:integer; var impar:integer);  
var  
    resto:integer;  
begin  
    par:= 0;  
    impar:=0;  
    while (num <> 0) do  
        begin  
            resto:= num MOD 10;  
            if (resto MOD 2 = 0) then par:= par + 1  
            else impar:= impar + 1;  
            num:= num DIV 10;  
        end;  
    end;
```

num = 2681

resto = 1	Impar = 1
resto = 8	Impar = 1
resto = 6	Par = 1
resto = 2	Impar = 1
resto = 2	Par = 2
resto = 2	Impar = 1
resto = 3	Par = 3

CADP – COMUNICACION ENTRE MODULOS



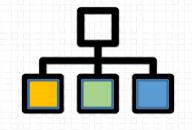
Implemente un programa que lea un valor entero e invoque a un módulo al cual le envía el valor leído y le devuelve cuántos dígitos pares y cuántos impares componen el numero recibido

```
program uno;
procedure contador (num:integer; var par:integer; var impar:integer);
var
begin
    ....
end;

var
    valor,pares,impares:integer;

begin
    read(valor);
    contador (valor,pares,impares);
    write (pares,impares);
end.
```

CADP – COMUNICACION ENTRE MODULOS



```
program uno;  
  
procedure uno (num:integer; var res:integer);
```

```
begin  
  ...  
end;
```

```
begin  
  uno (8,4);  
end.
```

Es correcta la
invocación?

CADP – COMUNICACION ENTRE MODULOS



```
program uno;  
  
procedure uno (num:integer; var res:integer);  
  
begin  
  ...  
end;  
  
var  
  x,pos:integer;  
begin  
  uno (8,pos);  
  uno (x,pos);  
end.
```

Es correcta la
invocación?

CADP – COMUNICACION ENTRE MODULOS



```
program uno;
const
  x = 49;

procedure uno (num:integer; var res:integer);

begin
  ...
end;

begin
  uno (8,x);
end.
```

Es correcta la
invocación?



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



Estructuras de Datos



SIMPLE: aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.

TIPO DE DATO

COMPUESTO: pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.

SIMPLE

DEFINIDO POR EL LENGUAJE

Integer
Real
Char
Boolean

DEFINIDO POR EL PROGRAMADOR

Subrango

COMPUESTO

DEFINIDO POR EL LENGUAJE

String

DEFINIDO POR EL PROGRAMADOR



Supongamos que se quiere representar la información de las distintas razas de animales que existen en una veterinaria. Para simplificar el problema supongamos que la veterinaria atiende solamente perros. De cada animal se conoce la raza, el nombre, la edad.



Qué información es
relevante para un perro?

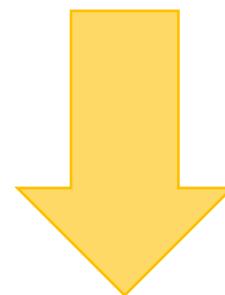
*Con lo que sabemos
hasta ahora como lo
representamos?*



Supongamos que se quiere representar la información de las distintas razas de animales que existen en una veterinaria. Para simplificar el problema supongamos que la veterinaria atiende solamente perros. De cada animal se conoce la raza, el nombre, la edad.



Un string para la raza
Un string para el nombre
Un entero para la edad



Son variables sueltas pero no hay una sensación de tener la información junta

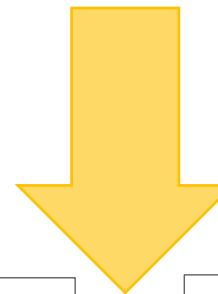


Supongamos que se quiere representar la información de las 35 materias que forman parte del plan de estudio de una carrera. De cada materia se conoce el nombre, año en que se cursa.

Plan de Estudios									
Ingeniería Química									
UTN FRM									
Títulos									
1ºAño									
2ºAño									
3ºAño									
4ºAño									
5ºAño									
6ºAño									
7ºAño									
8ºAño									
9ºAño									
10ºAño									
11ºAño									
12ºAño									
13ºAño									
14ºAño									
15ºAño									
16ºAño									
17ºAño									
18ºAño									
19ºAño									
20ºAño									
21ºAño									
22ºAño									
23ºAño									
24ºAño									
25ºAño									
26ºAño									
27ºAño									
28ºAño									
29ºAño									
30ºAño									
31ºAño									
32ºAño									
33ºAño									
34ºAño									
35ºAño									

Qué información es relevante para una materia?

Un string para el nombre
Un entero para el año



Son variables sueltas pero no hay una sensación de tener la información junta

Cómo guardo 35 materias?

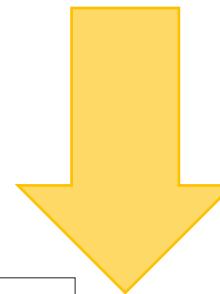


Supongamos que se quiere representar la información de los productos que vende un supermercado (la cantidad de productos es variable). De cada producto se conoce el nombre, cantidad que se tiene en stock y precio.



Qué información es relevante para un producto?

Un string para el nombre
Un entero para el stock
Un real para el precio



Son variables sueltas pero no hay una sensación de tener la información junta

Cómo guardo los productos aun si saber cuántos son?



ESTRUCTURA DE DATOS

Permite al programador definir un tipo al que se asocian diferentes datos que tienen valores lógicamente relacionados y asociados bajo un nombre único.

CLASIFICACION

Elementos	Acceso	Tamaño	Linealidad
Homogénea	Secuencial	Dinámica	Lineal
Heterogénea	Directo	Estática	No Lineal



ELEMENTOS

Depende si los elementos son del mismo tipo o no.

Homogénea



Los elementos que la componen son del mismo tipo

Heterogénea



Los elementos que la componen pueden ser de distinto tipo



TAMAÑO

Hace referencia a si la estructura puede variar su tamaño durante la ejecución del programa.

ESTATICA



El tamaño de la estructura no varía durante la ejecución del programa

DINAMICA



El tamaño de la estructura puede variar durante la ejecución del programa



ACCESO

Hace referencia a como se pueden acceder a los elementos que la componen.

SECUENCIAL



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.

DIRECTO



Se puede acceder a un elemento particular, directamente, sin necesidad de pasar por los anteriores a él, por ejemplo, referenciando una posición.



LINEALIDAD

Hace referencia a como se encuentran almacenados los elementos que la componen.

LINEAL

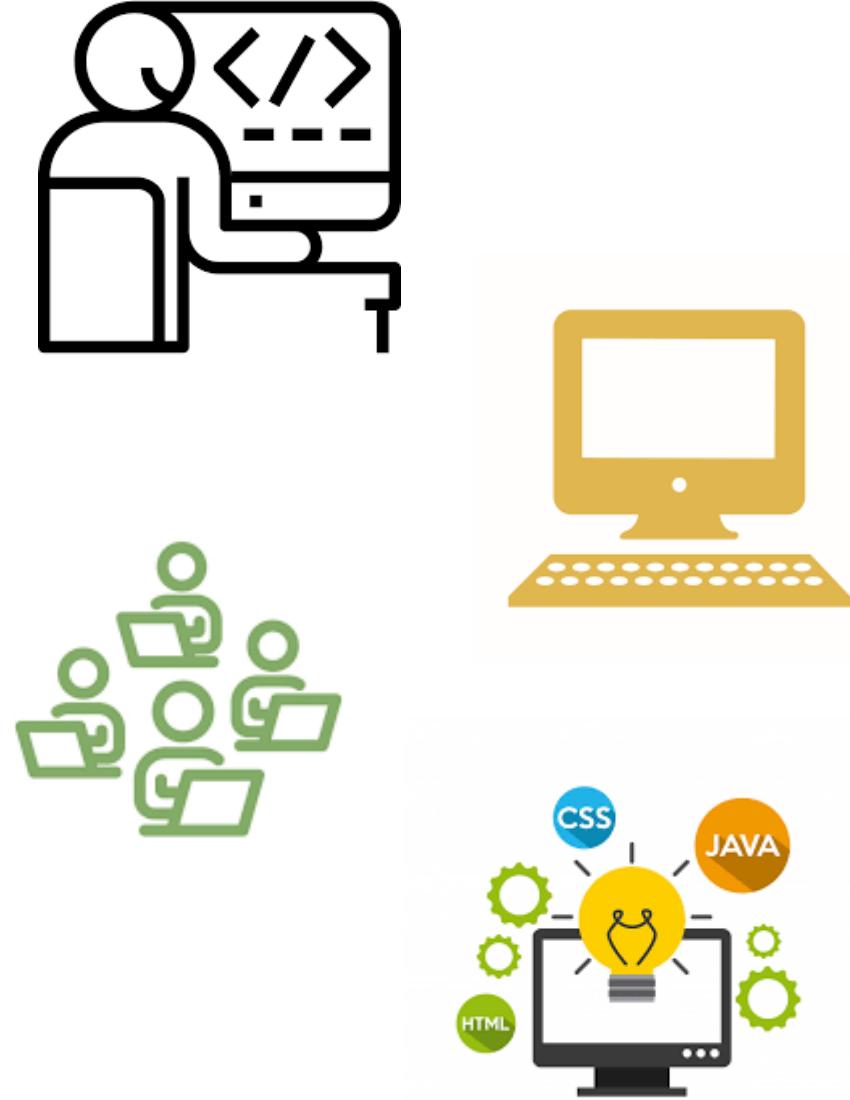


Está formada por ninguno, uno o varios elementos que guardan una relación de adyacencia ordenada donde a cada elemento le sigue uno y le precede uno, solamente.

NO LINEAL



Para un elemento dado pueden existir 0, 1 ó mas elementos que le suceden y 0, 1 ó mas elementos que le preceden.



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



Estructura de Datos REGISTRO

CADP – ESTRUCTURA DE DATOS

REGISTRO



SIMPLE: aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.

TIPO DE DATO

COMPUESTO: pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.

SIMPLE

COMPUESTO

DEFINIDO POR EL LENGUAJE

DEFINIDO POR EL PROGRAMADOR

DEFINIDO POR EL LENGUAJE

DEFINIDO POR EL PROGRAMADOR

Integer
Real
Char
Boolean

Subrango

String

Registro



Supongamos que se quiere representar la información de las distintas razas de animales que existen en una veterinaria. Para simplificar el problema supongamos que la veterinaria atiende solamente perros. De cada animal se conoce la raza, el nombre, la edad.



Qué información es
relevante para un perro?

*Con lo que sabemos
hasta ahora como lo
representamos?*



REGISTRO

Es un tipo de datos estructurado, que permite agrupar diferentes clases de datos en una estructura única bajo un sólo nombre

Raza
Nombre
Edad

Una manera natural y lógica de agrupar los datos de cada perro en una sola estructura es declarar un tipo **REGISTRO** asociando el conjunto de datos de cada uno.

REGISTRO
PERRO



Heterogénea ➔

Los elementos pueden ser de distinto tipo
(puede haber registros con todos elementos del mismo tipo)

Estática ➔

El tamaño no cambia durante la ejecución (se calcula en el momento de compilación)

Campos ➔

Representan cada uno de los datos que forman el registro



Program uno;

Const

...

Type

nombre = **record**

 campo1: tipo;

 campo2: tipo;

...

end;

Var

 variable: nombre;



Se nombra cada campo.

Se asigna un tipo a cada campo.

Los tipos de los campos deben ser estáticos.

*Cómo declaro el
registro PERRO?*



Program uno;

Const

...

Type

```
perro = record
    raza: string;
    nombre: string;
    edad: integer;
end;
```

Var

```
ani1, ani2: perro;
```

La característica principal es que un registro permite representar la información en una única estructura.

*Cómo se
trabaja con un
registro?*



**CON LA
VARIABLE
REGISTRO**

```
Program uno;  
Const  
....  
Type  
  
perro = record  
    raza: string;  
    nombre: string;  
    edad: integer;  
end;  
  
Var  
    ani1, ani2: perro;
```

Begin

....
ani2:= ani1;

...
End.



**La única operación
permitida es la asignación
entre dos variables del
mismo tipo**

CADP – ESTRUCTURA DE DATOS

REGISTRO



**CON LOS
CAMPOS DEL
REGISTRO**

Cómo se le
da valor?

```
Program uno;  
Const  
....  
Type  
  
perro = record  
    raza: string;  
    nombre: string;  
    edad: integer;  
end;  
  
Var  
    ani1, ani2: perro;
```

Begin

...

Puedo realizar las operaciones permitidas según el tipo de campo del registro

...

End.



La única forma de acceder a los campos es **variable.nombrecampo**

ani1.nombre

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
Program uno;
```

```
Const
```

```
...
```

```
Type
```

```
perro = record
```

```
  raza: string;
```

```
  nombre: string;
```

```
  edad: integer;
```

```
end;
```

```
Var
```

```
  ani1, ani2: perro;
```

```
Begin
```

```
  ani1.raza:=‘Callejero’;
```

```
  ani1.nombre:= ‘Bob’;
```

```
  ani1.edad:= 1;
```

```
End.
```

```
Begin
```

```
  read (ani1.raza);
```

```
  read(ani1.nombre);
```

```
  read(ani1.edad);
```

```
End.
```

Qué ocurre si no le
doy valor a todos los
campos?

Debo asignarlos en
el orden en que se
declararon?

MODULARIZAR?



No se puede hacer
read (ani1)

CADP – ESTRUCTURA DE DATOS

Procedure leer (**var** p:perro);

```
Begin  
  read (p.raza);  
  read(p.nombre);  
  read(p.edad);  
End.
```

Debo asignarlos en el orden en que se declararon?

Puede ser una función en vez de un procedimiento?

Cómo muestro el contenido de un registro?

Qué ocurre si no le doy valor a todos los campos?

REGISTRO

```
Program uno;  
Const  
  ...  
Type  
  perro = record  
    raza: string;  
    nombre: string;  
    edad: integer;  
  end;
```

```
Procedure leer (var p:perro);  
begin
```

```
  ...  
end;
```

```
Var  
  ani1, ani2: perro;
```

```
Begin  
  leer (ani1);  
  ani2:= ani1;  
End.
```



CADP – ESTRUCTURA DE DATOS

REGISTRO



```
Program uno;
Const
  ...
Type
  perro = record
    raza: string;
    nombre: string;
    edad: integer;
  end;
Var
  ani1, ani2: perro;
```

```
Begin
  leer (ani1);
  write (ani1.raza);
  write(ani1.nombre);
  write(ani1.edad);
End.
```

Qué ocurre si no le imprimo todos los campos?



No se puede hacer
write (ani1)

MODULARIZAR?

CADP – ESTRUCTURA DE DATOS



Procedure imprimir (p:perro);

```
Begin  
  write (p.raza);  
  write(p.nombre);  
  write(p.edad);  
End.
```

Debo asignarlos en el orden en que se declararon?

Puede ser una función en vez de un procedimiento?

Cómo se comparan dos registros?

Qué ocurre si no le imprimo todos los campos?

REGISTRO

```
Program uno;  
Const  
  ...  
Type  
  perro = record  
    raza: string;  
    nombre: string;  
    edad: integer;  
  end;  
Procedure leer (var p:perro);  
begin  
  ...  
end;  
Procedure imprimir (p:perro);  
begin  
  ...  
end;  
Var  
  ani1, ani2: perro;  
Begin  
  leer (ani1);  
  imprimir(ani1);  
End.
```

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
Program uno;  
Const  
  ...  
Type  
  
perro = record  
  raza: string;  
  nombre: string;  
  edad: integer;  
end;  
  
Var  
  ani1, ani2: perro;
```

Begin
 leer (ani1);
 leer (ani2),

 if ((ani1.raza = ani2.raza) and
 (ani1.nombre = ani2.nombre) and
 (ani1.edad = ani2.edad))
 then
 write (`Los registro son iguales`);
End.



No se puede hacer
ani1 = ani2

MODULARIZAR?



```
procedure iguales (p,p1:perro; var ok:boolean);  
Begin  
  if( (p.raza = p1.raza)and  
      (p.nombre = p1.nombre) and  
      (p.edad = p1.edad))  
  
    then ok:= true  
    else ok:= false;  
end;
```

*Puede ser una función
en vez de un
procedimiento?*

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
function iguales (p,p1:perro):boolean;  
Var  
  ok:Boolean;  
Begin  
  if( (p.raza = p1.raza)and (p.nombre = p1.nombre) and (p.edad = p1.edad))  
  then ok:= true  
  else ok:= false;  
  iguales:= ok;  
end;
```

```
function iguales (p,p1:perro):boolean;  
Var  
  ok:Boolean;  
Begin  
  ok:= ((p.raza = p1.raza)and  
         (p.nombre = p1.nombre)  
         and (p.edad = p1.edad))  
  iguales:= ok;  
end;
```

```
function iguales (p,p1:perro):boolean;  
Begin  
  iguales := ((p.raza = p1.raza)  
              and (p.nombre= p1.nombre)  
              and (p.edad = p1.edad));  
end;
```

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
Program uno;
Const
  ...
Type
perro = record
  raza: string;
  nombre: string;
  edad: integer;
end;
function iguales (p,p1:perro): boolean;
begin
  ...
end;
Procedure leer (var p:perro);
begin
  ...
end;
Procedure imprimir (p:perro);
begin
  ...
end;
```

Var
ani1, ani2: perro;

Begin

leer (ani1);
leer (ani2);
if (iguales (ani1,ani2) = true) then
 write (`Los registros son iguales`)
else write (`Los registros no son iguales`);
End.

Begin

leer (ani1);
leer (ani2);
if (**iguales (ani1,ani2)**) then
 write (`Los registros son iguales`)
else write (`Los registros no son iguales`);
End.



Escriba un programa que lea perros hasta leer un perro cuya raza es `XXX` Al finalizar informe de los perros en con nombre `Bob` y que tienen al menos 2 años

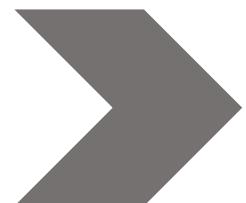
Raza `Ovejero`
Nombre `Bob`
edad:2

Raza `Callejero`
Nombre `Bob`
edad:1

Raza `Golden`
Nombre `Aragon`
edad:5

Raza `Ovejero`
Nombre `Lucy`
edad:3

Raza `Salchicha`
Nombre `Scoby`
edad:1



1



Escriba un programa que lea perros hasta leer un perro cuya raza es `XXX` Al finalizar informe de los perros en con nombre `Bob` y que tienen al menos 2 años

Inicializar contadores (cant)

Leer registro (ani)

While (no sea el ultimo registro) do
begin

 if (ani tiene nombre 'Bob') then
 if (ani tiene al menos dos años) then
 incremento (cant)

 leer registro (ani)

end;

Write (`La cantidad es`, cant);

Cuál es la estructura
de datos?

**Como verifico las
condiciones?**

Qué modularizo?

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
Program uno;
Const
  ...
Type
  perro = record
    raza: string;
    nombre: string;
    edad: integer;
  end;
// módulos
Var
  ani: perro;
  cant: integer;
```

Begin

cant:=0;

leer (ani);

while (ani.raza <> `XXX`) do

begin

if (cumpleNombre (ani) = true) then

if (edad (ani) = true) then

cant:= cant + 1;

leer (ani);

end;

write (`La cantidad es` , cant);

End.



```
procedure leer (var p:perro);
```

```
Begin  
    read(p.raza);  
    read(p.nombre);  
    read(p.edad);  
end;
```

Qué
alternativa
conviene?

```
procedure leer (var p:perro);
```

```
Begin  
    read(p.raza);  
    if (p.raza <> 'XXX') then  
        begin  
            read(p.nombre);  
            read(p.edad);  
        end;  
    end;
```



```
function cumpleNombre (p:perro): boolean;  
var  
    ok:boolean;  
  
begin  
    if (p.nombre = `Bob`) then  
        ok:= true  
    else  
        ok:= false;  
    cumpleNombre:= ok;  
end;
```

Otra opción

```
function cumpleNombre (p:perro): boolean;  
  
begin  
    cumpleNombre:= (p.nombre = `Bob`);  
end;
```



```
function edad (p:perro): boolean;  
var  
  ok:boolean;  
  
begin  
  if (p.edad >= 2) then  
    ok:= true  
  else  
    ok:= false;  
  edad:= ok;  
end;
```

Otra opción

```
function edad (p:perro): boolean;  
  
begin  
  edad:= (p.edad >= 2);  
end;
```

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
Program uno;
Type
perro = record
  raza: string;
  edad: integer;
  nombre: string;
end;
Procedure leer (p:perro);
begin
end;
function cumpleNombre (p:perro): boolean;
begin
  ...
end;

function edad (p:perro): boolean;
begin
  ...
end;
```

Var
ani:perro;
cant:integer;

Begin

cant:= 0;
 leer (ani);
 while (ani.raza <> `XXX`) do
 begin
 if (cumpleNombre (ani)) then
 if (edad (ani)) then
 cant:= cant + 1;
 leer (ani);
 end;
 write (`La cantidad es` , cant);

End.

Es necesario pasar todo el registro a las funciones?

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
Program uno;
Type
perro = record
  raza: string;
  edad: integer;
  nombre: string;
end;
Procedure leer (p:perro);
begin
end;
function cumpleNombre (n:string): boolean;
begin
  ...
end;

function edad (e:integer): boolean;
begin
  ...
end;
```

Var
ani:perro;
cant:integer;

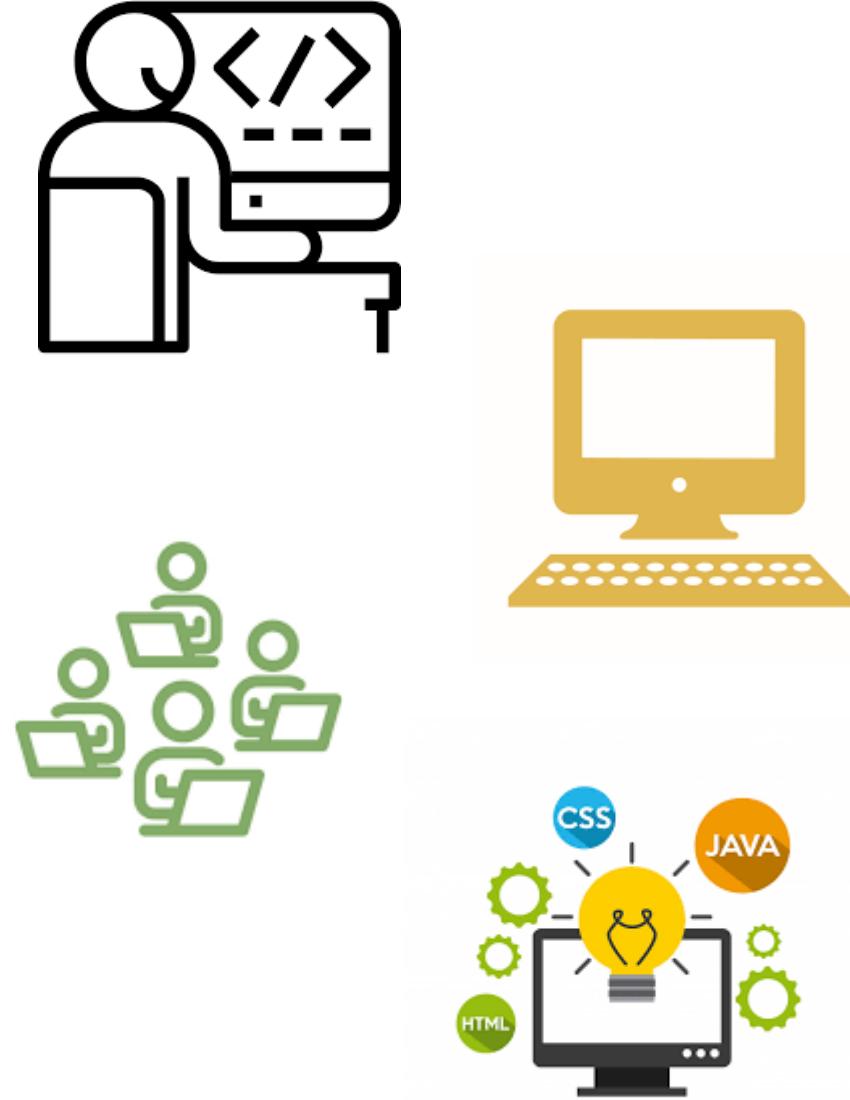
Begin
cant:= 0;
leer (ani);
while (ani.raza <> `XXX`) do
begin
 if (cumpleNombre (ani.nombre)) then
 if (edad (ani.edad)) then
 cant:= cant + 1;
 leer (ani);
end;
write (`La cantidad es` , cant);
End.

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
function cumpleNombre (nom:string): boolean;  
var  
  ok:boolean;  
begin  
  if (nom = 'Bob') then ok:= true  
  else ok:= false;  
  cumpleNombre:= ok;  
end;  
  
function edad (e:integer): boolean;  
var  
  ok:boolean;  
begin  
  if (e>= 2) then ok:= true  
  else ok:= false;  
  edad:= ok;  
end;
```



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



Estructura de Datos REGISTRO de REGISTRO



Supongamos que se quiere representar la información de las distintas razas de animales que existen en una veterinaria. Para simplificar el problema supongamos que la veterinaria atiende solamente perros. De cada animal se conoce la raza, el nombre, la edad y ahora nuestro perro tiene la fecha en que visitó por última vez la veterinaria



Con se representa
ahora?



Opción 1

```
Program uno;  
Type  
  perro = record  
    raza: string;  
    edad: integer;  
    nombre:string;  
    dia:integer;  
    mes:integer;  
    año:integer;  
  end;
```

```
Var  
  ani: perro;
```

```
Procedure leer (var p:perro);  
  
Begin  
  read (p.raza);  
  read(p.nombre);  
  read(p.edad);  
  read(p.dia);  
  read(p.mes);  
  read(p.año);  
End.
```

Otra
opción?



Opción 2

```
Program uno;  
Type  
  fecha = record  
    dia: integer;  
    mes:integer;  
    año:integer;  
  end;  
  perro = record  
    raza: string;  
    edad: integer;  
    nombre: string;  
    fechaVis:fecha;  
  end;
```

*Cómo hacemos
ahora el proceso
de lectura?*



```
procedure leer (var p:perro);  
Begin  
read (p.raza);  
read(p.nombre);  
read(p.edad);  
read(p.fechaVis);  
end;
```

Otra
alternativa?

NO SE PUEDE ya que **p.fechaVis** es
un registro y no se puede hacer
read directamente

```
procedure leer (var p:perro);  
Begin  
    read (p.raza);  
    read(p.nombre);  
    read(p.edad);  
    read(p.fechaVis.dia);  
    read(p.fechaVis.mes);  
    read(p.fechaVis.año);  
end;
```

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
procedure leerFecha (var f:fecha);
Begin
  read(f.dia);
  read(f.mes);
  read(f.año);
end;
```

```
procedure leer (var p:perro);
var
  fec:fecha;
Begin
  read(p.raza);
  read(p.nombre);
  read(p.edad);
  leerFecha (fec);
  p.fechaVis:= fec;
end;
```

Otra opción

```
procedure leer (var p:perro);
begin
  read(p.raza);
  read(p.nombre);
  read(p.edad);
  leerFecha (p.fechaVis);
end;
```



Escriba un programa que lea la información de 10 perros
Al finalizar informe de los perros que visitaron la
veterinaria en los primeros 15 días de los meses de enero
y febrero de 2023

Raza `Ovejero`
Nombre `Bob`
edad:2
Dia 8
Mes 1
Año 2022

Raza `Callejero`
Nombre `Bob`
edad:1
Dia 31
Mes 1
Año 2023

Raza `Golden`
Nombre `Aragon`
edad:5
Dia 4
Mes 2
Año 2023

Raza `Callejero`
Nombre `Laly`
edad:4
Dia 7
Mes 1
Año 2023



2



Escriba un programa que lea la información de 10 perros Al finalizar informe de los perros que visitaron la veterinaria en los primeros 15 días de los meses de enero y febrero de 2023

Iniciar contador (cant)

Repetir 10

begin

Leer registro (ani)

if (ani cumple fecha) then
incremento (cant)

end;

Write (`La cantidad es`, cant);

Cuál es la estructura de datos?

Como verifico las condiciones?

Qué modularizo?

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
Program uno;  
Type  
fecha = record  
  dia: integer;  
  mes:integer;  
  año:integer;  
end;  
  
perro = record  
  raza: string;  
  nombre: string;  
  edad: integer;  
  fechaVis:fecha;  
end;  
// módulos  
Var  
  ani: perro;  
  cant,i: integer;
```

Begin
cant:=0;

for i:= 1 to 10 do
begin

leer (ani);

if (cumpleFecha (ani) = true) then
 cant:= cant + 1;

end;

write (`La cantidad es` , cant);

End.



```
procedure leerFecha (var f:fecha);
Begin
    read(f.dia);
    read(f.mes);
    read(f.año);
end;
```

```
procedure leer (var p:perro);
Begin
    read(p.raza);
    read(p.nombre);
    read(p.edad);
    leerFecha(p.fechaVis);
end;
```



```
function cumpleFecha (p:perro): boolean;  
var  
    ok:boolean;  
  
begin  
    if ( ((p.fechaVis.dia >=1) and (p.fechaVis.dia <=15)) and  
        ((p.fechaVis.mes =1) or (p.fechaVis.mes =2)) and  
        (p.fechaVis.año = 2023) ) then  
        ok:= true  
  
    else  
        ok:= false;  
    cumpleFecha:= ok;  
end;
```

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
Program uno;
Type
fecha = record
  dia: integer;
  mes:integer;
  año:integer;
end;

perro = record
  raza: string;
  nombre: string;
  edad: integer;
  fechaVis:fecha;
end;
```

```
procedure leerFecha (var f:fecha);
Begin
  read(f.dia);
  read(f.mes);
  read(f.año);
end;
```

```
procedure leer (var p:perro);
Begin
  read(p.raza);
  read(p.nombre);
  read(p.edad);
  leerFecha(p.fechaVis);
end;
```

```
function cumpleFecha (p:perro): boolean;
var
  ok:boolean;

begin
  if ( ((p.fechaVis.dia >=1) and (p.fechaVis.dia <=15)) and
      ((p.fechaVis.mes =1) or (p.fechaVis.mes =2)) and
      (p.fechaVis.año = 2023) ) then
    ok:= true
  else
    ok:= false;
  cumpleFecha:= ok;
end;

Var
  ani:perro;
  I,cant:integer;

Begin
  cant:=0;

  for i:= 1 to 10 do
    begin
      leer (ani);
      if (cumpleFecha (ani) = true) then
        cant:= cant + 1;
    end;

  write (`La cantidad es` , cant);
End.
```

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
Program uno;  
Type  
fecha = record  
  dia: integer;  
  mes:integer;  
  año:integer;  
end;  
  
perro = record  
  raza: string;  
  nombre: string;  
  edad: integer;  
  fechaVis:fecha;  
end;  
// módulos  
Var  
  ani: perro;  
  cant,i: integer;
```

Begin

cant:=0;

for i:= 1 to 10 do
begin

leer (ani);

if (cumpleFecha (ani.fechaVis)=true) then
 cant:= cant + 1;

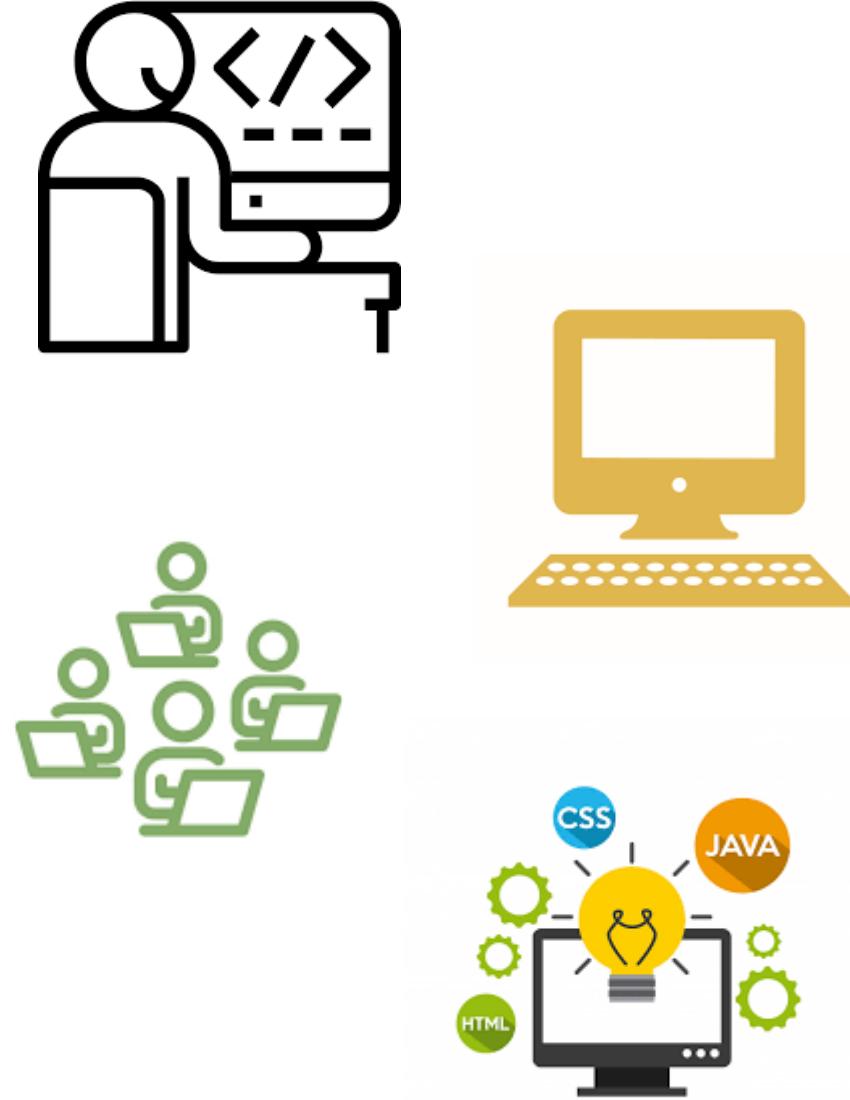
end;

write (`La cantidad es` , cant);

End.



```
function cumpleFecha (f:fecha): boolean;  
var  
    ok:boolean;  
  
begin  
    if ( ((f.dia >=1) and (f.dia <=15)) and  
        ((f.mes = 1) or (f.mes = 2)) and  
        (f.año = 2023) ) then  
        ok:= true  
  
    else  
        ok:= false;  
    cumpleFecha:= ok;  
end;
```



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



Corte de control con REGISTROS



Supongamos que se quiere representar la información de las distintas razas de animales que existen en una veterinaria. Para simplificar el problema supongamos que la veterinaria atiende solamente perros. De cada animal se conoce la raza, el nombre, la edad.

Se pide realizar un programa que lea perros hasta leer uno de raza 'XXX' y al final informe la cantidad de perros de cada raza.



**Cómo se
resuelve?**



Se pide realizar un programa que lea perros hasta leer uno de raza `XXX` y al final informe la cantidad de perros de cada raza.

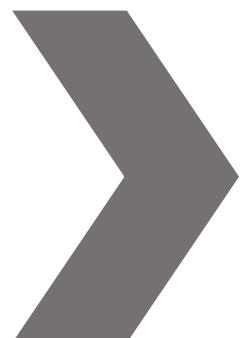
Raza `Ovejero`
Nombre `Bob`
edad:2

Raza `Callejero`
Nombre `Bob`
edad:1

Raza `Golden`
Nombre `Aragon`
edad:5

Raza `Callejero`
Nombre `Laly`
edad:4

Raza `XXX`
Nombre `Laly`
edad:4



**Ovejero 1
Callejero 2
Golden 1**

Qué necesito?

CADP – ESTRUCTURA DE DATOS

REGISTRO



Se pide realizar un programa que lea perros hasta leer uno de raza `XXX` y al final informe la cantidad de perros de cada raza.

Raza `Callejero`
Nombre `Bob`
edad:1

Raza `Callejero`
Nombre `Laly`
edad:4



Callejero 2

Raza `Golden`
Nombre `Aragon`
edad:5



Golden 1

Raza `Ovejero`
Nombre `Bob`
edad:2



Ovejero1

Raza `XXX`
Nombre `Laly`
edad:4

NECESITO

**Que los datos vengan
ORDENADOS**



Se pide realizar un programa que lea perros hasta leer uno de raza `XXX` y al final informe la cantidad de perros de cada raza.

```
Inicializar contador (cant)
Leer registro (ani)
Mientras (no sea la condición de fin) do
begin
    guardar la raza actual
    mientras (sea la misma raza) do
        begin
            contar perro
            Leer registro (ani)
        end;
    Write (`La cantidad es`, cant);
end;
```

Cuál es la estructura de datos?

Cómo se que la raza es la misma?

Donde se inicializa el contador?

CADP – ESTRUCTURA DE DATOS

REGISTRO



```
Program uno;  
Type  
perro = record  
    raza: string;  
    nombre: string;  
    edad: integer;  
end;  
  
// módulos  
  
Var  
    ani: perro;  
    cant: integer;  
    actual:string;
```

Begin

Dónde inicializo cant?
Dónde inicializo actual?
En el write es lo mismo informar actual que ani.raza?

leer (ani);
while (ani.raza <> 'XXX') do
begin
 cant:=0;
 actual:= ani.raza;
 while ((ani.raza <> 'XXX')and(ani.raza = actual)) do
begin
 cant:= cant + 1;
 leer (ani);
end;
write (`La cantidad de la raza`,actual, 'es', cant);
end;
End.

Qué modiflico si quiero informar la cantidad total y la por raza?

CADP – ESTRUCTURA DE DATOS

REGISTRO



Program uno;

Type

```
perro = record  
    raza: string;  
    nombre: string;  
    edad: integer;  
end;
```

// módulos

Var

```
    ani: perro;  
    cant: integer;  
    actual:string;  
    total:integer;
```

Begin
total:=0;

Dónde inicializo
cant?

```
leer (ani);  
while (ani.raza <> 'XXX') do  
begin
```

cant:=0;

actual:= ani.raza;

```
while ((ani.raza <> 'XXX')and(ani.raza = actual)) do  
begin
```

cant:= cant + 1;

leer (ani);

end;

total:= total + cant;

write (`La cantidad de la raza` ,actual, 'es' , cant);

end;

Write ('La cantidad total de perros es',total);

End.

Dónde inicializo
actual?

Dónde inicializo
total?

Qué modiflico si quiero
informar la raza con
mayor cantidad de
perros?

CADP – ESTRUCTURA DE DATOS

REGISTRO



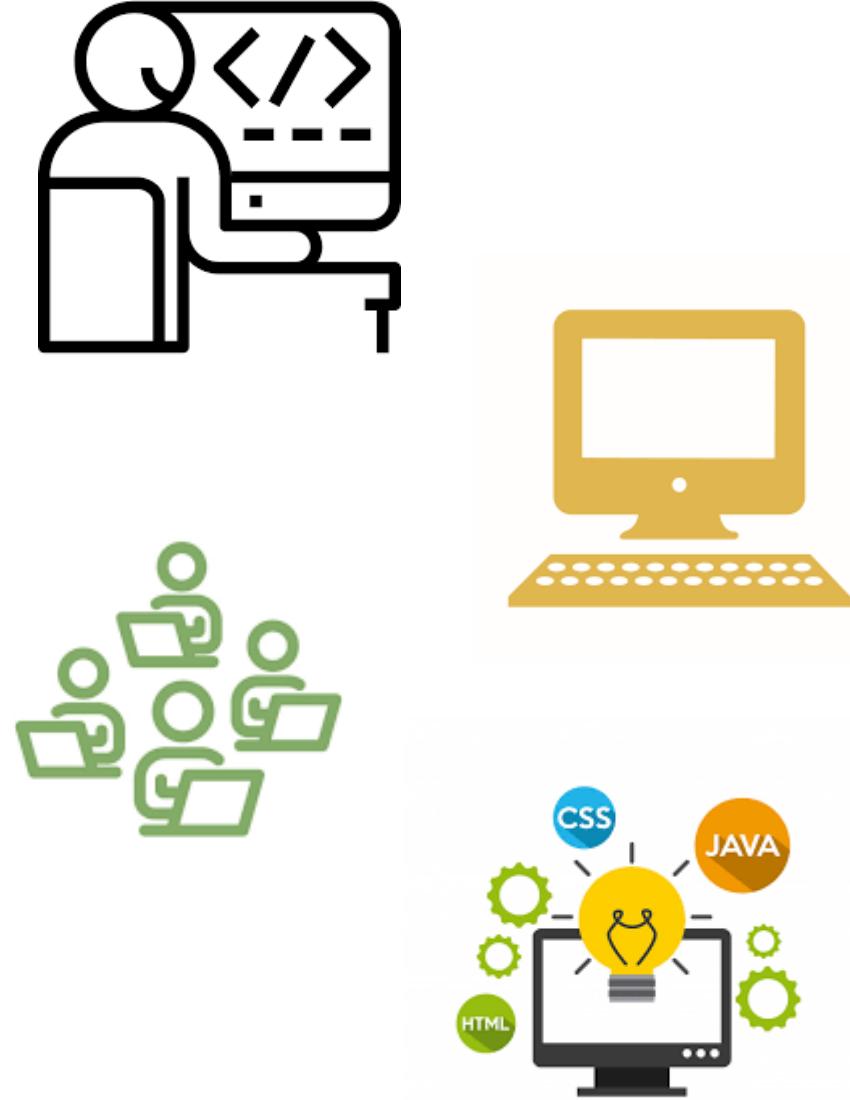
```
Program uno;
Type
  perro = record
    raza: string;
    nombre: string;
    edad: integer;
  end;
// módulos
Var
  ani: perro;
  cant: integer;
  actual:string;
  total:integer;
  max:integer;
  razaMax:string;
```

Begin
max:=-1;

leer (ani);
while (ani.raza <> 'XXX') do
begin
 cant:=0;
 actual:= ani.raza;
 while ((ani.raza <> 'XXX')and(ani.raza = actual)) do
 begin
 cant:= cant + 1;
 leer (ani);
 end;
 if (cant >= max) then begin
 max:= cant;
 razaMax:= actual;
 end;
end;
Write ('La raza con más perros es',razaMax);
End.

Dónde inicializo max?

Dónde inicializo cant?



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



- Estructura de datos ARREGLO
- Definición y características
- Operaciones esenciales



SIMPLE: aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.

TIPO DE DATO

COMPUESTO: pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.

SIMPLE

DEFINIDO POR EL LENGUAJE

Integer
Real
Char
Boolean

DEFINIDO POR EL PROGRAMADOR

Subrango

COMPUESTO

DEFINIDO POR EL LENGUAJE

String

DEFINIDO POR EL PROGRAMADOR

Registros
Arreglos

Supongamos que se quiere leer la edad de 10 personas y al finalizar informar cuantas veces apareció la edad máxima.

Edades Leídas

20

77

68

2

77

23

4

15

15

3



Máximo 77

Y ahora que se que la edad máxima es 77 cómo informo cuantas veces apareció?

Con lo que sabemos hasta ahora tenemos dos alternativas

Supongamos que se quiere leer la edad de 10 personas y al finalizar informar cuantas veces apareció la edad máxima.

Solución 1

Ingresar los valores.

Calcular el máximo.

Ingresar los valores nuevamente e imprimir cuáles coinciden con el máximo calculado.

PROBLEMA: se debe garantizar que el usuario ingrese los mismos valores. Cuantos mas valores se lean el problema es mas grande.

Solución 2

Ingresar los valores y guardar cada valor en una variable.

Calcular el máximo.

Comparar cada variable con el máximo calculado.

PROBLEMA la cantidad de variables a usar, la legibilidad del programa. Cuantos mas valores se lean el problema es mas grande

SOLUCION?

Supongamos que se quiere leer la edad de 10 personas y al finalizar informar cuantas veces apareció la edad máxima.



Disponer de alguna **ESTRUCTURA** donde almacenar los números, para luego calcular el máximo, y así finalmente poder compararlo contra los valores almacenados.

Ler los números y almacenarlos

20	77	68	2	77	23	4	15	15	3
----	----	----	---	----	----	---	----	----	---

Recorrer la estructura y obtener el máximo

20	77	68	2	77	23	4	15	15	3	77
----	----	----	---	----	----	---	----	----	---	----

Recorrer la estructura y comparar con el máximo

20	77	68	2	77	23	4	15	15	3	2
----	----	----	---	----	----	---	----	----	---	---



ARREGLO

Un arreglo (ARRAY) es una estructura de datos compuesta que permite acceder a cada componente por una variable índice, que da la posición de la componente dentro de la estructura de datos.

ARREGLO



INDICE

COMPONENTE



VECTOR (arreglo de una dimensión)

Es una colección de elementos que se guardan consecutivamente en la memoria y se pueden referenciar a través de un índice.

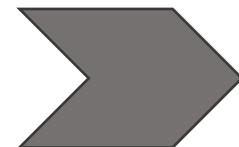
Cómo se
declara?

HOMOGENEA



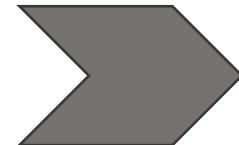
Los elementos pueden ser del mismo tipo .

ESTATICA



El tamaño no cambia durante la ejecución (se calcula en el momento de compilación)

INDEXADA



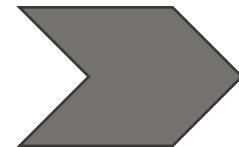
Para acceder a cada elemento de la estructura se debe utilizar una variable ‘índice’ que es de tipo ordinal.



VECTOR (arreglo de una dimensión)

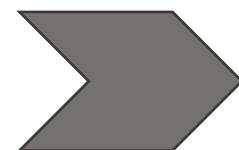
Es una colección de elementos que se guardan consecutivamente en la memoria y se pueden referenciar a través de un índice.

HOMOGENEA



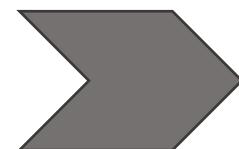
Los elementos pueden ser del mismo tipo .

ESTATICA



El tamaño no cambia durante la ejecución (se calcula en el momento de compilación)

INDEXADA



Para acceder a cada elemento de la estructura se debe utilizar una variable ‘índice’ que es de tipo ordinal.



VECTOR

Program uno;
Const

....

Type

vector = array [rango] of tipo;

Var

variable: vector;



El **rango** debe ser un tipo ordinal
char, entero, booleano, subrango

Unos

ejemplos ...



El **tipo** debe ser un tipo estático
char, entero, booleano, subrango, real
registro, vector

type

```

numeros = array [1..10] of real;
frecuen = array [char] of real;
otros = array ['h'..'m'] of integer;

```

Cómo
trabajamos?

Var

num: numeros; num reserva memoria para 10 números reales

15,25	-7	179,3	0	8,45	10,25	9	8,45	10,5	9
1	2	3	4	5	6	7	8	9	10

nuevo: frecuen; nuevo reserva memoria para 256 números reales

15,25	-7,5	179,3							19
A									z

otro: otros; otro reserva memoria para 6 números enteros

15	-7	1879	0	8	10
h	i	j	k	l	m

Carga de valores

Lectura / Escritura

Recorridos

Agregar elementos al final

Insertar elementos

Borrar elementos

Búsqueda de un elemento

Ordenación de los elementos





CON LA VARIABLE VECTOR

Program uno;

Const

....

Type

```
vector = array [1..10] of integer;
```

Var

```
v1,v2: vector;
```

Begin

....

```
v2:= v1;
```

...

End.



**La única operación permitida
es la asignación entre dos
variables del mismo tipo**



Program uno;

Const

...

Type

vector = array [1..10] of integer;

Var

v1,v2: vector;

Begin

...

End.



La única forma de acceder a los elementos es utilizando un índice

variable [pos]

variable [4]

Las operaciones con el elemento son las permitidas para el tipo de datos del elemento

CADP – TIPO DE DATOS VECTOR - CARGA



Program uno;

Const

....

Type

```
vector = array [1..10] of integer;
```

Var

```
v1: vector;
```

	?	?	?	?	?	?	?	?	?	?
v1	1	2	3	4	5	6	7	8	9	10

Begin

```
v1[1]:= 4;
```

```
v1[3]:= 8;
```

End.

	4	?	?	?	?	?	?	?	?	?
v1	1	2	3	4	5	6	7	8	9	10

	4	?	8	?	?	?	?	?	?	?
v1	1	2	3	4	5	6	7	8	9	10

Cómo se carga
completo?

CADP – TIPO DE DATOS VECTOR - CARGA



Program uno;

```
Const  
  tam = 10;
```

Type

```
  vector = array [1..tam] of integer;
```

Var

```
  v1:vector;  
  i,valor:integer;
```

v1	?	?	?	?	?	?	?	?	?	?
	1	2	3	4	5	6	7	8	9	10

Begin

```
  for i:= 1 to tam do  
    begin  
      read (valor);  
      v1[i]:= valor;  
    end;
```

End.

v1	-1	18	4	0	5	57	-2	3	8	4
	1	2	3	4	5	6	7	8	9	10



No se puede hacer
read(v1)

Cómo se
modulariza?

ALTERNATIVA

```
Begin  
  for i:= 1 to tam do  
    begin  
      read(v1[i]);  
    end;  
End.
```

CADP – TIPO DE DATOS VECTOR - CARGA



Procedure carga (var v: vector);

var
i,valor:integer;

begin
for i:= 1 to tam do
begin
read (valor);
v[i]:= valor;
end;
end;

v	?	?	?	?	?	?	?	?	?	?
	1	2	3	4	5	6	7	8	9	10

v	-1	18	4	0	5	57	-2	3	8	4
	1	2	3	4	5	6	7	8	9	10

ALTERNATIVA

Procedure carga (var v: vector);
var
i:integer;
begin
for i:= 1 to tam do
read (v[i]);
end;

Puede ser una función? Se puede utilizar tam?

Cómo muestro los datos?

CADP – TIPO DE DATOS VECTOR - MUESTRA



Program uno;

```
Const  
  tam = 10;
```

Type

```
  vector = array [1..tam]  of  integer;
```

Var

```
  v1:vector;
```

```
  i,valor:integer;
```

Begin

```
  carga (v1);
```

```
  for i:= 1 to tam do
```

begin

```
      valor:= v1[i];  
      write (valor);
```

end;

End.



No se puede hacer
`write(v1)`

	?	?	?	?	?	?	?	?	?	?
v1	1	2	3	4	5	6	7	8	9	10

v1	1	2	3	4	5	6	7	8	9	10
	-1	18	4	0	5	57	-2	3	8	4

ALTERNATIVA

Cómo se
modulariza?

```
Begin  
  for i:= 1 to tam do  
    begin  
      write(v1[i]);  
    end;  
End.
```

CADP – TIPO DE DATOS VECTOR - CARGA



Procedure imprimir (v: vector);

```
var  
  i, valor:integer;
```

```
begin
```

```
  for i:= 1 to tam do  
    begin
```

```
      valor:= v[i];  
      write(valor);
```

```
    end;
```

```
end;
```

v	-1	18	4	0	5	57	-2	3	8	4
	1	2	3	4	5	6	7	8	9	10

ALTERNATIVA

Procedure imprimir (v: vector);

```
var  
  i:integer;
```

```
begin
```

```
  for i:= 1 to tam do  
    write (v[i]);
```

```
end;
```

Cómo solucionamos
nuestro problema inicial?

Puede ser
una función?

CADP – TIPO DE DATOS VECTOR -



Escriba un programa que lea 10 números enteros y al finalizar informe cuantas veces apareció el número máximo.

Cargar vector (**v**)

Calcular el máximo (**v** , **max**)

Verificar cuantas veces aparece **max** en el vector **v**

**Cómo cargo el
vector?**

**Cómo calculo el
máximo?**

**Cómo verifico cuántas
veces apareció el maximo?**

v	-1	18	4	0	5	57	-2	3	57	4
	1	2	3	4	5	6	7	8	9	10

CADP – TIPO DE DATOS VECTOR -



```
Program uno;
```

```
Const
```

```
    tam = 10;
```

	v1	?	?	?	?	?	?	?	?	?	?
		1	2	3	4	5	6	7	8	9	10

```
Type
```

```
    vector = array [1..tam]      of      integer;
```

```
Var
```

```
    v1:vector;
```

```
    i,max,cant:integer;
```

```
Begin
```

```
    carga (v1);
```

```
    max:= máximo (v1);
```

```
    cant:= verificar (v1,max);
```

```
    write (cant);
```

```
End.
```

ALTERNATIVA

```
Begin
```

```
    carga (v1);
```

```
    max:= máximo (v1);
```

```
    write(verificar (v1,max));
```

```
End.
```

CADP – TIPO DE DATOS VECTOR -



```
function máximo (v:vector):integer;
```

Var

i,max,valor:integer;

v

-1	18	4	0	5	57	-2	3	57	4
1	2	3	4	5	6	7	8	9	10

Begin

max:= -9990;

for i:= 1 to tam do
begin

valor:= v[i];

if (valor >= max) then

max:= v[i];

end;

maximo:= max;

End;

```
function máximo (v:vector):integer;
```

Var

i,max:integer;

Begin

max:= -9999;

for i:= 1 to tam do

begin

if (v[i] >= max) then

max:= v[i];

end;

maximo:= max;

End;

ALTERNATIVA

CADP – TIPO DE DATOS VECTOR -



```
function verificar (v:vector; valor:integer):integer;
```

Var

i,cant,aux:integer;

Begin

cant:= 0;

for i:= 1 to tam do

begin

aux:= v[i];

if (valor = aux) then

 cant:= cant + 1;

end;

 verificar:= cant;

End;

v

-1	18	4	0	5	57	-2	3	57	4
1	2	3	4	5	6	7	8	9	10

57

```
function verificar (v:vector; valor:integer):integer;
```

Var

i,cant:integer;

Begin

cant:= 0;

for i:= 1 to tam do

begin

 if (valor = v[i]) then

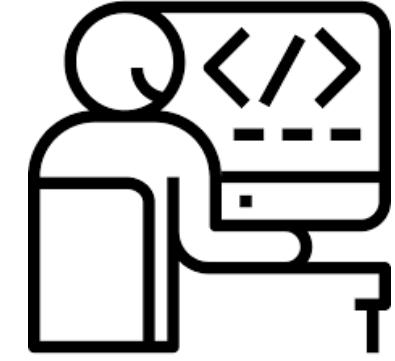
 cant:= cant + 1;

end;

 verificar:= cant;

End;

ALTERNATIVA



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



- Estructura de datos ARREGLO
- Recorridos totales
- Recorridos parciales



RECORRIDOS

Consiste en recorrer el vector de manera total o parcial, para realizar algún proceso sobre sus elementos.

RECORRIDO - TOTAL

Implica analizar todos los elementos del vector, lo que lleva a recorrer completamente la estructura.

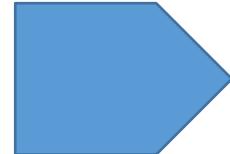
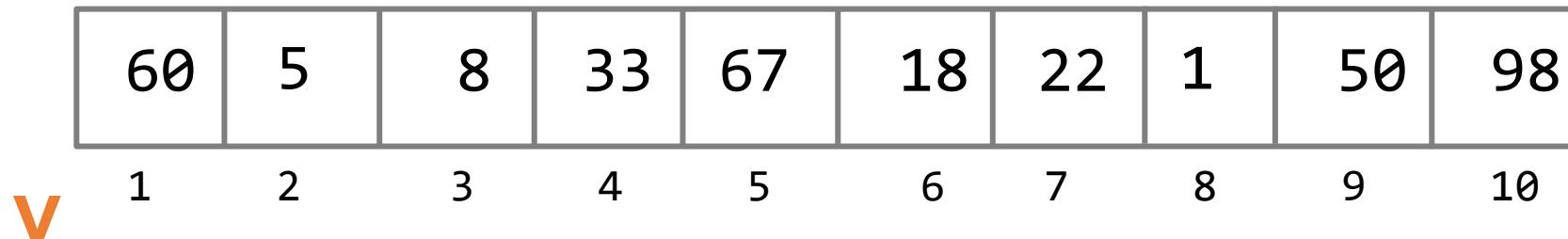
Qué estructura de control implica cada uno?

RECORRIDO - PARCIAL

Implica analizar los elementos del vector, hasta encontrar aquel que cumple con lo pedido. Puede ocurrir que se recorra todo el vector



Realice un programa que llene un vector de 10 elementos enteros positivos y luego informe la cantidad de números múltiplos de 3. Suponga que los nros leídos son positivos.

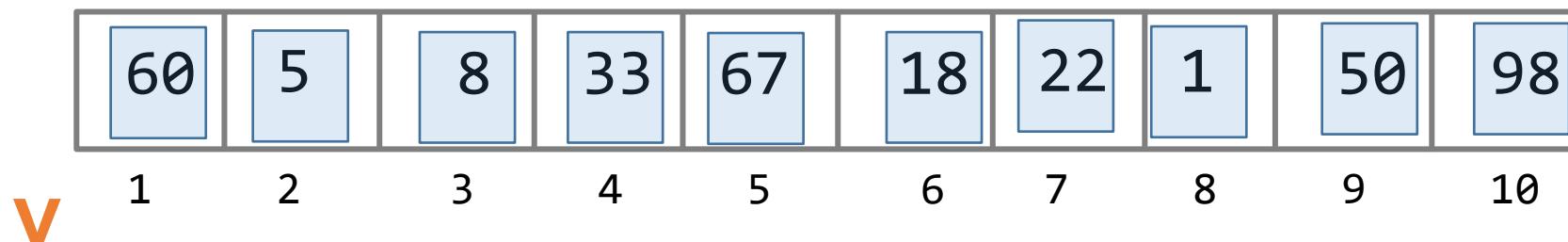


Cant 0

Cant 1

Cant 2

Cant 3



Qué estructura
de control?

Qué
modularizo?

Cómo lo
implemento?



Program uno;

Const

 tam=10;

 multi=3;

Type

 vector = array [1..tam] of integer;

Var

 v1:vector;

 cant:integer;

Begin

cargar (v1);

 cant:= **múltiplos** (v1);

 write (“La cantidad de múltiplos de”, multi, “es”, cant);

End.



```
Procedure cargar (var v:vector);
Var
  i,valor:integer;
Begin
  for i:= 1 to tam do
    begin
      read(valor);
      v[i]:= valor;
    end;
End;
```

ALTERNATIVA

```
Procedure cargar (var v:vector);
Var
  i:integer;
Begin
  for i:= 1 to tam do
    begin
      read(v[i]);
    end;
End;
```

```
function multiplos (V:vector):integer;
```

Var

i,cant,resto: integer;



Begin

cant:=0;

for i:= 1 to tam do

begin

resto:= V[i] MOD multi;

if (resto = 0) then

 cant:= cant + 1;

end;

multiplos:= cant;

End;

ALTERNATIVA

```
function multiplos (v:vector):integer;
```

Var

i,cant: integer;

Begin

cant:=0;

for i:= 1 to tam do

begin

 if ((v[i] MOD multi) = 0) then

 cant:= cant + 1;

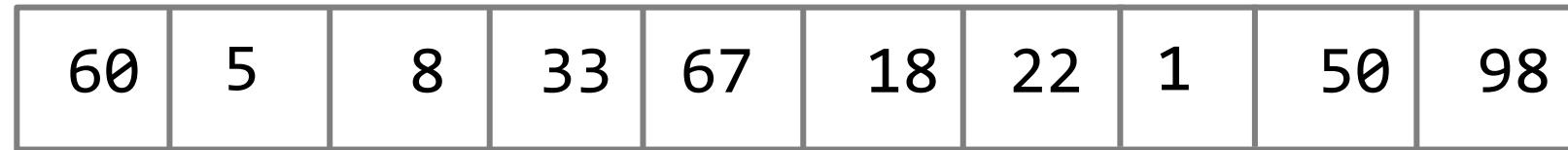
end;

multiplos:= cant;

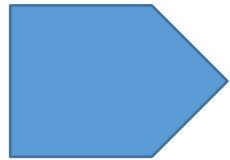
End;



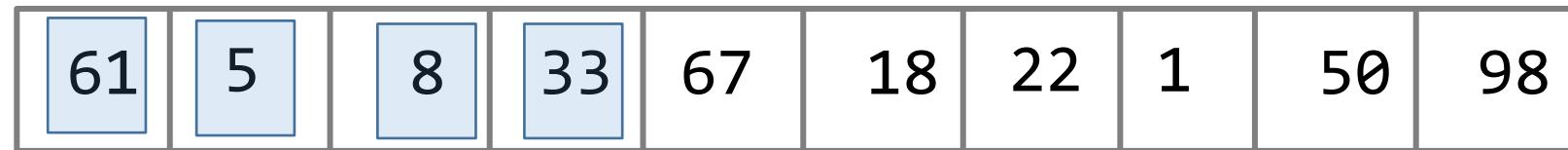
Realice un programa que llene un vector de 10 elementos enteros positivos y luego informe la primer posición donde aparece un múltiplos de 3. Suponga que los nros leídos son positivos y que existe al menos un múltiplo de 3.



V



POS 4

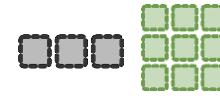


V

Qué estructura
de control?

Qué
modularizo?

Cómo lo
implemento?



Program uno;

Const

 tam=10;

 multi=3;

Type

 vector = array [1..tam] of integer;

Var

 v:vector;

 pos:integer;

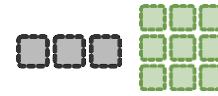
Begin

cargar (v);

 pos:= **posicion** (v);

 write (“La posición del primer múltiplo de”, multi, “es”, pos);

End.



```
function posicion (v: vector): integer;
```

```
var
```

```
    pos,resto:integer;  
    seguir:boolean;
```

```
begin
```

```
    seguir:= true; pos:=1;
```

```
    while (seguir = true) do
```

```
        begin
```

```
            resto:= v[pos] MOD multi;
```

```
            if (resto = 0) then
```

```
                seguir:= false
```

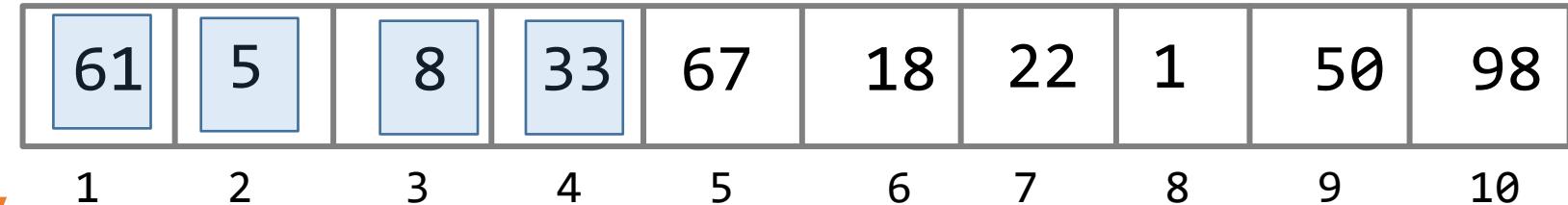
```
            else
```

```
                pos:= pos + 1;
```

```
            end;
```

```
            posicion:= pos;
```

```
        end;
```



Por qué se

inicializa pos en 1?

Por qué pos se

incrementa en el else?

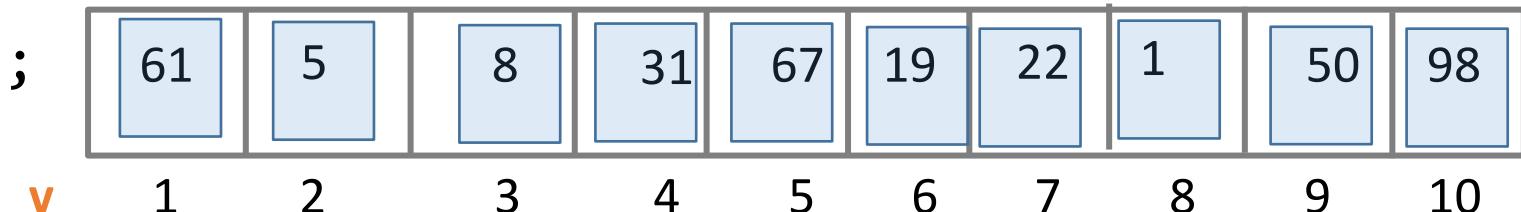
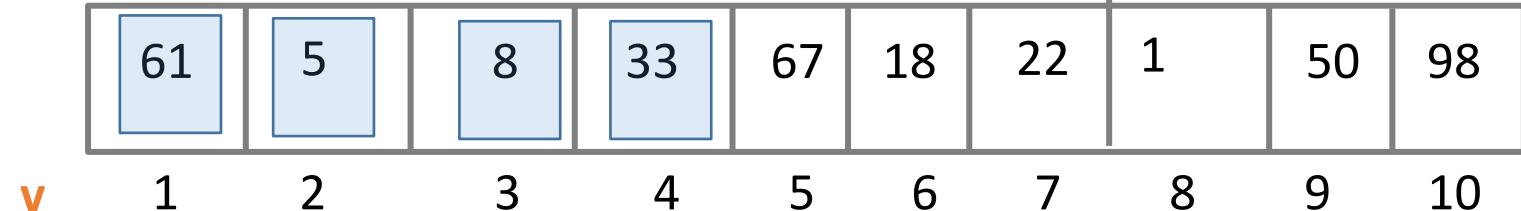
Qué cambio si el
enunciado no asegura
que haya al menos un
múltiplo de 3?

CADP – VECTOR

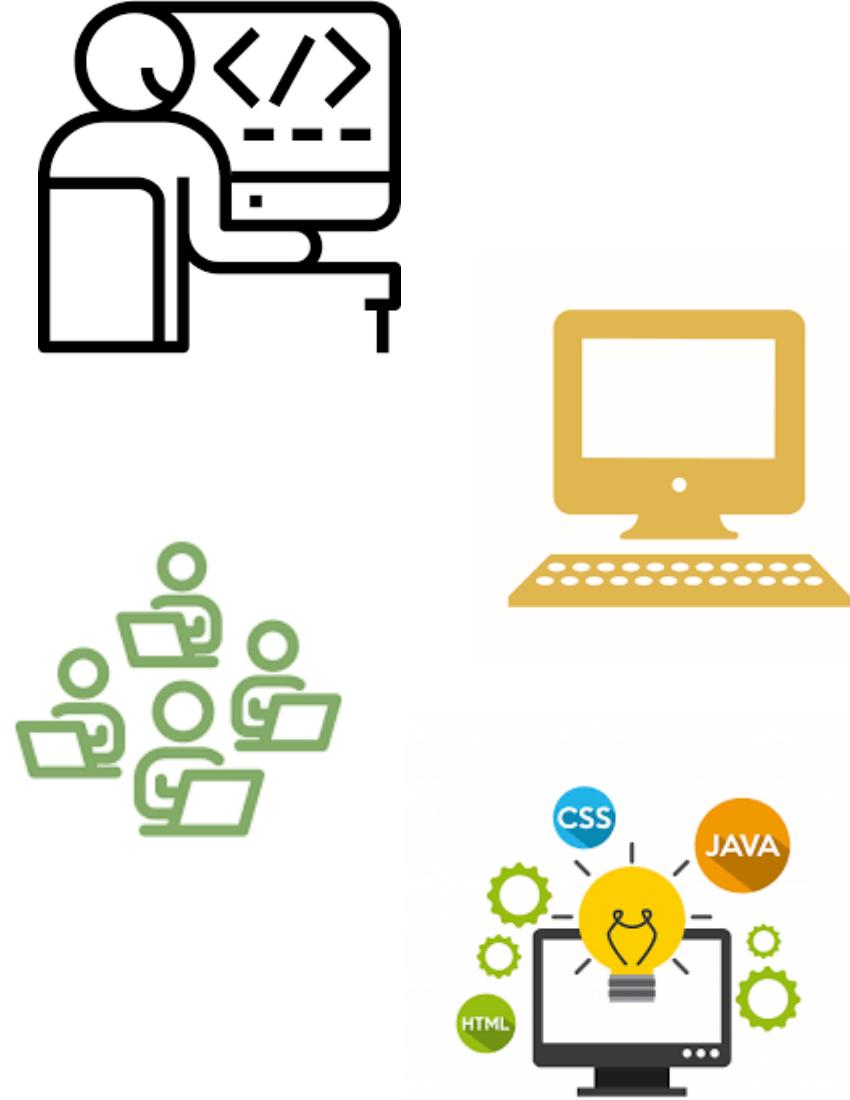
RECORRIDO PARCIAL



```
function posicion (v: vector): integer;  
  
var  
  pos,resto:integer;  
  seguir:boolean;  
  
begin  
  seguir:= true; pos:=1;  
  while ((pos<= tam) and (seguir = true)) do  
    begin  
      resto:= v[pos] MOD multi;  
      if (resto = 0) then  
        seguir:= false  
      else  
        pos:= pos + 1;  
    end;  
    if (seguir = false) then posicion:= pos  
    else posicion:= -1;  
end;
```



Es necesario la
última condición
del if?



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



Estructura de datos ARREGLO



Dimensión física y dimensión lógica

Carga de valores

Lectura / Escritura

Recorridos

Dimensión física y lógica

Agregar elementos al final

Insertar elementos

Borrar elementos

Búsqueda de un elemento

Ordenación de los elementos



Supongamos que se existe un vector cargado de 10 elementos como máximo, pero por alguna circunstancia se cargaron sólo los primeros 4 valores.

20	77	68	2	?	?	?	?	?	?
----	----	----	---	---	---	---	---	---	---

a

Supongamos que sin saber que esto ocurrió se imprime el contenido del vector:

```
for i:= 1 to 10 do  
    write (a[i]);
```

Que se
obtendrá con la
 impresión?



DIMENSION FISICA

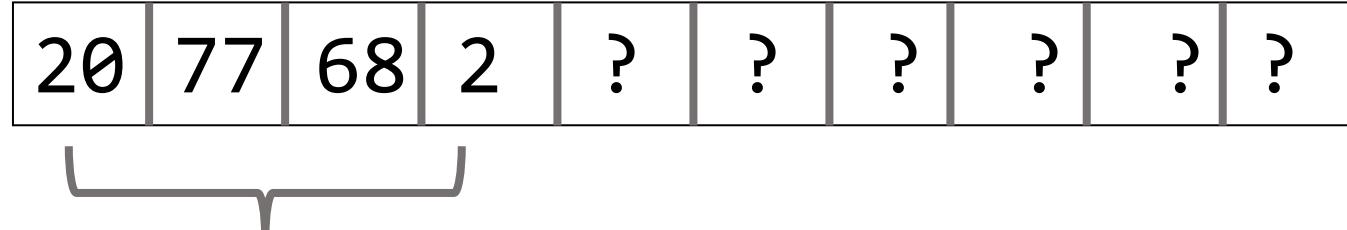
Se especifica en el momento de la declaración y determina su ocupación máxima de memoria.

La cantidad de memoria total reservada no variará durante la ejecución del programa.

DIMENSION LOGICA

Se determina cuando se cargan contenidos a los elementos del arreglo.

Indica la cantidad de posiciones de memoria ocupadas con contenido real. Nunca puede superar la dimensión física.

a

DIMENSION FISICA

Es la cantidad máxima de elementos que se pueden guardar en el arreglo.
No puede modificarse durante la ejecución del programa

DIMENSION LOGICA

Es la cantidad de elementos reales que se guardan en el arreglo.

Puede modificarse durante la ejecución del programa

Nunca puede ser mayor a la dimensión física (se debe controlar)

Cuándo se determina cada una?
Donde se declaran?

CADP – TIPOS DE DATOS

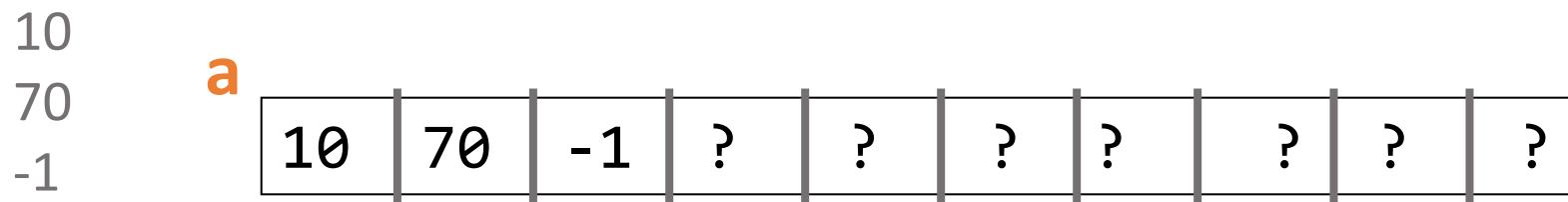


VECTORES

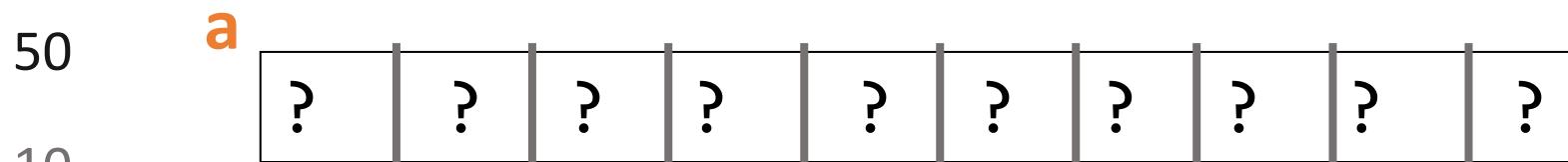


Realizar un programa que cargue un arreglo con números enteros hasta leer el número 50, a lo sumo se cargan 10 números.

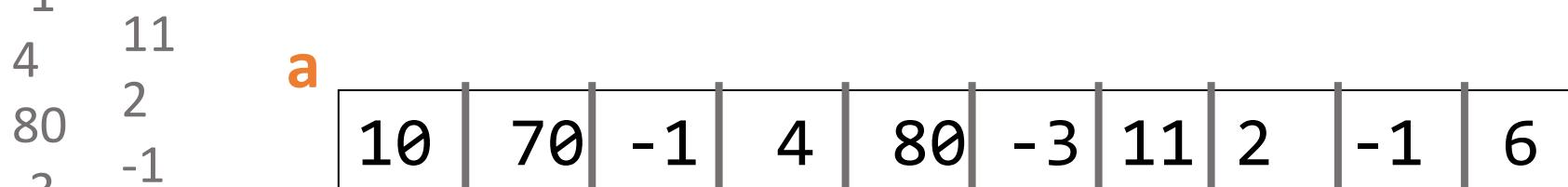
Luego de terminar la carga informe cuál es el número mas grande de los leídos.



$$\begin{aligned}DF &= 10 \\ DL &= 3\end{aligned}$$



$$\begin{aligned}DF &= 10 \\ DL &= 0\end{aligned}$$



$$\begin{aligned}DF &= 10 \\ DL &= 10\end{aligned}$$



Realizar un programa que cargue un arreglo con números enteros hasta leer el número 50, a lo sumo se cargan 10 números.

Luego de terminar la carga informe cuál es el número mas grande de los leídos.

Program uno;

Const

DF = 10

Type

valores = array [1..DF] of integer;

Var

v: valores;
max:integer;
dL:integer;

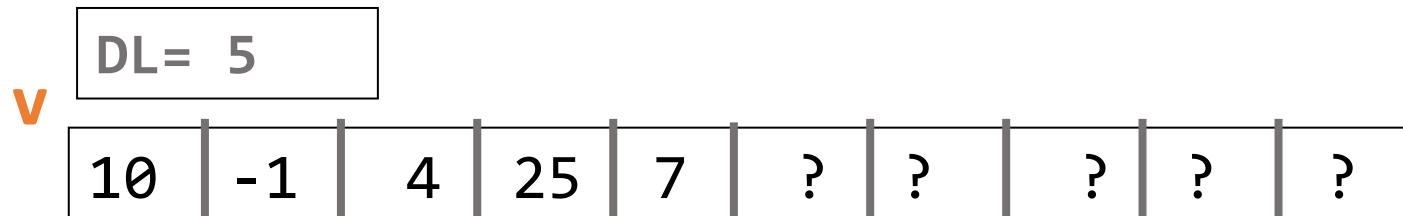
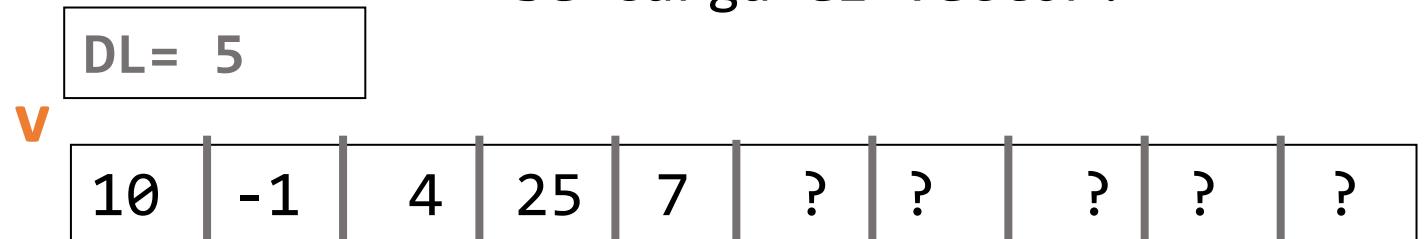
Begin

cargarValores (v , dL);
max:= maximo (v , dL);

End.

La dimensión física es una constante.

La dimensión lógica es una variable y toma valor cuando se carga el vector.



```
Procedure cargarValores (var a: números; var dimL:integer);
```

```
Var  
    num:integer;
```

```
Begin  
    dimL:=0;  
    read (num);  
    while (num <> 50) do  
        begin  
            a[dimL]:= num;  
            read(num);  
        end;  
    End;
```

Es correcto?



Cómo dimL, está inicializado en 0, la primera vez se accede a la posición a[0] y no es válida

```
Procedure cargarValores (var a: números; var dimL:integer);
```

```
Var  
    num:integer;  
  
Begin  
    dimL:=1;  
    read (num);  
    while (num <> 50) do  
        begin  
            a[dimL]:= num;  
            read(num);  
        end;  
    End;
```

Es correcto?



Cómo dimL, nunca se incrementa, entonces carga siempre en la misma posición a[1]

```
Procedure cargarValores (var a: números; var dimL:integer);
```

```
Var  
    num:integer;
```

```
Begin  
    dimL:=1;  
    read (num);  
    while (num <> 50) do  
        begin  
            a[dimL+1]:= num;  
            read(num);  
        end;  
    End;
```

Es correcto?



Cómo dimL, nunca se incrementa, entonces carga siempre en la misma posición a[1]

```
Procedure cargarValores (var a: números; var dimL:integer);
```

```
Var  
    num:integer;
```

```
Begin  
    dimL:=1;  
    read (num);  
    while (num <> 50) do  
        begin  
            a[dimL]:= num;  
            dimL:= dimL+1;  
            read(num);  
        end;  
    End;
```

Es correcto?



Si el primer número leído es 50, no entra al while, y como dimL está inicializado en 1, entonces devuelve que se cargó un elemento

```
Procedure cargarValores (var a: números; var dimL:integer);
```

```
Var  
    num:integer;  
  
Begin  
    dimL:=0;  
    read (num);  
    while (num <> 50) do  
        begin  
            dimL:= dimL+1;  
            a[dimL]:= num;  
            read(num);  
        end;  
    End;
```

Es correcto?



Qué pasa si leo mas de 10
números (el valor 50 no
apareció y ya lei 10 valores)

```
Procedure cargarValores (var a: números; var dimL:integer);
```

```
Var
```

```
    num:integer;
```

```
Begin
```

```
    dimL:=0;
```

```
    read (num);
```

```
    while ((dimL < dF) and (num <> 50)) do
```

```
        begin
```

```
            dimL:= dimL+1;
```

```
            a[dimL]:= num;
```

```
            read(num);
```

```
        end;
```

```
    End;
```

Es correcto?



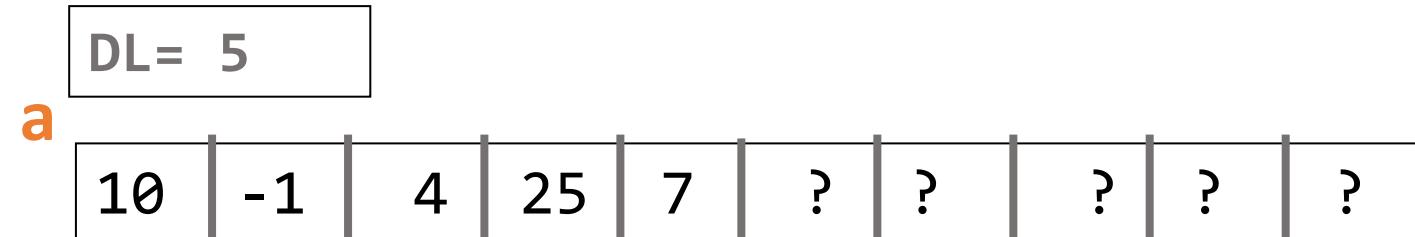
SI!!!!

```
function maximo (a: números; dimL:integer):integer;
```

```
Var  
    max,i:integer;
```

```
Begin  
    max:=-9999;  
    for i:= 1 to dF do  
        begin  
            if (a[i]>= max) then max:= a[i];  
        end;
```

```
    maximo:= max;  
End;
```



Es correcto?



NO! Sólo hay que recorrer
hasta la cantidad de elementos
cargados realmente



```
function maximo (a: números; dimL:integer):integer;
```

```
Var
```

```
  max,i:integer;
```

```
Begin
```

```
  max:=-9999;
```

```
  for i:= 1 to dimL do
```

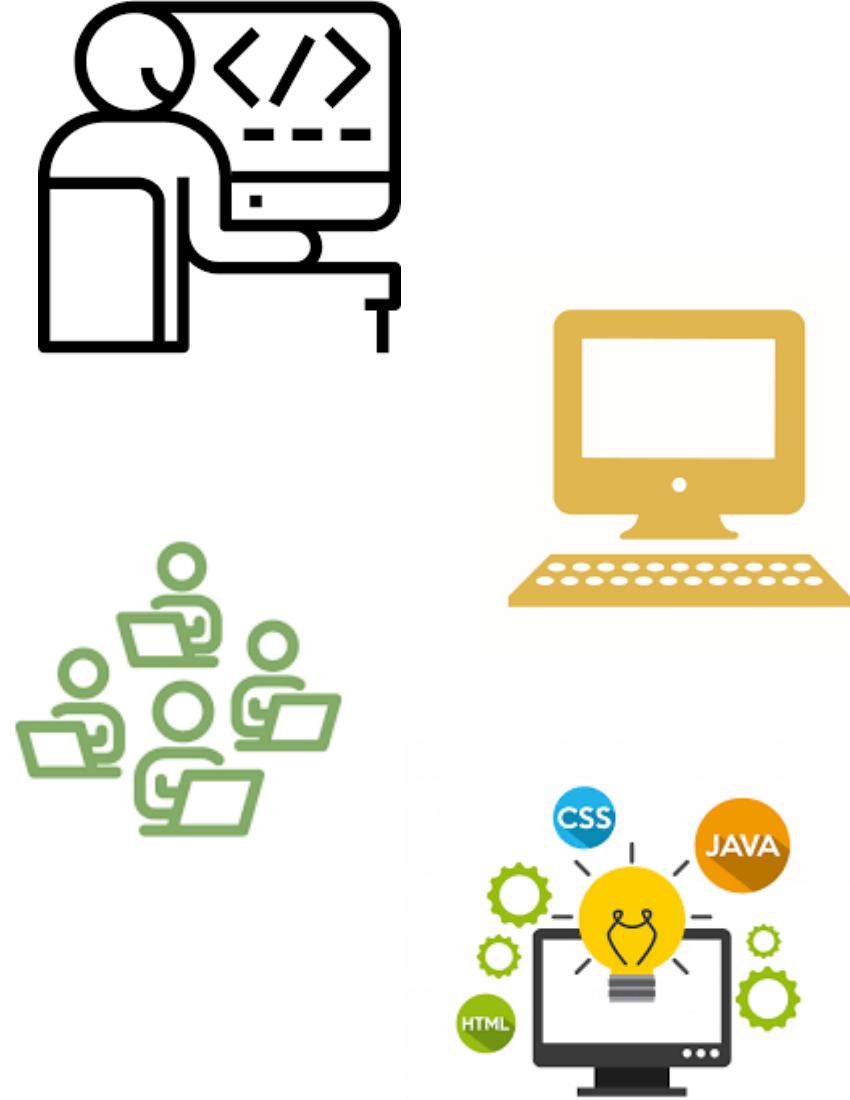
```
    begin
```

```
      if (a[i]>= max) then max:= a[i];
```

```
    end;
```

```
  maximo:= max;
```

```
End;
```



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



- Estructura de datos ARREGLO
- Agregar elementos
- Insertar elementos
- Eliminar elementos

Carga de valores

Lectura / Escritura

Recorridos

Dimensión física y lógica

Agregar elementos

Insertar elementos

Borrar elementos

Búsqueda de un elemento





Significa agregar en el vector un elemento detrás del último elemento cargado en el vector. Puede pasar que esta operación no se pueda realizar si el vector está lleno

Dl = 4

45

a

34	10	-1	5						
1	2	3	4	5	6	7	8	9	10

Qué pasos
considero?



Significa agregar en el vector un elemento detrás del último elemento cargado en el vector. Puede pasar que esta operación no se pueda realizar si el vector está lleno

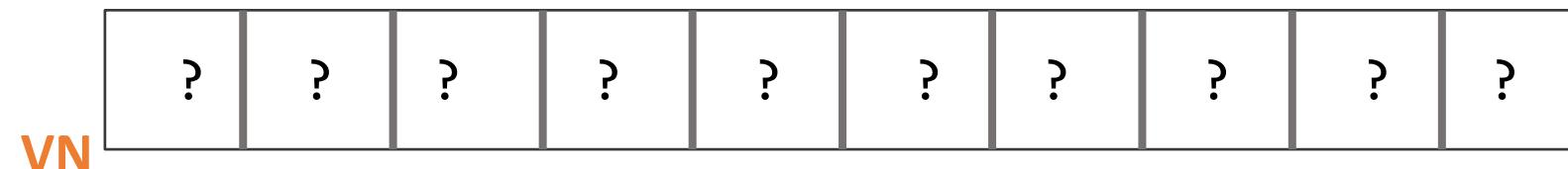
- 1- Verificar si hay espacio (cantidad de elementos actuales es menor a la cantidad de elementos posibles)
- 2- Agregar al final de los elementos ya existentes el elemento nuevo.
- 3- Incrementar la cantidad de elementos actuales.

Cómo se
implementa?

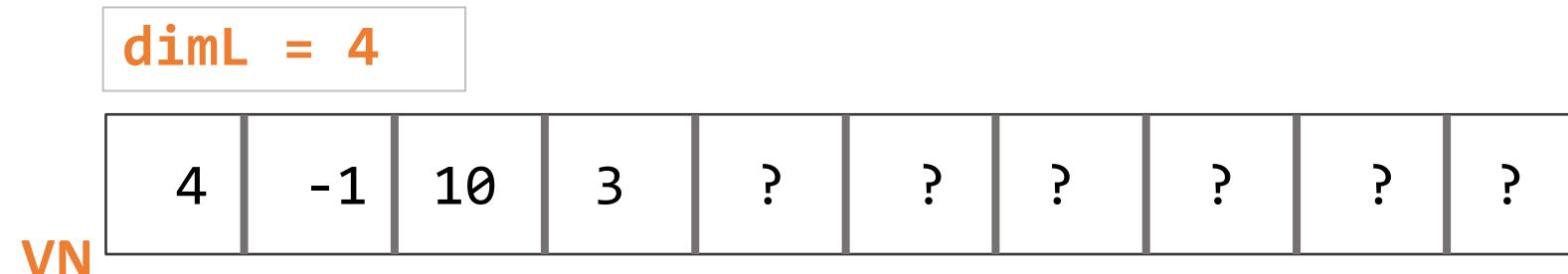


Dado un vector de números enteros (10 elementos como máximo) realice un programa que lea un nuevo número e invoque a un módulo que agregue el elemento en el vector.

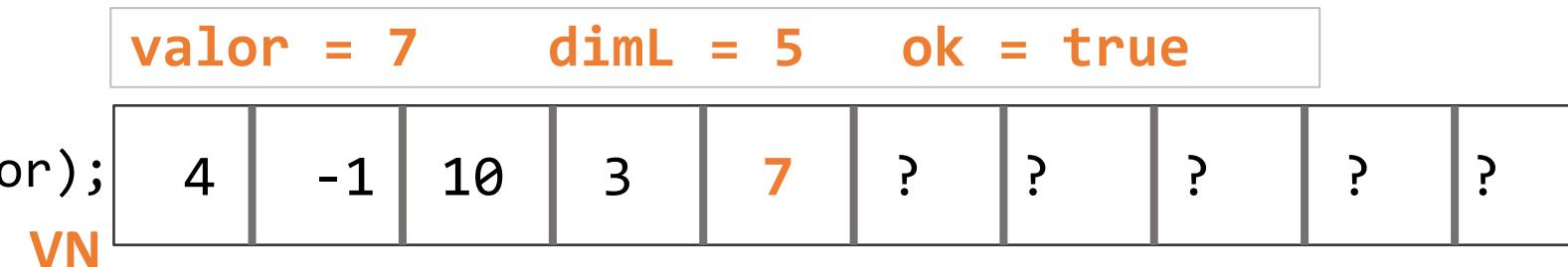
```
Program uno;
const
  fisica = 10;
type
  numeros= array [1..fisica] of integer;
```



```
var
  VN: numeros;
  dimL, valor:integer;
  ok:boolean;
```



```
Begin
  cargar (VN,dimL);
  read(valor);
  agregar(VN,dimL,ok,valor);
End.
```



```
Procedure agregar (var a :números; var dL:integer; var pude:boolean; num:integer);
```

```
Begin
```

```
    pude:= false;
```

Verifico si hay espacio



```
if ((dL + 1) <= física) then
```

```
begin
```

```
    pude:= true;  
    dL:= dL + 1;  
    a[dL]:= num;
```

```
end;
```

```
end.
```

Registro que se pudo realizar
Incremento la dimensión lógica
Agrego elelemento

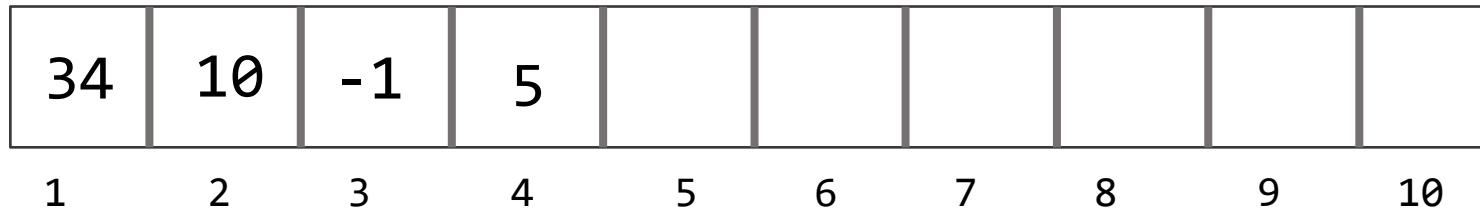


Significa agregar en el vector un elemento en una posición determinada. Puede pasar que esta operación no se pueda realizar si el vector está lleno o si la posición no es válida

D₁ = 4

45

pos = 2 ^a



Qué pasos
considero?



Significa agregar en el vector un elemento en una posición determinada. Puede pasar que esta operación no se pueda realizar si el vector está lleno o si la posición no es válida

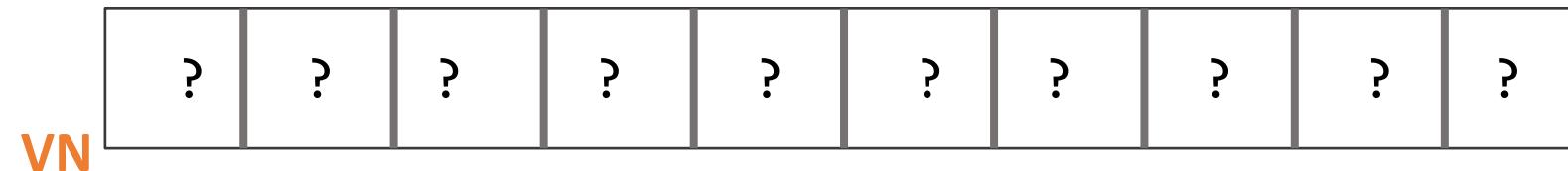
- 1- Verificar si hay espacio (cantidad de elementos actuales es menor a la cantidad de elementos posibles)
- 2- Verificar que la posición sea válida (esté entre los valores de dimensión definida del vector y la dimensión lógica).
- 3- Hacer lugar para poder insertar el elemento.
- 4- Incrementar la cantidad de elementos actuales.

**Cómo se
implementa?**

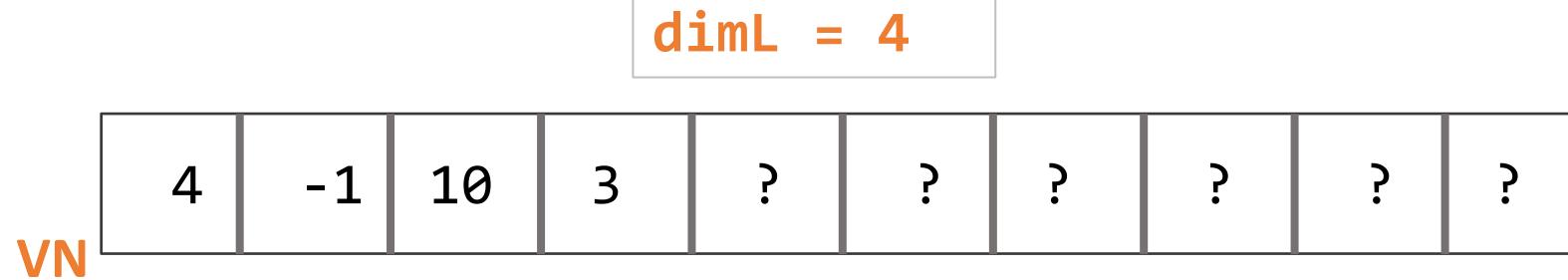


Dado un vector de números enteros (10 elementos como máximo) realice un programa que lea un nuevo número y una posición e invoque a un módulo que inserte el elemento en el vector en la posición leída.

```
Program uno;
  const
    fisica = 10;
  type
    numeros= array [1..fisica] of integer;
```



```
var
  VN: numeros;
  dimL, valor,pos:integer;
  ok:boolean;
```



```
Begin
  cargar (VN,dimL);
  read(valor); read(pos);
  insertar(VN,dimL,valor,ok,pos);
End.
```



```
Procedure insertar (var a :números; var dL:integer; var pudo:boolean;  
Var  
    i:integer;
```

```
Begin  
    pudo:= false;
```

Verifico si hay espacio y si la
posición es válida

```
if ((dL + 1) <= física) and (pos>= 1) and (pos <= dL )then begin
```

```
    for i:= dL downto pos do  
        a[i+1]:= a[i];
```

Corro los elementos empezando desde atrás hasta
la posición a insertar para hacer el hueco donde
se va a insertar el elemento

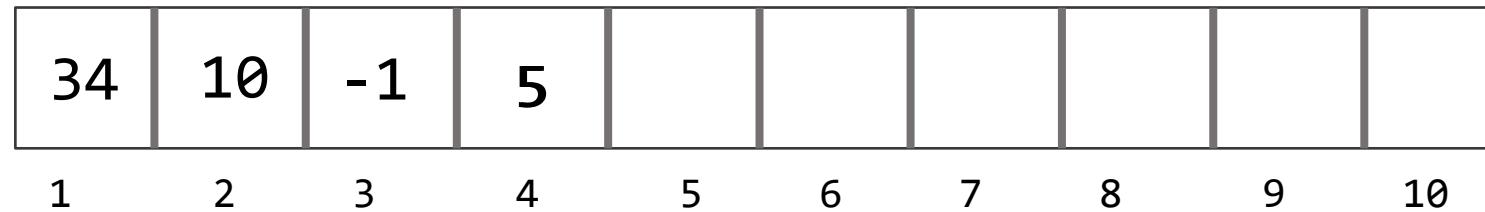
```
    pudo:= true;  
    a[pos]:= num;  
    dL:= dL + 1;  
end;
```

Registro que se pudo realizar
Inserto el elemento
Incremento la dimensión lógica



Significa borrar (lógicamente) en el vector un elemento en una posición determinada, o un valor determinado. Puede pasar que esta operación no se pueda realizar si la posición no es válida, o en el caso de eliminar un elemento si el mismo no está

pos = 2
D1 = 4



Qué pasos
considero?



Significa borrar (lógicamente) en el vector un elemento en una posición determinada, o un valor determinado. Puede pasar que esta operación no se pueda realizar si la posición no es válida, o en el caso de eliminar un elemento si el mismo no está

- 1- Verificar que la posición sea válida (esté entre los valores de dimensión definida del vector y la dimensión lógica).
- 2- Hacer el corrimiento a partir de la posición y hasta el final.
- 3- Decrementar la cantidad de elementos actuales

Cómo se
implementa?



Dado un vector de números enteros (10 elementos como máximo) realice un programa que lea una posición e invoque a un módulo que elimine el elemento en el vector en la posición leída.

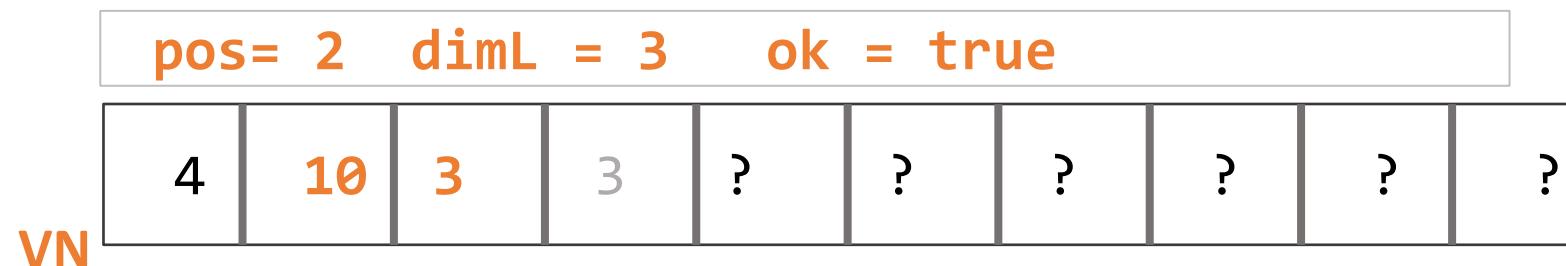
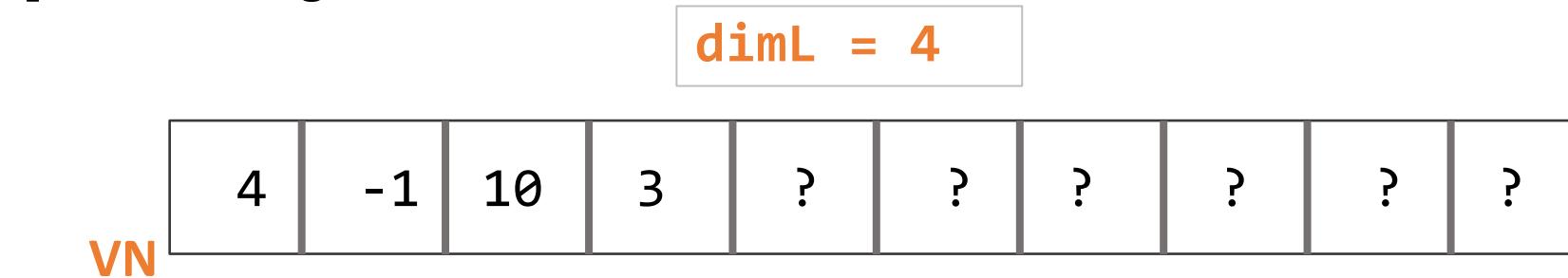
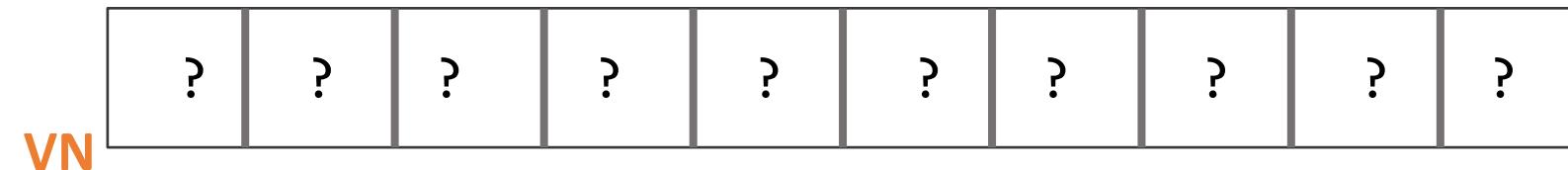
```
Program uno;
  const
    fisica = 10;
  type
    numeros= array [1..fisica] of integer;
```

```
var
  VN: numeros;
  dimL,pos:integer;
  ok:boolean;
```

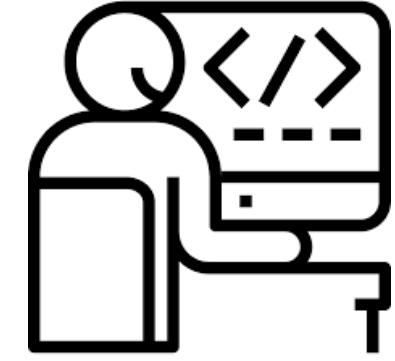
```
Begin
  cargar (VN,dimL);
  read(pos);
  eliminar(VN,dimL,ok,pos);
```

End.

Clase 7-2



```
Procedure eliminar (var a :números; var dL:integer; var pudo:boolean;pos: integer);  
Var  
  i:integer;  
  
Begin  
  pudo:= false;           Verifico si la posición es válida  
  if ((pos>= 1) and (pos <= dL) )then begin  
    for i:= pos to (dL-1) do  
      a[i]:= a[i+1];          Corro los elementos empezando desde la posición  
                                hasta la dimensión lógica-1 para “tapar” el  
                                elemento a eliminar  
    pudo:= true;  
    dL:= dL - 1;             Registro que se pudo realizar  
  end;                     Decremento la dimensión lógica  
end;
```



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



- Estructura de datos ARREGLO
- Búsqueda en un vector desordenado
- Búsqueda en un vector ordenado

Carga de valores

Lectura / Escritura

Recorridos

Dimensión física y lógica

Agregar elementos

Insertar elementos

Borrar elementos

Búsqueda de un elemento





Significa recorrer el vector buscando un valor que puede o no estar en el vector. Se debe tener en cuenta que no es lo mismo buscar en un vector ordenado que en uno que no lo este.

Vector Desordenado

- Se debe recorrer todo el vector (en el peor de los casos), y detener la búsqueda en el momento que se encuentra el dato buscado o en el que se terminó el vector.

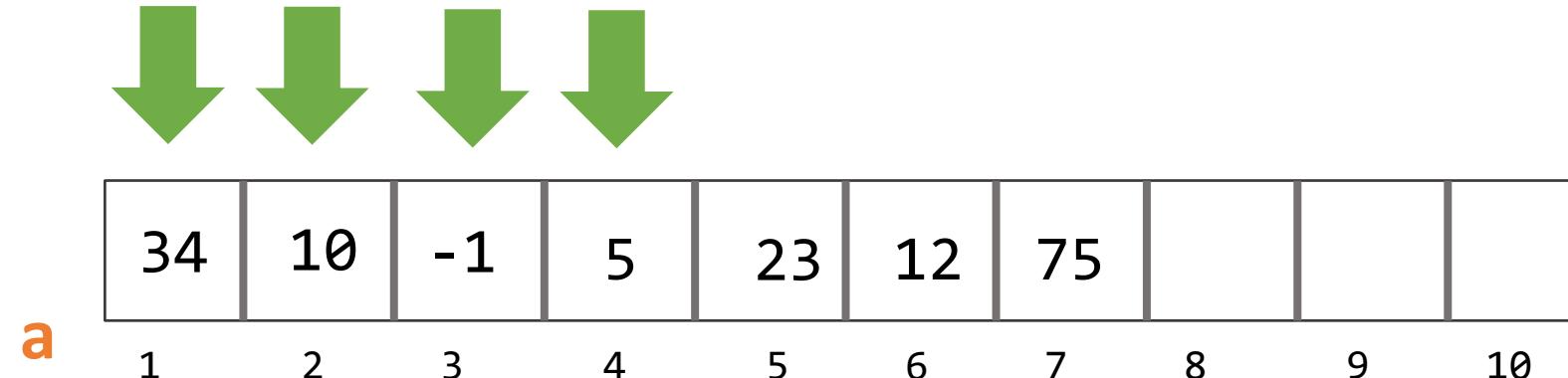
Vector Ordenado

- Se debe recorrer el vector teniendo en cuenta el orden:
 - BUSQUEDA MEJORADA
 - BUSQUEDA BINARIA

Vector Desordenado

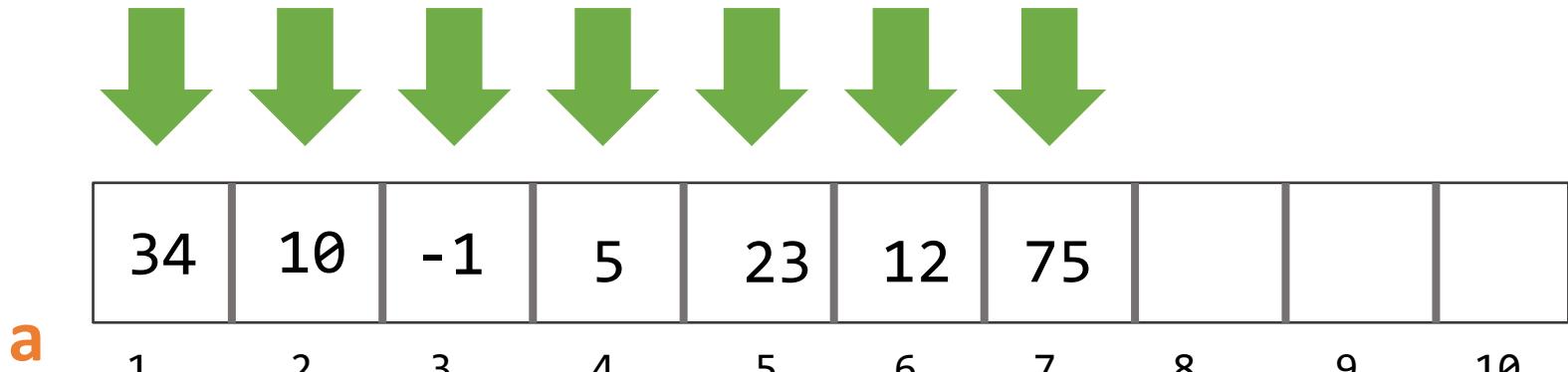
D1 = 7

5



D1 = 7

25



Qué pasos
considero?

CADP – TIPOS DE DATOS VECTORES – BUSCAR DESORDENADO



Se debe recorrer todo el vector (en el peor de los casos), y detener la búsqueda en el momento que se encuentra el dato buscado o en el que se terminó el vector.

- 1- Inicializar la búsqueda desde la posición 1 (pos).
- 2- Mientras ((el elemento buscado no se igual al valor en el arreglo[pos]) y (no se termine el arreglo))
 - 2.1 Avanzo una posición
- 3- Determino porque condición se ha terminado el while y devuelvo el resultado.

Cómo se
implementa?

CADP – TIPOS DE DATOS VECTORES – BUSCAR DESORDENADO

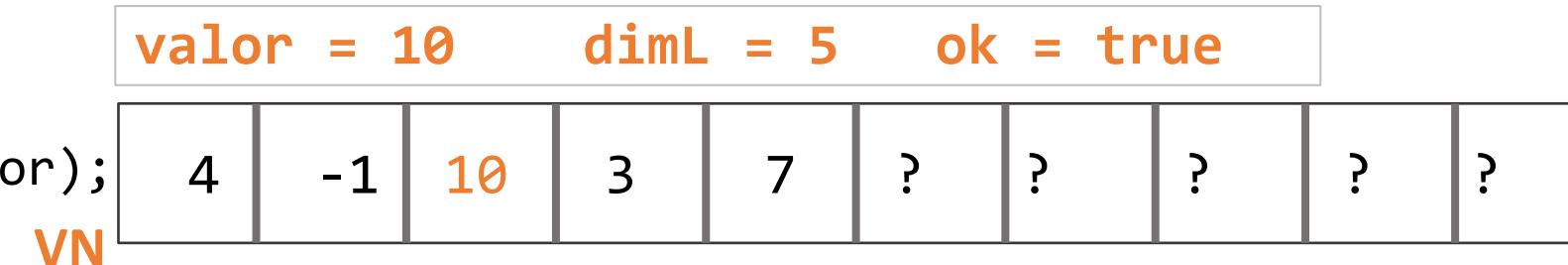
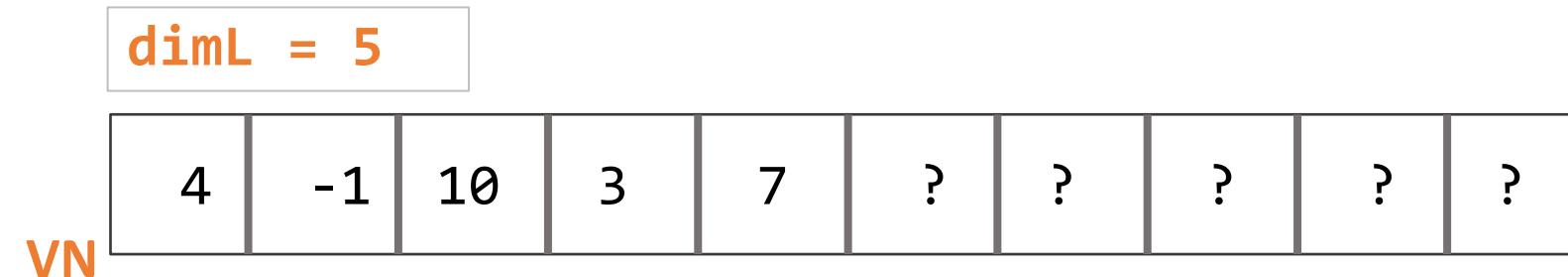
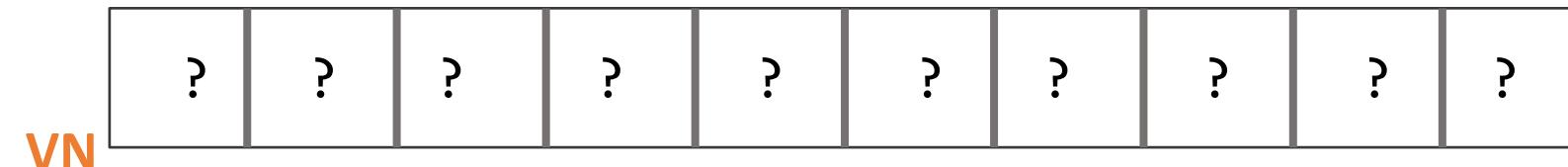


Dado un vector de números enteros (10 elementos como máximo) realice un programa que lea un nuevo número y determine si el valor se encuentra en el vector.

```
Program uno;
const
  fisica = 10;
type
  numeros= array [1..fisica] of integer;
```

```
var
  VN: numeros;
  dimL, valor:integer;
  ok:boolean;
```

```
Begin
  cargar (VN,dimL);
  read(valor);
  res:= buscar(VN,dimL,valor);
End.
```



CADP – TIPOS DE DATOS VECTORES – BUSCAR DESORDENADO



```
function buscar (a :números; dL:integer; valor:integer): boolean;  
Var  
    pos:integer;  
  
Begin  
    pos:=1;  
    while ( (pos <= dL ) and (a[pos] <> valor) ) do  
        begin  
            pos:= pos + 1;  
        end;  
    buscar:= (a[pos] = valor);  
end.
```

Es correcto?



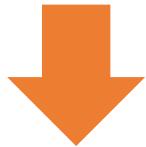
Si el elemento no está, pos en este caso quedaría en 11, y en la última línea de la función estaría asignando el resultado de comparar a[11] = valor

CADP – TIPOS DE DATOS VECTORES – BUSCAR DESORDENADO



```
function buscar (a :números; dL:integer; valor:integer): boolean;  
Var  
    pos:integer;  
  
Begin  
    pos:=1;  
    while ((a[pos] <> valor) and (pos <= dL) ) do  
        begin  
            pos:= pos + 1;  
        end;  
    buscar:=(a[pos]=valor);  
end.
```

Es correcto?



Si el elemento no está, pos en este caso quedaría en 11, y en el while se pregunta a[11] y no es válido

CADP – TIPOS DE DATOS VECTORES – BUSCAR DESORDENADO



```
function buscar (a :números; dL:integer; valor:integer): boolean;  
Var  
    pos:integer;  
  
Begin  
    pos:=1;  
    while ((pos <= dL) and (a[pos] <> valor) ) do  
        begin  
            pos:= pos + 1;  
        end;  
    buscar:= (pos <= dL);  
end.
```

Es correcto?



Si pos no es <= dL no significa que haya estado el elemento

CADP – TIPOS DE DATOS VECTORES – BUSCAR DESORDENADO

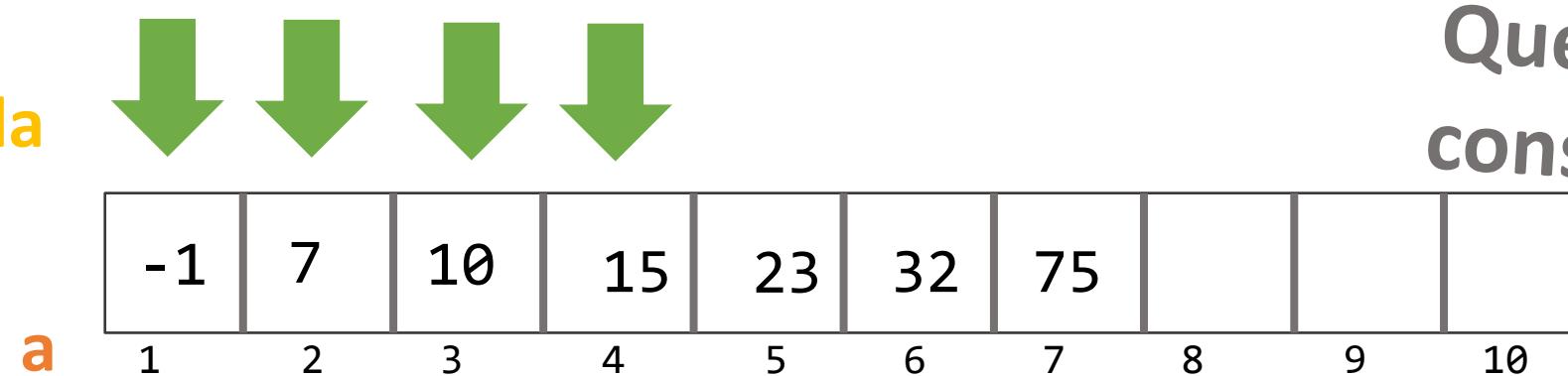


```
function buscar (a :números; dL:integer; valor:integer): boolean;  
Var  
    pos:integer;  
    esta:boolean;  
  
Begin  
    esta:= false;  
    pos:=1;  
    while ( (pos <= dL) and (not esta) ) do  
        begin  
            if (a[pos]= valor) then esta:= true  
            else  
                pos:= pos + 1;  
        end;  
    buscar:= esta;  
end.
```

Vector Ordenado
Búsqueda Mejorada

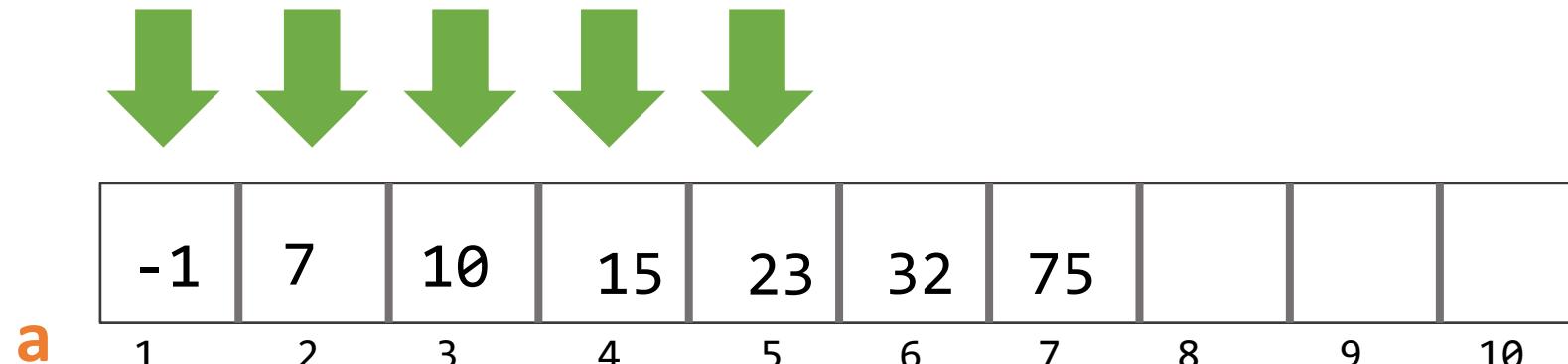
D1 = 7

15



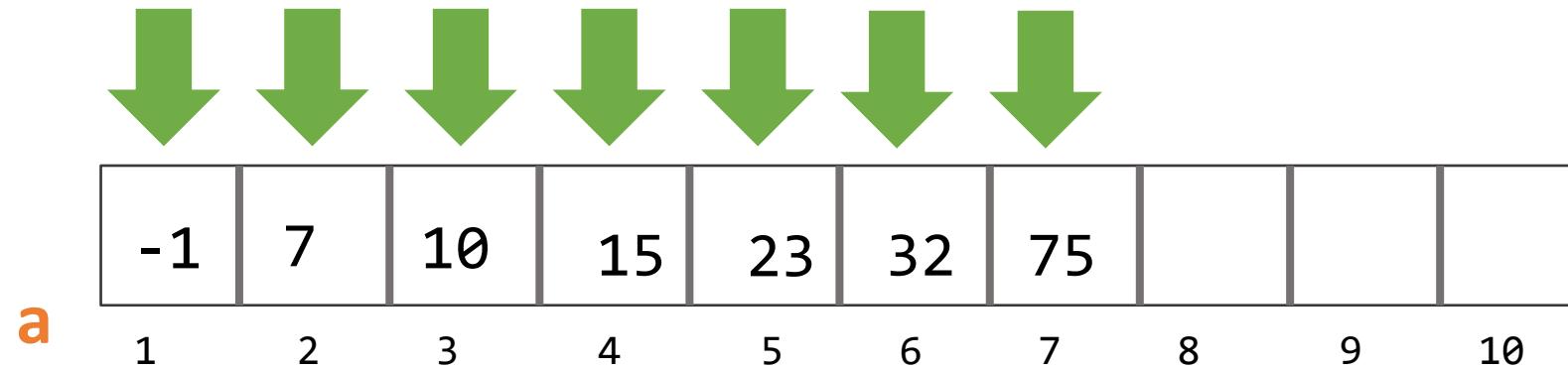
D1 = 7

16



D1 = 7

80



Qué pasos
considero?



BUSQUEDA MEJORADA

- 1- Inicializar la búsqueda desde la posición 1 (pos).
- 2- Mientras ((el elemento buscado sea menor al valor en el arreglo[pos]) y (no se termine el arreglo))
 - 2.1 Avanzo una posición
- 3- Determino porque condición se ha terminado el while y devuelvo el resultado.

**Cómo se
implementa?**

CADP – TIPOS DE DATOS



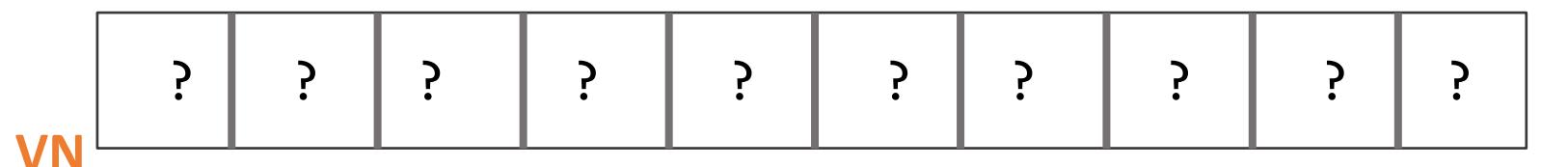
Dado un vector de números enteros (10 elementos como máximo) ordenado realice un programa que lea un número e invoque a un módulo que retorne si el número se encuentra en el vector.

```
Program uno;
  const
    fisica = 10;
  type
    numeros= array [1..fisica] of integer;
```

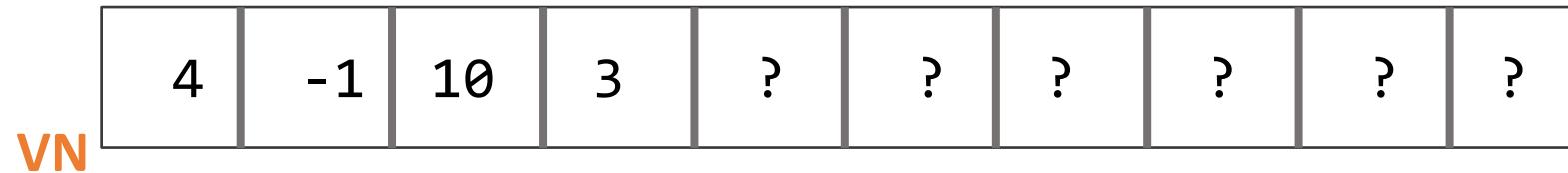
```
var
  VN: numeros;
  dimL,pos:integer;
  ok:boolean;
```

```
Begin
  cargar (VN,dimL);
  read(valor);
  ok:= existe(VN,dimL,valor);
```

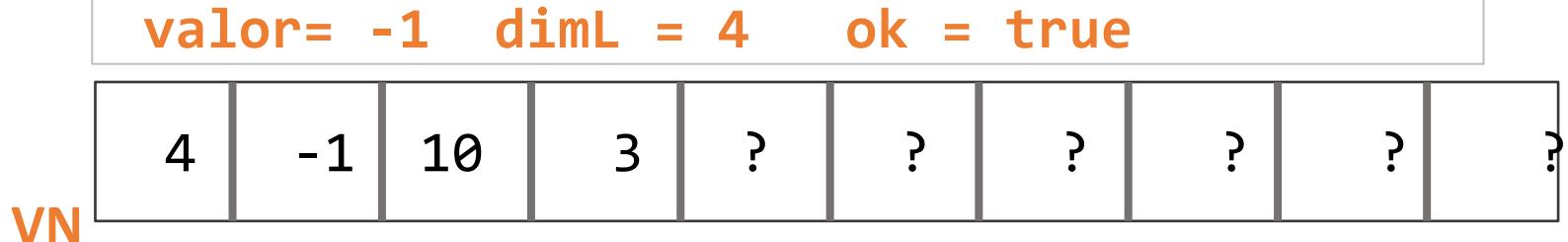
End.
Clase 7-3



```
dimL = 4
```



```
valor= -1  dimL = 4  ok = true
```



```
Function existe (a:números; dL:integer; valor:integer):boolean;
```

```
Var
```

```
    pos:integer;
```

```
Begin
```

```
    pos:=1;
```

```
    while ( (pos <= dL) and (a[pos]< valor)) do
```

```
        begin
```

```
            pos:= pos + 1;
```

```
        end;
```

```
        if ( (pos <= dL) and (a[pos]= valor)) then buscar:=true  
        else buscar:= false;
```

```
    end.
```

Importa el orden
en la condición
del while?

Alcanza con
preguntar por sólo
una de las dos
condiciones?

Vector Ordenado

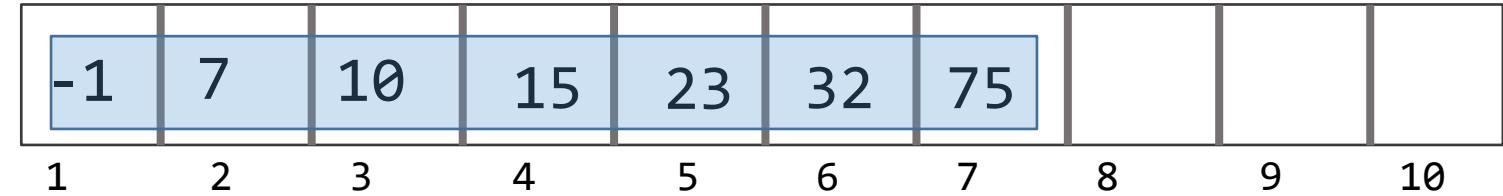
Búsqueda DICOTOMICA

D1 = 7**10**

Inf 1

Sup 7

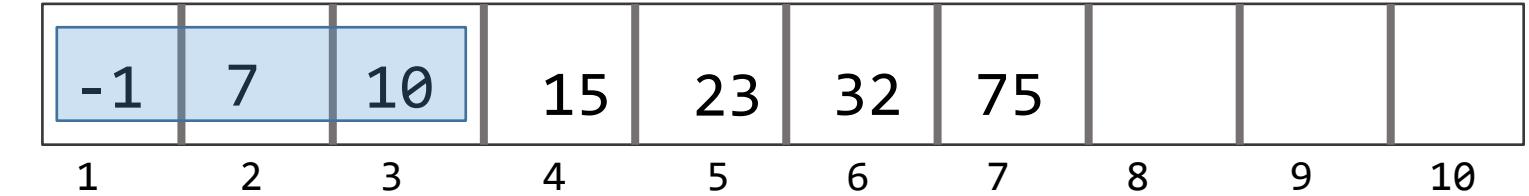
Medio 4

a**D1 = 7****10**

Inf 1

Sup 3 (medio-1)

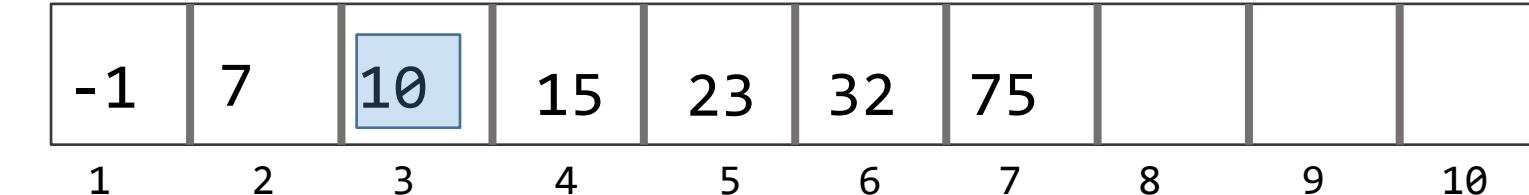
Medio 2

a**D1 = 7****80**

Inf 3 (medio+1)

Sup 3

Medio 3

aQué pasos
considero?



BUSQUEDA DICOTOMICA

Cómo se
implementa?

- 1- Se calcula la posición media del vector (teniendo en cuenta la cantidad de elementos)

- 2- Mientras ((el elemento buscado sea <> arreglo[medio]) y (inf <= sup))
 Si ((el elemento buscado sea < arreglo[medio])) entonces
 Actualizo sup
 Sino
 Actualizo inf
 Calculo nuevamente el medio

- 3- Determino porque condición se ha terminado el while y devuelvo el resultado.

CADP – TIPOS DE DATOS



Dado un vector de números enteros (10 elementos como máximo) ordenado realice un programa que lea un número e invoque a un módulo que retorne si el número se encuentra en el vector.

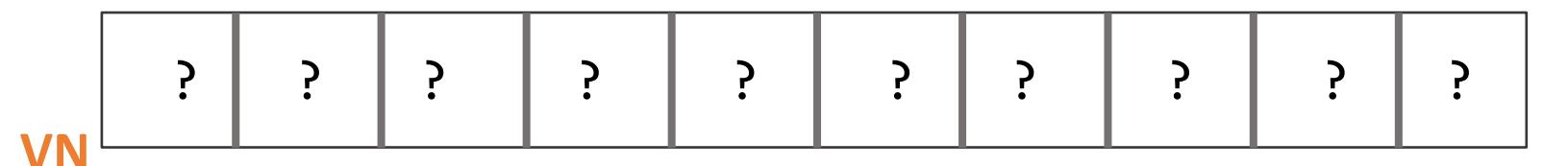
```
Program uno;
  const
    fisica = 10;
  type
    numeros= array [1..fisica] of integer;
```

```
var
  VN: numeros;
  dimL,pos:integer;
  ok:boolean;
```

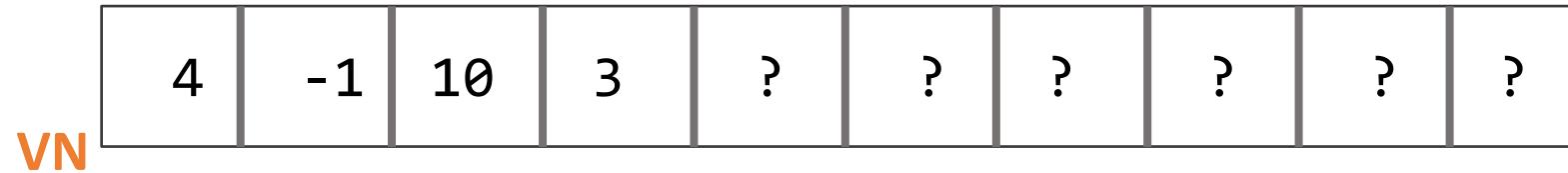
```
Begin
  cargar (VN,dimL);
  read(valor);
  ok:= dicotomica(VN,dimL,valor);
```

```
End.
```

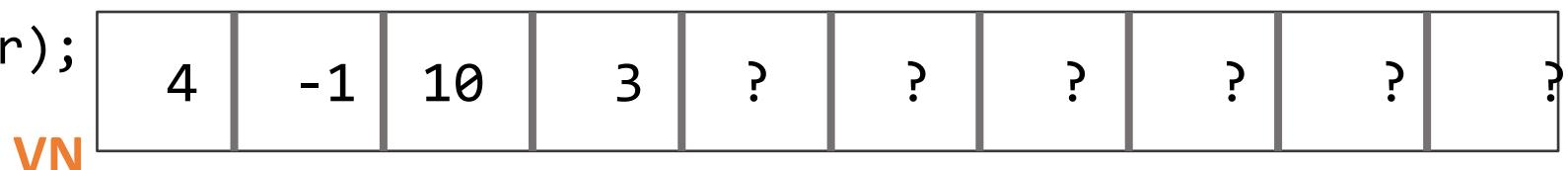
Clase 7-3



dimL = 4



valor= -1 dimL = 4 ok = true





```
Function dicotomica (a:números; dL:integer; valor:integer):boolean;
```

```
Var
```

```
    pri, ult, medio : integer;  
    ok:boolean
```

```
Begin
```

```
    ok:= false;  
    pri:= 1 ;  ult:= dL;  medio := (pri + ult ) div 2 ;
```

```
    While ( pri < = ult ) and ( valor <> vec[medio] ) do
```

```
        begin
```

```
            if ( valor < vec[medio] ) then
```

```
                ult:= medio -1 ;
```

```
            else pri:= medio+1 ;
```

```
            medio := ( pri + ult ) div 2 ;
```

```
        end;
```

```
        if (pri <=ult) and (valor = vec[medio]) then ok:=true;
```

```
    end;
```

```
    dicotomica:= ok;
```

```
end.
```



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



Alocación estática – Alocación dinámica

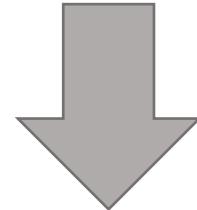
CADP – ALOCACION DE MEMORIA

MEMORIA
ESTATICA

char, boolean,
integer, real,
string, subrango,
registro, vector

Hasta ahora, cualquier variable que se declare en un programa es alojada en la memoria estática de la CPU

MEMORIA
DINAMICA



Las variables declaradas permanecen en la memoria estática durante toda la ejecución del programa, mas allá de que sigan siendo utilizadas o no.

Obviamente al permanecer en la memoria siguen ocupando memoria



CADP – ALOCACION DE MEMORIA

MEMORIA
ESTATICA

MEMORIA
DINAMICA

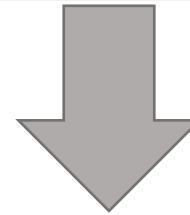
Tipo de variable	Bytes que ocupa
Char	1 byte
Boolean	1 byte
Integer	6 bytes
Real	8 bytes
String	Tamaño + 1
Subrango	Depende el tipo
Registro	La suma de sus campos
Vector	Dimensión física * tipo elemento

CADP – ALOCACION DE MEMORIA

MEMORIA
ESTATICA

MEMORIA
DINAMICA

Para solucionar los problemas mencionados anteriormente los lenguajes permiten la utilización de tipos de datos que permiten reservar y liberar memoria dinámica durante la ejecución del programa a medida que el programador lo requiera

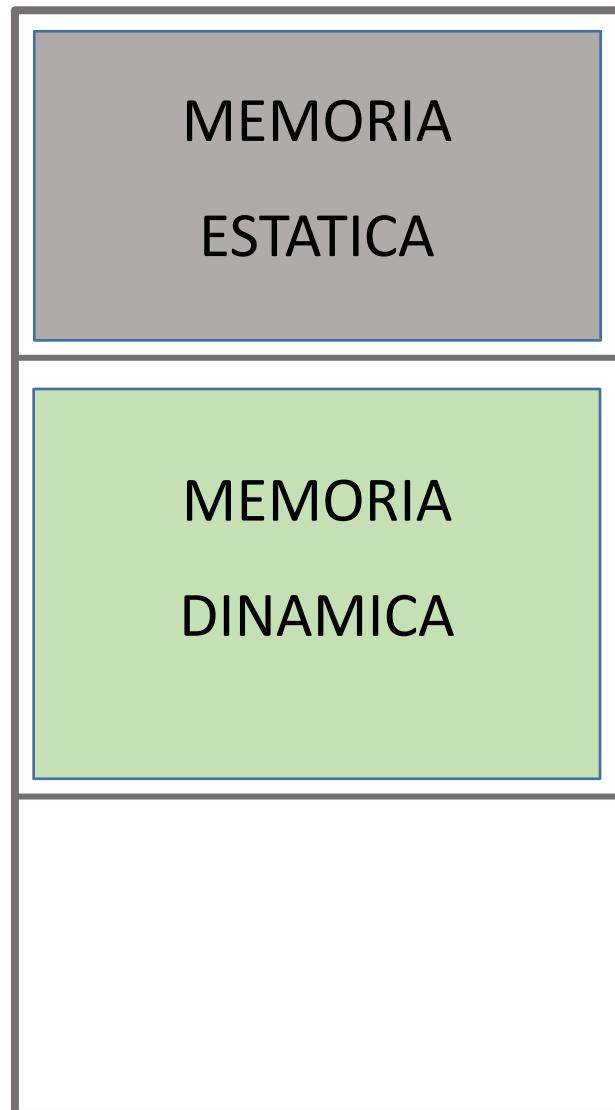


PUNTERO

Una variable puntero se aloja en la memoria estática, pero puede reservar memoria dinámica para su contenido. Siempre ocupa 4 bytes de memoria

Cuando quiere cargar contenido reserva memoria dinámica y cuando no necesita mas el contenido la libera

CADP – ALOCACION DE MEMORIA



Tipo de variable	Bytes que ocupa
Char	1 byte
Boolean	1 byte
Integer	6 bytes
Real	8 bytes
String	Tamaño + 1
Subrango	Depende el tipo
Registro	La suma de sus campos
Vector	Dimensión física * tipo elemento
Puntero	4 bytes



Cuando la variable puntero reserve memoria ahí se ocupará la memoria dinámica (la cantidad de bytes de memoria dinámica dependerá del tipo de elementos que maneje el puntero)



CADP – ALOCACION DE MEMORIA

Tipo de variable	Bytes que ocupa
Char	1 byte
Boolean	1 byte
Integer	6 bytes
Real	8 bytes
String	Tamaño + 1
Subrango	Depende el tipo
Registro	La suma de sus campos
Vector	Dimensión física * tipo elemento
Puntero	4 bytes

Type

```
vector = array[1..5] of real;
```

Var

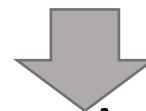
```
v:vector;
```

```
letra:char;
```

```
num:integer;
```

```
ok:boolean;
```

```
p:punteroAEntero; //ya veremos como
```



Al comenzar mi programa ocupa

```
v = 5*8 = 40 bytes
```

```
letra = 1 byte
```

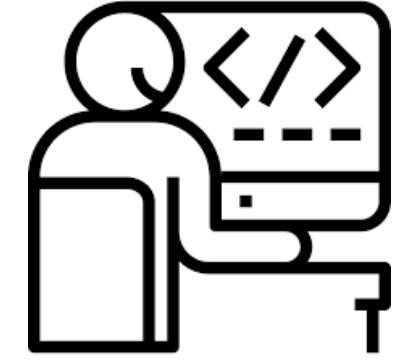
```
num = 6 bytes
```

```
ok = 1 byte
```

```
p = 4 bytes
```

52 bytes de memoria
estática

Si durante la ejecución del programa p **reserva** memoria se ocuparán tantos bytes de memoria dinámica como sea el contenido de p (en este caso 6 bytes de memoria dinámica). Luego p podrá **liberar** esa memoria dinámica durante la ejecución del programa.



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



- Alocación estática – Alocación dinámica
- Tipo de datos PUNTERO



CADP – TIPO DE DATO

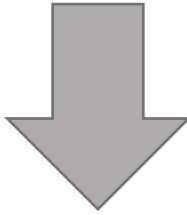
PUNTERO

MEMORIA
ESTATICA

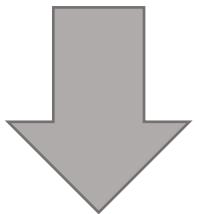
char, boolean,
integer, real,
string, subrango,
registro, vector
PUNTERO

MEMORIA
DINAMICA

Hasta ahora, cualquier variable que se declare en un programa es alojada en la memoria estática de la CPU



Una variable de tipo puntero contiene como dato una dirección de memoria dinámica.



En esa dirección de memoria se encuentra el dato que realmente se quiere guardar.



CADP – TIPO DE DATO

PUNTERO

MEMORIA

ESTATICA

MEMORIA

DINAMICA

Tipo de variable	Bytes que ocupa
Char	1 byte
Boolean	1 byte
Integer	6 bytes
Real	8 bytes
String	Tamaño + 1
Subrango	Depende el tipo
Registro	La suma de sus campos
Vector	Dimensión física * tipo elemento
Puntero	4 bytes



CADP – TIPO DE DATO

PUNTERO

SIMPLE: aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.

COMPUESTO: pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.

TIPO DE DATO

SIMPLE

COMPUESTO

DEFINIDO POR EL LENGUAJE

Integer

Real

Char

Boolean

Puntero

DEFINIDO POR EL PROGRAMADOR

Subrango

DEFINIDO POR EL LENGUAJE

String

DEFINIDO POR EL PROGRAMADOR

Registros

Arreglos



Es un tipo de variable usada para almacenar una dirección en memoria dinámica. En esa dirección de memoria se encuentra el valor real que almacena. El valor puede ser de cualquiera de los tipos vistos (char, boolean, integer, real, string, registro, arreglo u otro puntero).

Un puntero es un tipo de datos simple.

Cómo se ve
gráficamente?

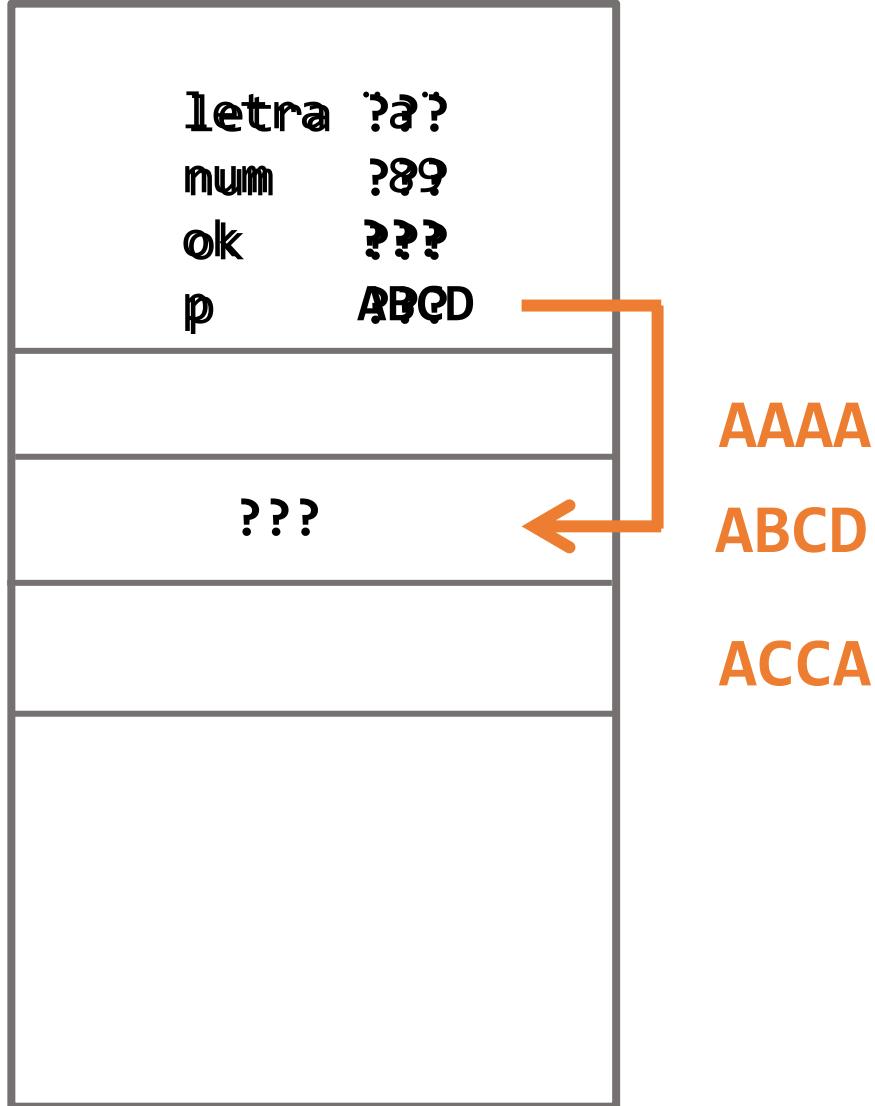


CADP – TIPO DE DATO

PUNTERO

MEMORIA
ESTATICA

MEMORIA
DINAMICA



Type

puntero //ya veremos como

Var

```
letra:char;  
num:integer;  
ok:boolean;  
p:puntero;
```

Cómo se declaran?

Begin

```
letra:= "a";  
num:= 89;  
p: pide memoria //ya veremos como
```

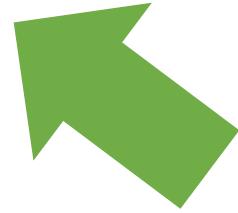
...

End.



Type

```
puntero = ^ tipo de datos;
```



Var

```
p:puntero;
```

Puede ser cualquiera de los tipos vistos previamente: integer, boolean, char, real, subrango, registro, vector.

Ejemplos



CADP – TIPO DE DATO PUNTERO

Type

```
  TipoCadena = array [1..10] of char;
```

```
  PunCadena = ^TipoCadena;
```

```
  PunReal = ^real;
```

```
  PunString = ^string;
```

```
Datos = record
    nombre: string[10];
    apellido: string[10];
    altura: real;
  end;
```

```
PunDatos = ^datos;
```

var

```
  pReal: PunReal;
  t: PunString;
  r: PunString;
  puntero: PunCadena;
  p,q: PunDatos;
  d:datos;
```

begin

....

end.



Cómo vamos a trabajar?

Una variable de tipo puntero ocupa una cantidad de memoria fija, independiente del tipo de dato al que apunta (4 bytes). Es un tipo de datos simple.

Una variable de tipo puntero puede reservar y liberar memoria durante la ejecución de un programa para almacenar su contenido

Un dato referenciado o apuntado, como los ejemplos vistos, no tienen memoria asignada, o lo que es lo mismo no existe inicialmente espacio reservado en memoria para este dato.



Creación de una variable puntero.

Destrucción de una variable puntero.

Asignación entre variables puntero.

Asignación de un valor al contenido de una variable puntero.

Comparación de una variable puntero



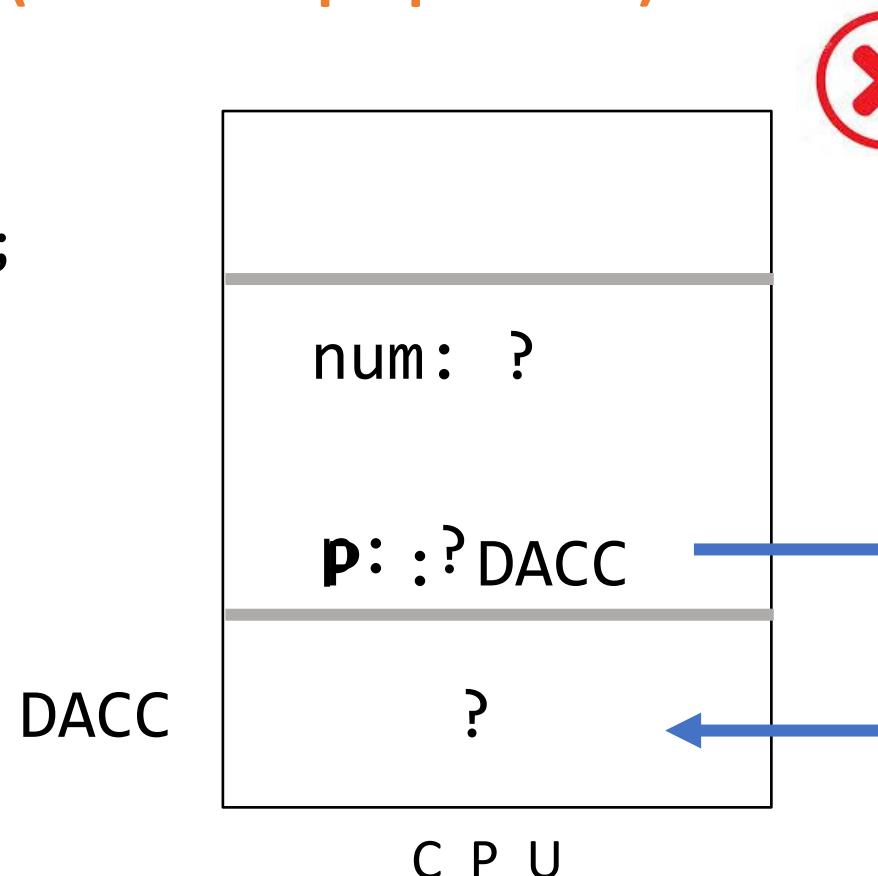


CREACION

Implica reservar una dirección memoria dinámica libre para poder asignarle contenidos a la dirección que contiene la variable de tipo puntero. **new(variable tipo puntero)**

```
Program uno;
Type
  puntero = ^integer;
Var
  num:integer;
  p:puntero;

Begin
  new (p);
  ...
End.
```



No se puede asignar a
un puntero una
dirección específica
(p:= ABCD)



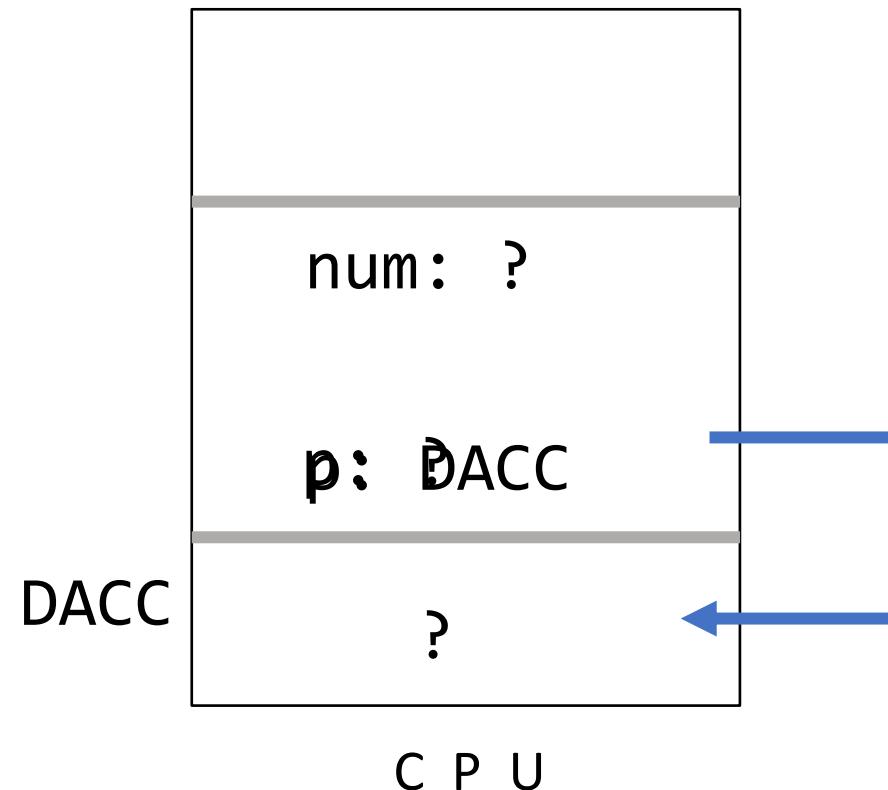
ELIMINACION

Implica liberar la memoria dinámica que contenía la variable de tipo puntero. **dispose(variable tipo puntero)**

```
Program uno;
Type
    puntero = ^integer;
```

```
Var
    num:integer;
    p:puntero;
```

```
Begin
    new (p);
    dispose (p);
End.
```





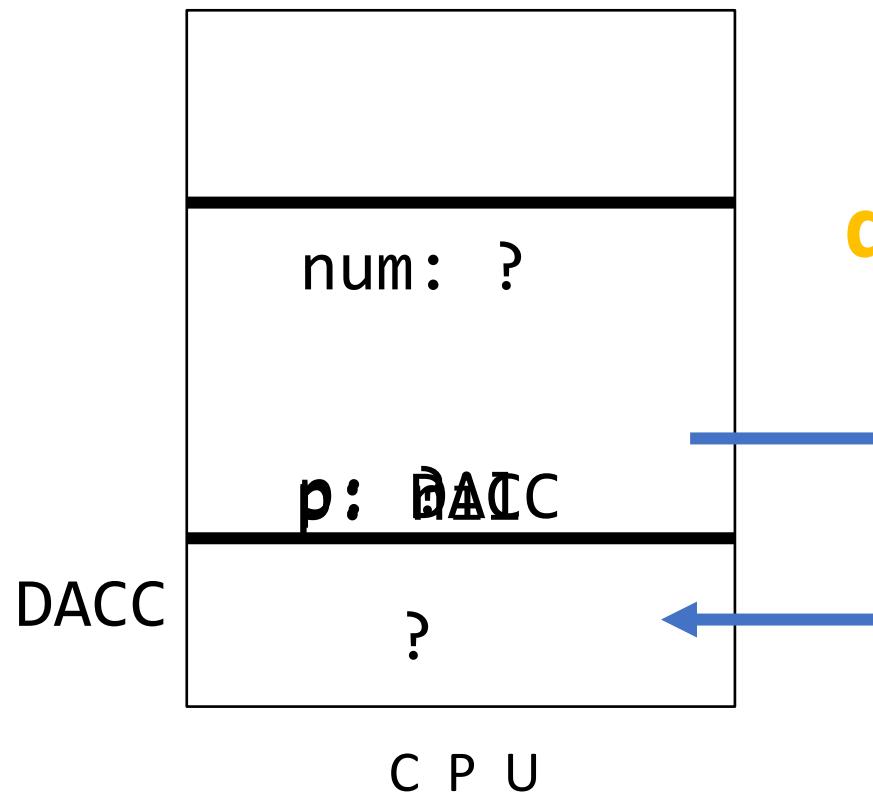
LIBERACION

Implica cortar el enlace que existe con la memoria dinámica. La misma queda ocupada pero ya no se puede acceder. **nil**

```
Program uno;
Type
    puntero = ^integer;
```

```
Var
    num:integer;
    p:puntero;
```

```
Begin
    new (p);
    p:= nil;
End.
```



Cuál es la
diferencia?



CADP – TIPO DE DATO

PUNTERO

DISPOSE (p)

Libera la conexión que existe entre la variable y la posición de memoria.

Libera la posición de memoria.

La memoria liberada puede utilizarse en otro momento del programa.



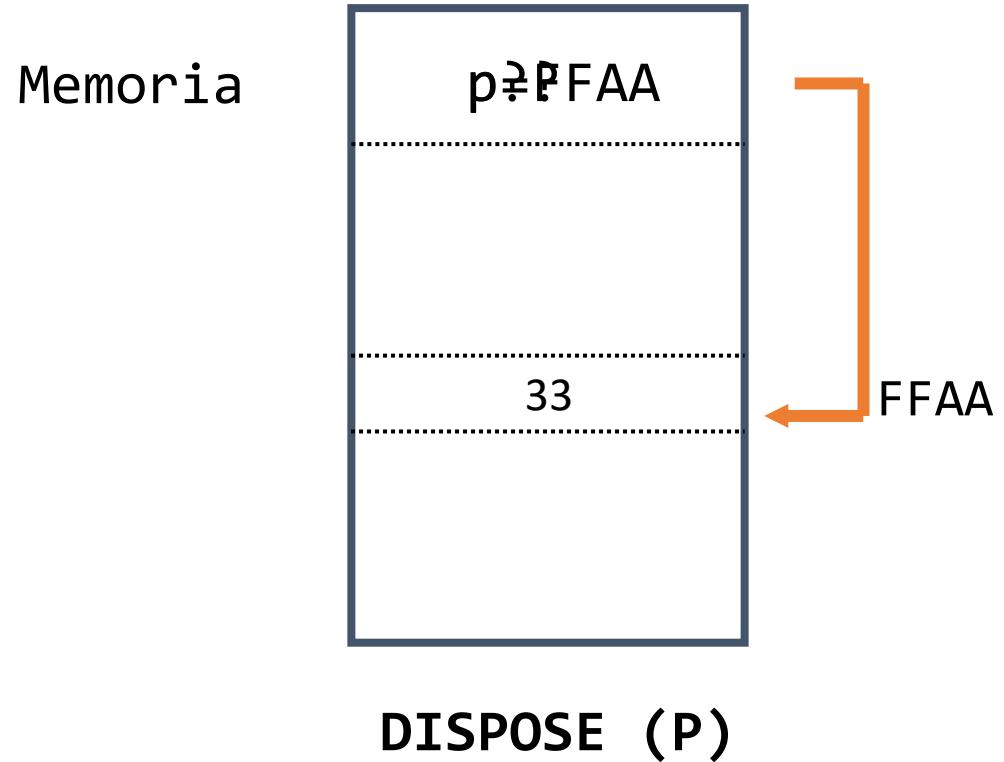
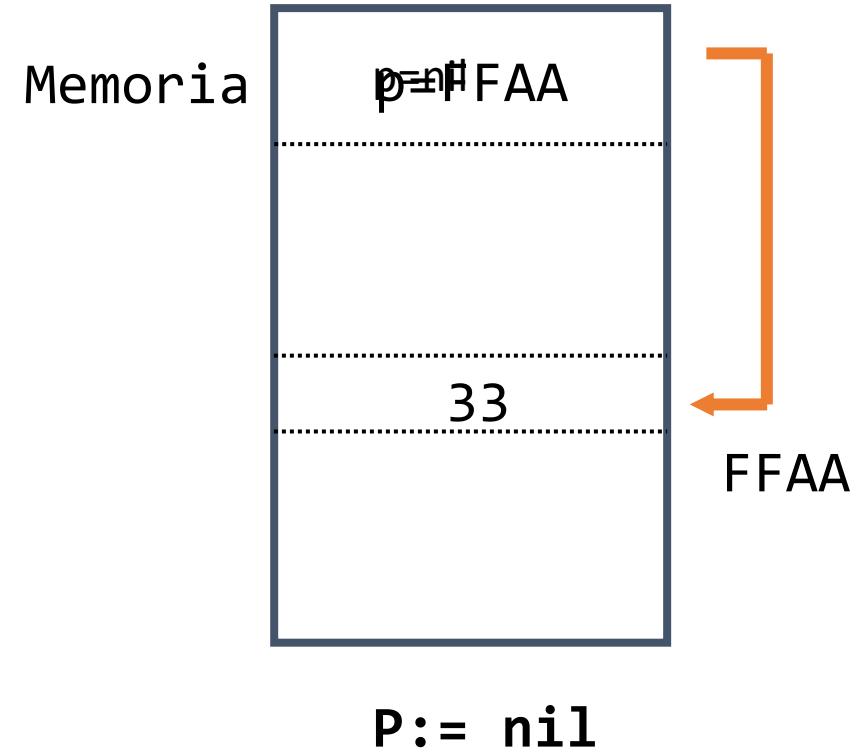
p:=nil

Libera la conexión que existe entre la variable y la posición de memoria.

La memoria sigue ocupada.

La memoria no se puede referenciar ni utilizar.

Gráficamente ...?

**DISPOSE (p)****p:=nil**



ASIGNACION entre punteros

Implica asignar la dirección de un puntero a otra variable puntero del mismo tipo. :=

```
Program uno;
Type
  puntero = ^integer;
```

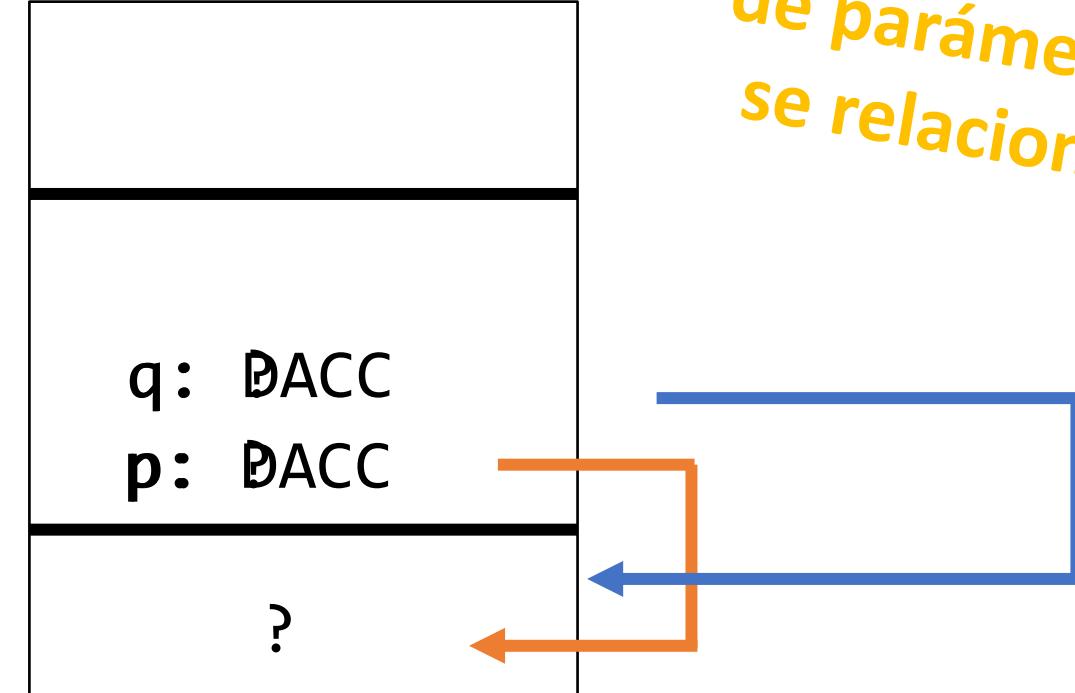
```
Var
  q:puntero;
  p:puntero;
```

```
Begin
  new (p);
  q:=p;
```

```
End.
```

DACC

C P U





CADP – TIPO DE DATO

PUNTERO

Program uno;

Type

```
puntero = ^integer;
```

Var

```
q:puntero;  
p:puntero;
```

Begin

```
new (p);  
q:=p;  
dispose (p);
```

End.

Cómo queda la memoria en cada programa?

Program dos;

Type

```
puntero = ^integer;
```

Var

```
q:puntero;  
p:puntero;
```

Begin

```
new (p);  
q:=p;  
p:= nil;
```

End.



CADP – TIPO DE DATO

PUNTERO

Var

 p,q:pun;

Begin

 new (p);

 q:=p;

 dispose(p);

End.

P = ???

q = ???

P = CDAA

q = ???

P = CDAA
q = ???

???

CDAA

???

CDAA

P = ???
q = ???



CADP – TIPO DE DATO

PUNTERO

Var

 p,q:pun;

Begin

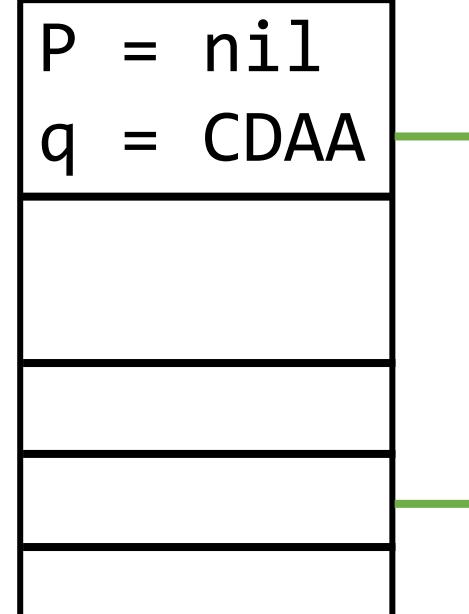
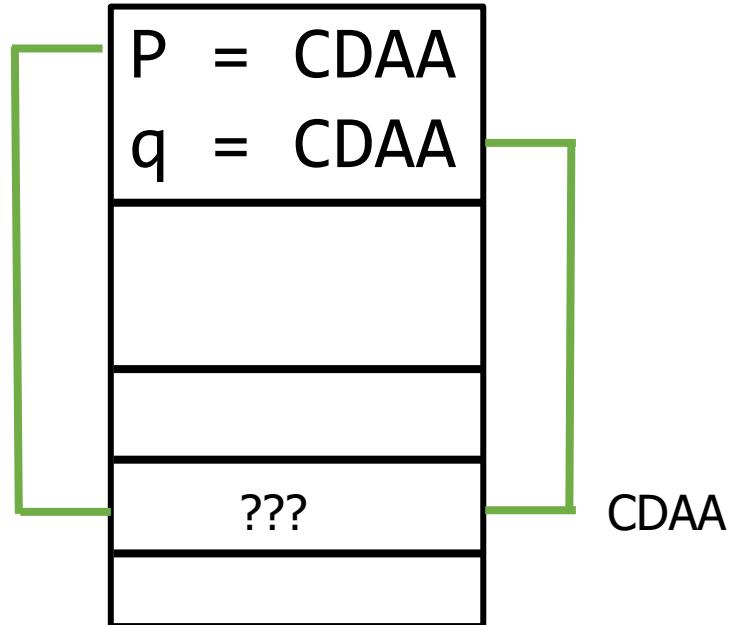
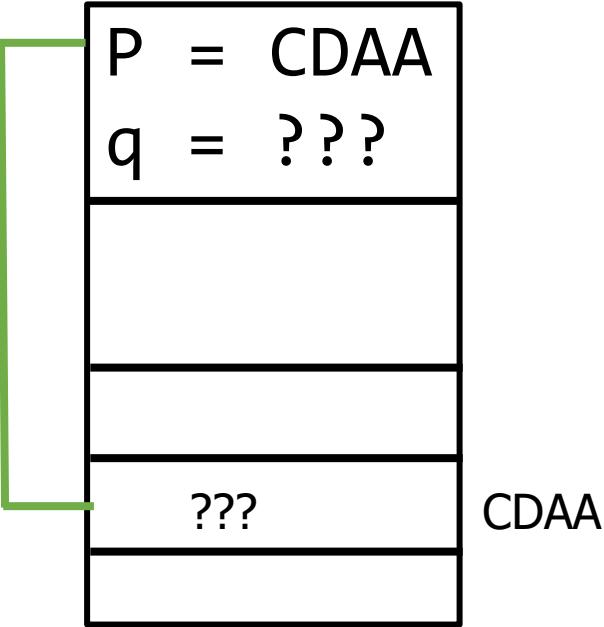
 new (p);

 q:=p;

 p:= nil;

End.

P = ???
q = ???





CONTENIDO de un puntero

Implica poder acceder al contenido que contiene la dirección de memoria que tiene una variable de tipo puntero. ^

```
Program uno;  
Type  
    puntero = ^integer;
```

```
Var  
    p:puntero;
```

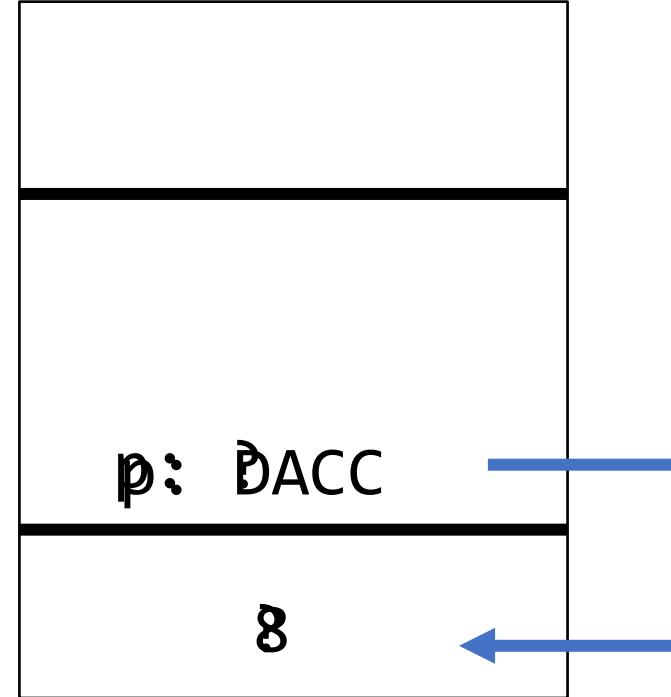
```
Begin  
    new (p);  
    p^:=8;
```

```
End.
```

DACC

8

C P U



Qué operaciones podré hacer?

Ejemplos ...



CADP – TIPO DE DATO

PUNTERO

EJEMPLOS – Cómo varía la memoria?

Qué imprime cada programa?

```
Program uno;
Type
  punt = ^integer;
Var
  p,q:punt;
  num:integer;

Begin
  num:= 63;
  new (p);
  new(q);
  q^:= num - 10;

  write(q^);
  write(p^);
end.
```

```
Program dos;
Type
  punt = ^integer;
Var
  p,q:punt;

Begin
  new (p);
  p^:= 14;
  write (p^);
  q:=p;
  q^:= q^*10;
  write (p^);
  write(q^);
  dispose (q);
  write (p^);
  write (q^);
end.
```

```
Program tres;
Type
  punt= ^integer;
Var
  p,q:punt;

Begin
  new (p);
  new(q);
  p:= q;
  q^:=10;

  write(q^);
  write(p^);
end.
```

```
Program cuatro;
Type
  punt = ^integer;
Var
  p,q:punt;

Begin
  new (p);
  p^:= 14;
  write (p^);
  q:=p;
  q^:= q^*10;
  write (p^);
  write(q^);
  q=nil;
  write (p^);
  write(q^);
End.
```



- if ($p = \text{nil}$) then, compara si el puntero p no tiene dirección asignada.
- if ($p = q$) then, compara si los punteros p y q apuntan a la misma dirección de memoria.
- if ($p^\wedge = q^\wedge$) then, compara si los punteros p y q tienen el mismo contenido.
- no se puede hacer read (p), ni write (p), siendo p una variable puntero.
- no se puede asignar una dirección de manera directa a un puntero, $p := ABCD$
- no se pueden comparar las direcciones de dos punteros ($p < q$).



Conceptos de Algoritmos Datos y Programas

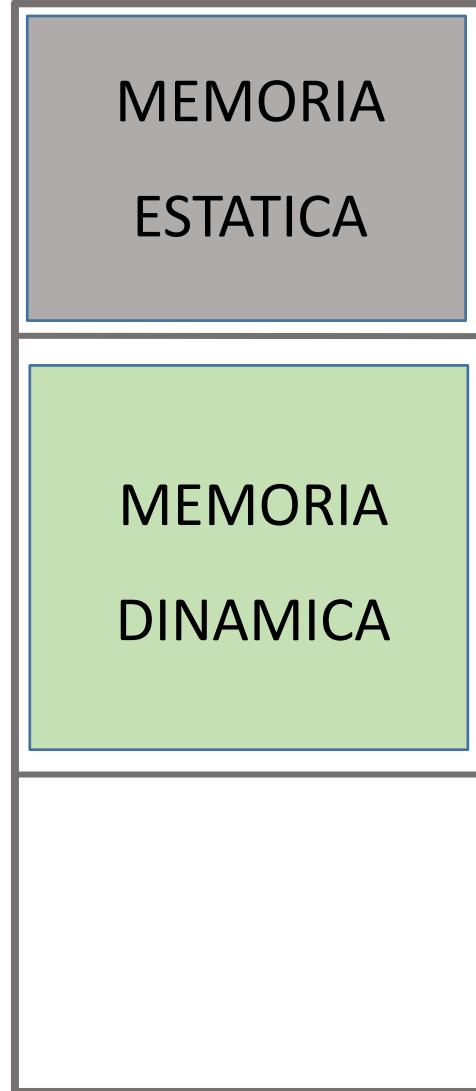
CADP – TEMAS



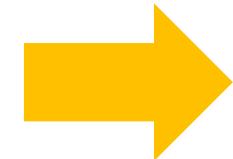
- Alocación de memoria
- Cálculo en la utilización de la memoria



CADP – MEMORIA DE UN PROGRAMA



En rasgos generales la memoria necesaria para la ejecución de un programa puede dividirse en dos.



MEMORIA ESTATICA: a modo de simplicidad consideraremos sólo las variables locales, variables globales de programa y constantes.



MEMORIA DINAMICA: a modo de simplicidad consideraremos sólo cuando en la ejecución de un programa se reserva o libera memoria.

Cómo
trabajamos?



CADP – MEMORIA DE UN PROGRAMA

```
Program uno;
Const
  DF = 10;
Type
  puntero = ^real;
  puntero2 = ^char;
  persona = record
    nombre:string[20];
    dni:integer;
  end;
  puntPer = ^persona;

Var
  p:puntero; q:puntero2;
  per: puntPer;
  perso:persona;

Begin
End.
```

Tabla de ocupación:

char,	(1 byte)
boolean,	(1 byte)
integer	(4 bytes)
real,	(8 bytes)
string,	(tamaño + 1 byte)
subrango,	(depende el tipo)
registro,	(suma de sus campos)
arreglos	(dimFísica*tipo elemento)
puntero	(4 bytes)

CALCULO DE MEMORIA ESTATICA

DF = 4 bytes	
p = 4 bytes	
q = 4 bytes	41 bytes
per = 4 bytes	
perso = 21 + 4 = 25 bytes	

CALCULO DE MEMORIA DINAMICA

Como no hay operaciones de new()
ni dispose(), no se modifica la
memoria dinámica **0 bytes**



CADP – MEMORIA DE UN PROGRAMA

```
Program uno;
Const
  DF = 10;
Type
  puntero = ^real;
  puntero2 = ^char;
  persona = record
    nombre:string[20];
    dni:integer;
  end;
  puntPer = ^persona;

Var
  p:puntero; q:puntero2;
  per: puntPer;
  perso:persona;
Begin
  new(p);
End.
```

Tabla de ocupación:

char,	(1 byte)
boolean,	(1 byte)
integer	(4 bytes)
real,	(8 bytes)
string,	(tamaño + 1 byte)
subrango,	(depende el tipo)
registro,	(suma de sus campos)
arreglos	(dimFísica*tipo elemento)
puntero	(4 bytes)

CALCULO DE MEMORIA ESTATICA

DF = 4 bytes	
p = 4 bytes	
q = 4 bytes	41 bytes
per = 4 bytes	
perso = 21 + 4 = 25 bytes	

CALCULO DE MEMORIA DINAMICA

Al hacer un new se reserva memoria para el contenido de p
8 bytes



CADP – MEMORIA DE UN PROGRAMA

```
Program uno;
Const
  DF = 10;
Type
  puntero = ^real;
  puntero2 = ^char;
  persona = record
    nombre:string[20];
    dni:integer;
  end;
  puntPer = ^persona;

Var
  p:puntero; q:puntero2;
  per: puntPer;
  perso:persona;
Begin
  new(p);
  new(per);
End.
```

Tabla de ocupación:

char,	(1 byte)
boolean,	(1 byte)
integer	(4 bytes)
real,	(8 bytes)
string,	(tamaño + 1 byte)
subrango,	(depende el tipo)
registro,	(suma de sus campos)
arreglos	(dimFísica*tipo elemento)
puntero	(4 bytes)

CALCULO DE MEMORIA ESTATICA

DF = 4 bytes	
p = 4 bytes	
q = 4 bytes	41 bytes
per = 4 bytes	
perso = 21 + 4 = 25 bytes	

CALCULO DE MEMORIA DINAMICA

Al hacer un new se reserva memoria para el contenido de p y per (8 + 25) 33 bytes



CADP – MEMORIA DE UN PROGRAMA

```
Program uno;
Const
  DF = 10;
Type
  puntero = ^real;
  puntero2 = ^char;
  persona = record
    nombre:string[20];
    dni:integer;
  end;
  puntPer = ^persona;
Var
  p:puntero; q:puntero2;
  per: puntPer;
  perso:persona;
Begin
  new(p);
  new(per);
  read (per^.nombre);
End.
```

Tabla de ocupación:

char,	(1 byte)
boolean,	(1 byte)
integer	(4 bytes)
real,	(8 bytes)
string,	(tamaño + 1 byte)
subrango,	(depende el tipo)
registro,	(suma de sus campos)
arreglos	(dimFísica*tipo elemento)
puntero	(4 bytes)

CALCULO DE MEMORIA ESTATICA

DF = 4 bytes
p = 4 bytes
q = 4 bytes
per = 4 bytes
perso = 21 + 4 = 25 bytes

41 bytes

CALCULO DE MEMORIA DINAMICA

Al hacer un new se reserva memoria para el contenido de p y per (8 + 25), el read NO altera la memoria

33 bytes



CADP – MEMORIA DE UN PROGRAMA

```
Program uno;
Const
  DF = 10;
Type
  puntero = ^real;
  puntero2 = ^char;
  persona = record
    nombre:string[20];
    dni:integer;
  end;
  puntPer = ^persona;

Var
  p,p1:puntero; q:puntero2;
  per: puntPer;
  perso:persona;
Begin
  new(p);
  p1:= p;
End.
```

Tabla de ocupación:

char,	(1 byte)
boolean,	(1 byte)
integer	(4 bytes)
real,	(8 bytes)
string,	(tamaño + 1 byte)
subrango,	(depende el tipo)
registro,	(suma de sus campos)
arreglos	(dimFísica*tipo elemento)
puntero	(4 bytes)

CALCULO DE MEMORIA ESTATICA

DF = 4 bytes	
p,p1 = 8 bytes	
q = 4 bytes	45 bytes
per = 4 bytes	
perso = 21 + 4 = 25 bytes	

CALCULO DE MEMORIA DINAMICA

Al hacer un new se reserva memoria para el contenido de p, la asignación de direcciones NO altera la memoria **8 bytes**

CADP – MEMORIA DE UN PROGRAMA



```
Program uno;
Const
  DF = 10;
Type
  puntero = ^real;
  puntero2 = ^char;
  persona = record
    nombre:string[20];
    dni:integer;
  end;
  puntPer = ^persona;

Var
  p:puntero; q:puntero2;
  per: puntPer;
  perso:persona;
Begin
  new(p);
  dispose(p);
End.
```

Tabla de ocupación:

char, (1 byte)
boolean, (1 byte)
integer (4 bytes)
real, (8 bytes)
string, (tamaño + 1 byte)
subrango, (depende el tipo)
registro, (suma de sus campos)
arreglos (dimFísica*tipo elemento)
puntero (4 bytes)

CALCULO DE MEMORIA ESTATICA

DF = 4 bytes
p = 4 bytes
q = 4 bytes
per = 4 bytes
perso = 21 + 4 = 25 bytes

41 bytes

CALCULO DE MEMORIA DINAMICA

Al hacer un new se reserva memoria para el contenido de p, luego al hacer dispose(p) se libera la memoria

0 bytes



CADP – MEMORIA DE UN PROGRAMA

```
Program uno;
Const
  DF = 10;
Type
  puntero = ^real;
  puntero2 = ^char;
  persona = record
    nombre:string[20];
    dni:integer;
  end;
  puntPer = ^persona;

Var
  p:puntero; q:puntero2;
  per: puntPer;
  perso:persona;
Begin
  new(p);
  p:= nil;
End.
```

Tabla de ocupación:

char,	(1 byte)
boolean,	(1 byte)
integer	(4 bytes)
real,	(8 bytes)
string,	(tamaño + 1 byte)
subrango,	(depende el tipo)
registro,	(suma de sus campos)
arreglos	(dimFísica*tipo elemento)
puntero	(4 bytes)

CALCULO DE MEMORIA ESTATICA

DF = 4 bytes	
p = 4 bytes	
q = 4 bytes	41 bytes
per = 4 bytes	
perso = 21 + 4 = 25 bytes	

CALCULO DE MEMORIA DINAMICA

Al hacer un new se reserva memoria para el contenido de p, luego al hacer nil NO se libera la memoria

8 bytes



CADP – MEMORIA DE UN PROGRAMA

```
Program uno;  
Type  
    puntero = ^real;  
    puntero2 = ^char;  
    persona = record  
        nombre:string[20];  
        dni:integer;  
    end;  
    punPer = ^persona;
```

```
Var  
p,p1:puntero;  
per: punPer;
```

```
Begin  
    new(per);  
    new(p);  
    p^:= 8;  
    p1:= p;  
    dispose(p1);
```

End.

Tabla de ocupación:

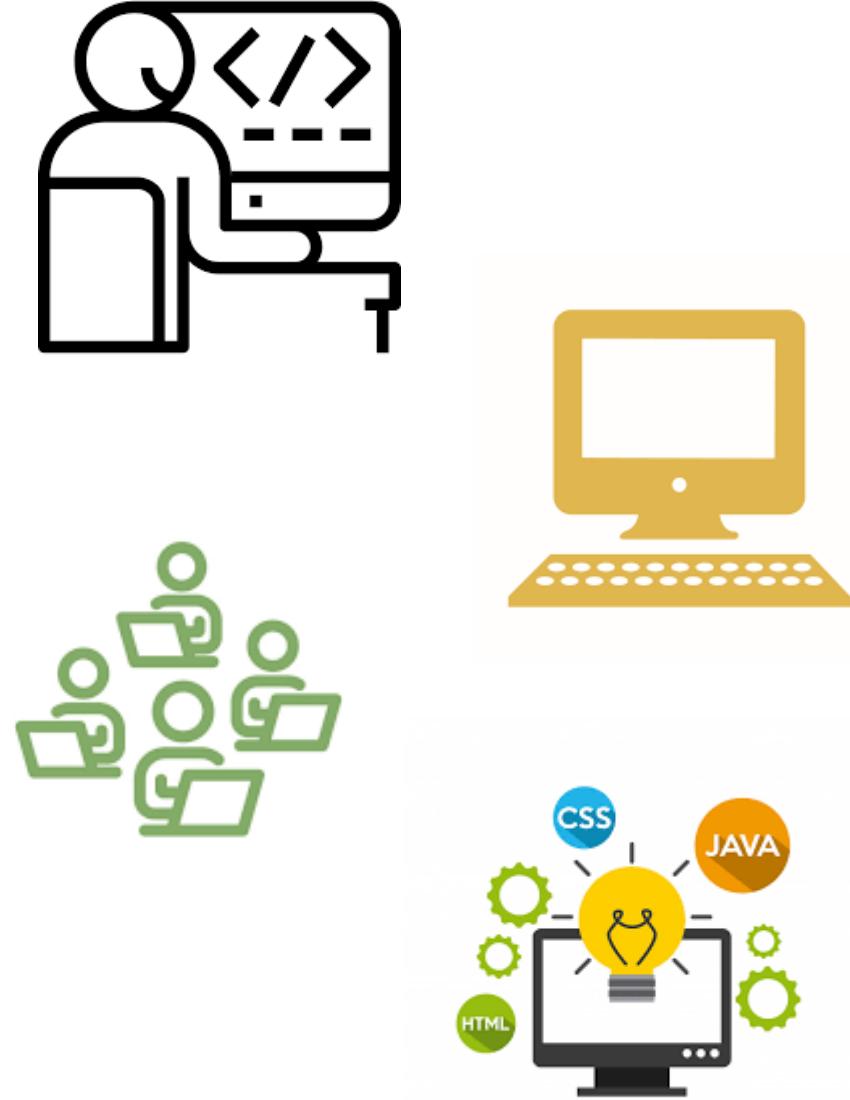
char,	(1 byte)
boolean,	(1 byte)
integer	(4 bytes)
real,	(8 bytes)
string,	(tamaño + 1 byte)
subrango,	(depende el tipo)
registro,	(suma de sus campos)
arreglos	(dimFísica*tipo elemento)
puntero	(4 bytes)

CALCULO DE MEMORIA ESTATICA

p = 4 bytes	
p1 = 4 bytes	12 bytes
per = 4 bytes	

CALCULO DE MEMORIA DINAMICA

new(per) = 25 bytes	
new(p) = 8 bytes	25 bytes
dispose(p1) = -8 bytes	



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



- Estructura de Datos - LISTA
- Características de una LISTA
- Operaciones de una LISTA

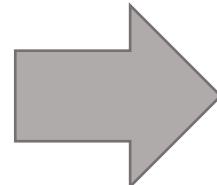


CADP – TIPOS DE DATOS - LISTA

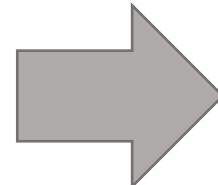


Realizar un programa que lea números que representan edades de personas hasta leer la edad -1. Finalizada la lectura se quiere informar cual fue la edad máxima leída.

10
4
57
62
39
-1

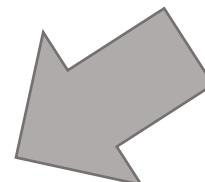


Dónde
almaceno las
edades?



Necesito una estructura que pueda ir agregando datos y por lo tanto su tamaño pueda ir variando en la ejecución del programa (estructura dinámica)

LISTA





CADP – TIPOS DE DATOS - LISTA

SIMPLE: aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.

TIPO DE DATO

COMPUESTO

SIMPLE

DEFINIDO POR EL LENGUAJE

DEFINIDO POR EL PROGRAMADOR

Integer
Real
Char
Boolean
Puntero

Subrango

DEFINIDO POR EL LENGUAJE

String

DEFINIDO POR EL PROGRAMADOR

Registros
Arreglos
Lista

COMPUESTO: pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.

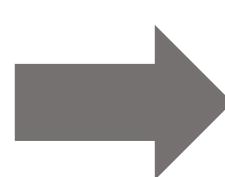


CADP – TIPOS DE DATOS - LISTA



Es una colección de nodos. Cada nodo contiene un elemento (valor que se quiere almacenar en la lista) y una dirección de memoria dinámica que indica donde se encuentra el siguiente nodo de la lista.

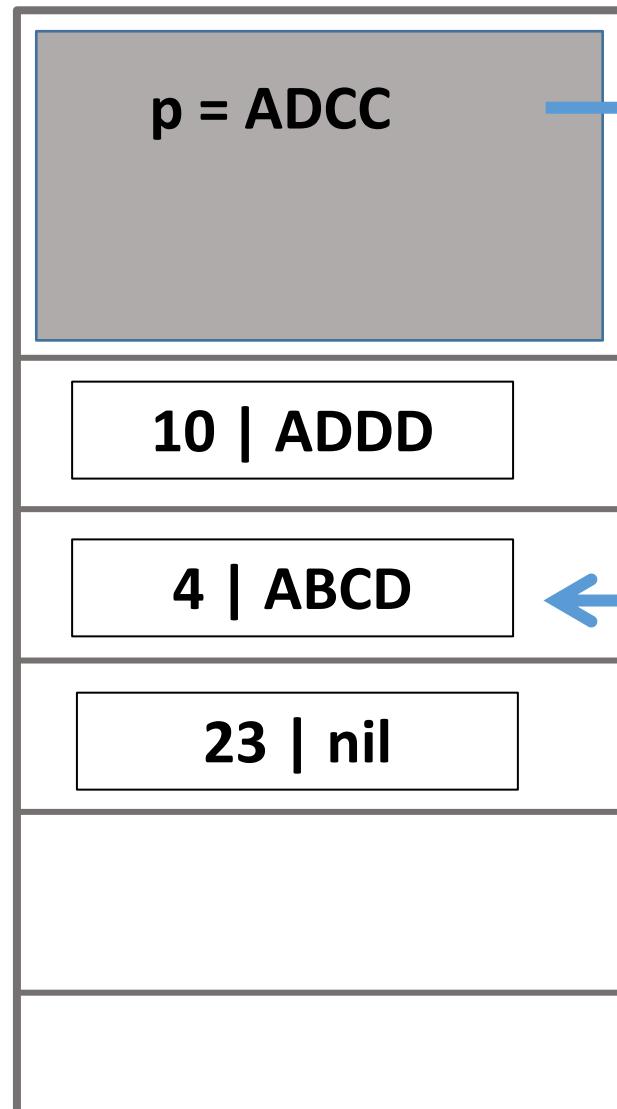
Toda lista tiene un nodo inicial.



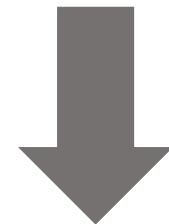
Los **nodos** que la componen pueden no ocupar posiciones contiguas de memoria. Es decir pueden aparecer dispersos en la memoria, pero mantienen un orden lógico interno.

Gráficamente ...

CADP – TIPOS DE DATOS - LISTA

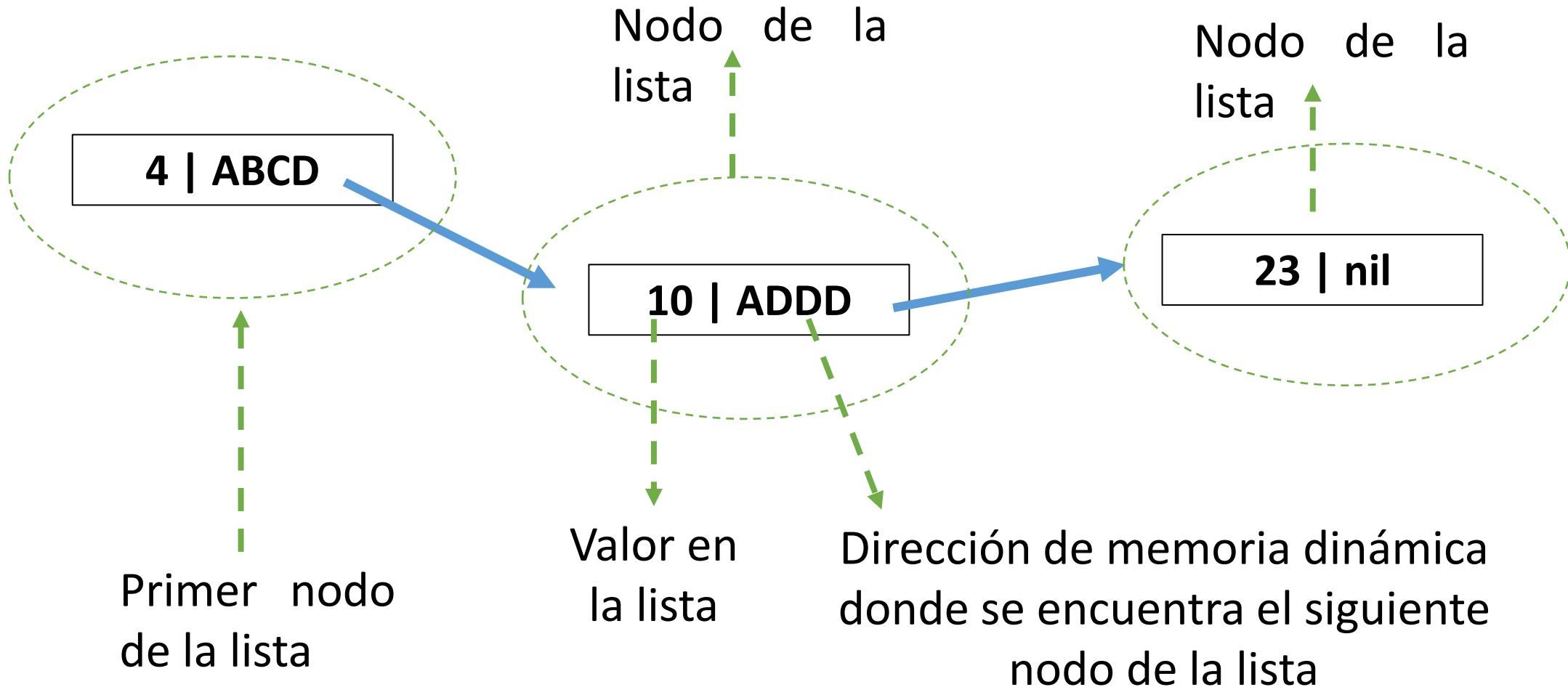


En **memoria estática** se declara una variable tipo PUNTERO (ya que son las únicas que pueden almacenar direcciones). La dirección almacenada en esa variable representa la dirección donde comienza la lista. Inicialmente ese puntero no contiene ninguna dirección.



Luego a medida que se quiere agregar elementos a la lista (nodo), se reserva una dirección de **memoria dinámica** y se carga el valor que se quiere guardar. El último nodo de la lista indica que la dirección que le sigue es nil.

CADP – TIPOS DE DATOS - LISTA





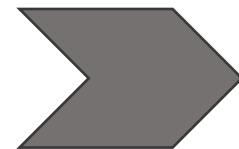
CADP – TIPOS DE DATOS - LISTA



CARACTERISTICAS

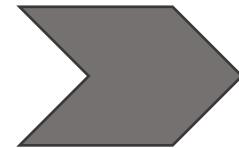
Cómo se
declara?

HOMOGENEA



Los elementos pueden ser del mismo tipo .

DINAMICA



El tamaño puede cambiar durante la ejecución del programa.

LINEAL



Cada nodo de la lista tiene un nodo que lo sigue (salvo el último) y uno que lo antecede (salvo el primero).

SECUENCIAL

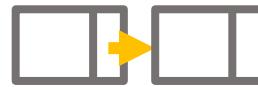


El acceso a cada elemento es de manera secuencial, es decir, para acceder al elemento 5 (por ejemplo) debo pasar por los 4 anteriores.



Cada vez que se necesite agregar un nodo se deberá reservar memoria dinámica (new) y cuando se quiera eliminar un nodo se debe liberar la memoria dinámica (dispose) .

CADP – TIPOS DE DATOS - LISTA DECLARACION



Program uno;

Type

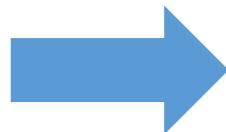
```
 nombreTipo= ^nombreNodo;
```

```
 nombreNodo = record
    elemento: tipoElemento;
    punteroSig: nombreTipo;
```

```
end;
```

Var

```
Pri: nombreTipo;
```



tipoElemento es cualquiera de los tipos vistos (entero,char,boolean,registro,arreglo,real,subrangol).

Es una estructura recursiva.

El orden de la declaración debe respetarse



```
Program uno;
```

```
Type
```

```
 listaE= ^nodo;
```

```
nodo = record
```

```
    elemento: integer;
```

```
    punteroSig: listaE;
```

```
end;
```

```
Var
```

```
Pri: listaE;
```



Pri



ABCD

23 | nil

ADDD

CADP – TIPOS DE DATOS - LISTA

DECLARACION



Program dos;

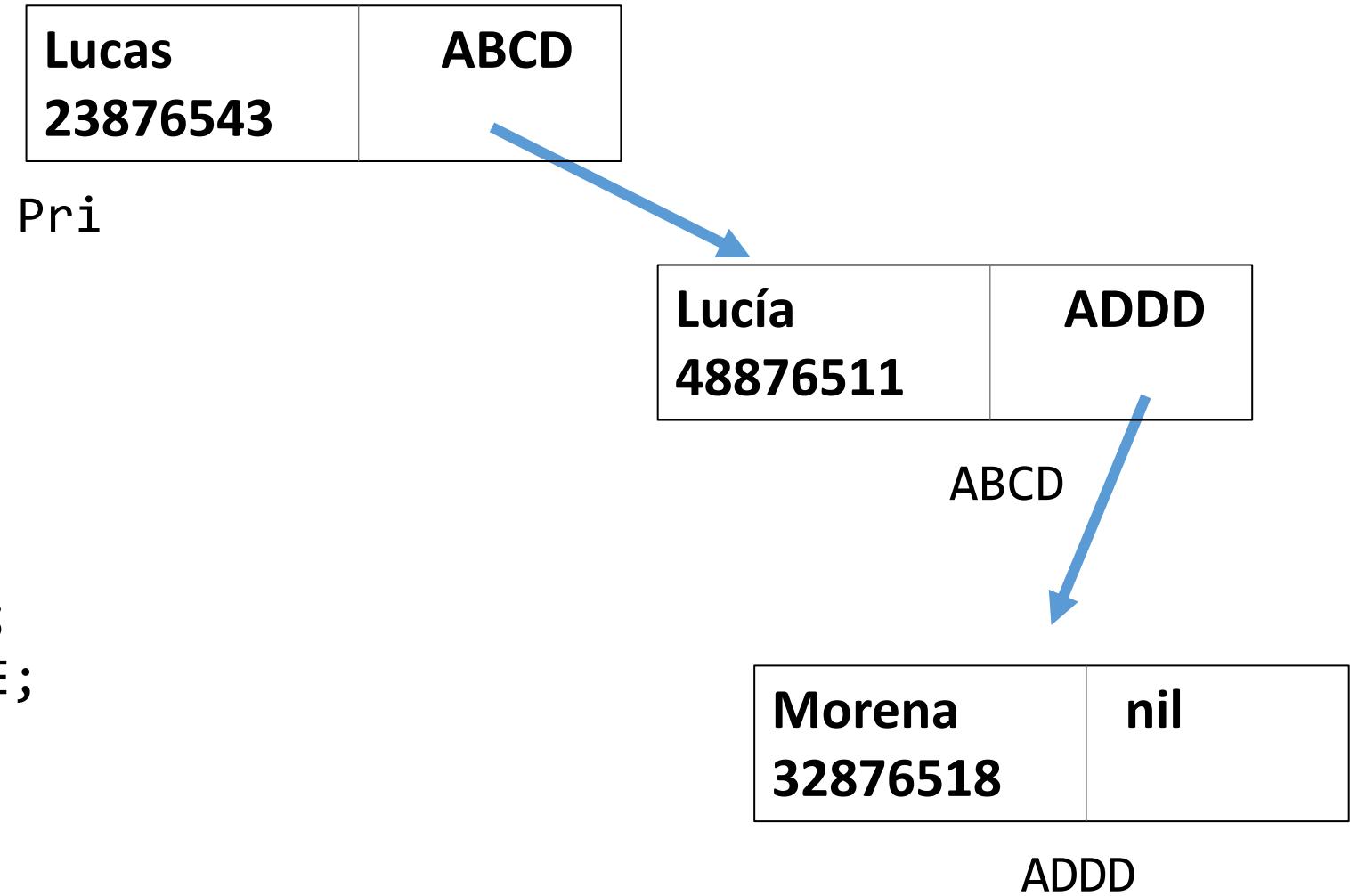
Type

```
persona = record
    nom:string;
    dni:integer;
end;
```

```
listaE= ^nodo;
nodo = record
    elemento: persona;
    punteroSig: listaE;
end;
```

Var

```
Pri: listaE;
```



CADP – TIPOS DE DATOS - LISTA

Creación de una lista.

Agregar nodos al comienzo de la lista.

Recorrido de una lista.

Agregar nodos al final de la lista.

Insertar nodos en una lista ordenada

Eliminar nodos de una lista





Conceptos de Algoritmos Datos y Programas

CADP – TIPOS DE DATOS - LISTA

Creación de una lista.

Agregar nodos al comienzo de la lista.

Recorrido de una lista.

Agregar nodos al final de la lista.

Insertar nodos en una lista ordenada

Eliminar nodos de una lista



CADP – TEMAS



● Estructura de Datos - LISTA

● Operación de CREACION

● Operación de RECORRIDO



CADP – TIPOS DE DATOS - LISTA



CREAR UNA LISTA

Implica marcar que la lista no tiene una dirección inicial de comienzo.

Qué valor se le asigna a un puntero para indicar que no tiene una dirección asignada?

Program uno;

Type listaE= ^datosEnteros;

datosEnteros= record
 elem:integer;
 sig:listaE;
end;

Var

pri: listaE; {Memoria estática reservada}

CADP – TIPOS DE DATOS - LISTA

CREAR



Program uno;

```
Type listaE= ^datosEnteros;  
  
    datosEnteros= record  
        elem:integer;  
        sig:listaE;  
    end;
```

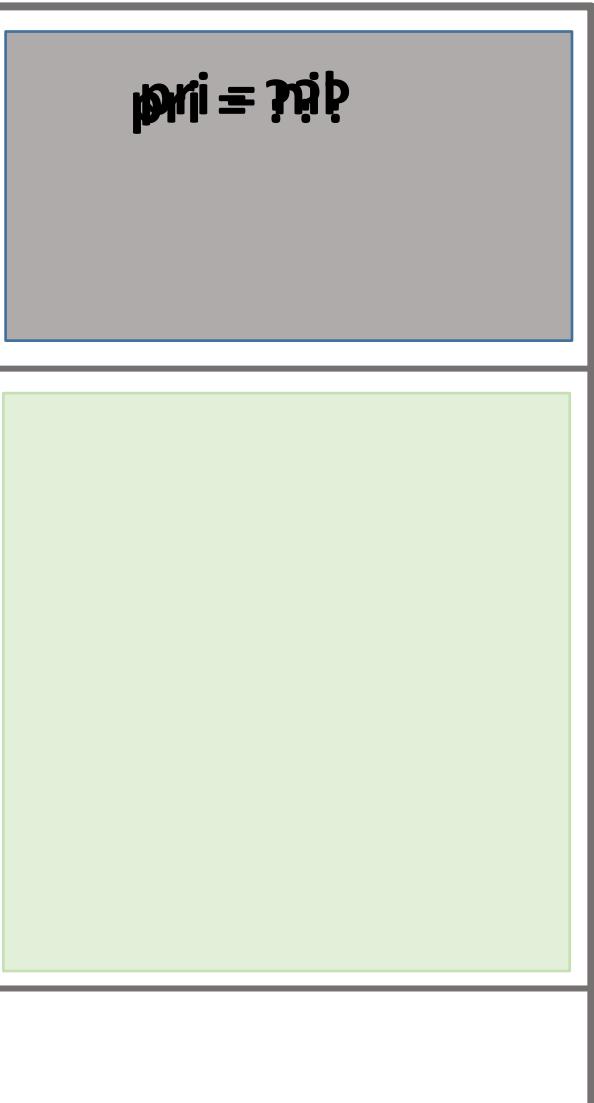
Var

```
    pri: listaE;
```

```
Begin  
    pri:=nil;  
End.
```

Por qué no se hace
new (pri)?

Se puede
modularizar el
crear?



CADP – TIPOS DE DATOS - LISTA

CREAR



Program uno;

Type listaE= ^datosEnteros;

```
    datosEnteros= record
        elem:integer;
        sig:listaE;
    end;
```

Procedure crear (var p: listaE);

begin
 p:= nil;
end;

Var
 pri: listaE;

Begin
 crear (pri);
End.

pri = nil?



CADP – TIPOS DE DATOS - LISTA



RECORRER UNA LISTA

Implica posicionarse al comienzo de la lista y a partir de allí ir “pasando” por cada elemento de la misma hasta llegar al final.

```
Program uno;
```

```
Type listaE= ^datosEnteros;
```

```
datosEnteros= record
    elem:integer;
    sig:listaE;
end;
```

```
Var
```

```
    pri: listaE;
```

CADP – TIPOS DE DATOS - LISTA

RECORRER UNA LISTA



Program uno;

```
Type listaE= ^datosEnteros;
```

```
    datosEnteros= record
        elem:integer;
        sig:listaE;
    end;
```

Var

```
    pri: listaE;
```

Begin

```
    crear (pri);
```

```
    cargarLista (pri); //Lo implementaremos más adelante
```

```
    recorrerLista (pri);
```

End.

ACDD

ADCD

ADDA

pri = ???

pri = nil

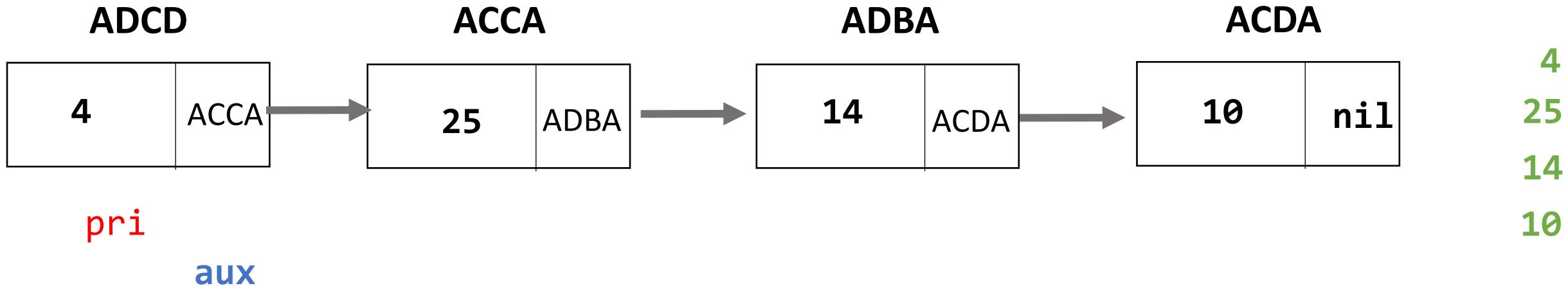
pri = ADCD

23 | ADDA

10 | ACDD

4 | nil





Inicialezo una variable **auxiliar** con la dirección del puntero inicial de la lista

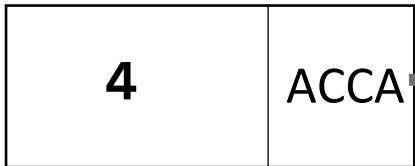
mientras (no sea el final de la lista)
proceso el elemento (ej: imprimo, sumo, modifico)
avanzo al siguiente elemento de **auxiliar**

CADP – TIPOS DE DATOS - LISTA

RECORRER UNA LISTA

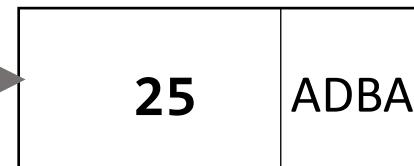


ADCD

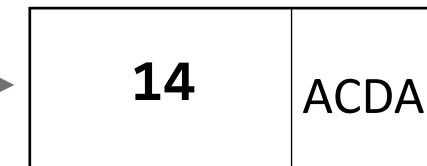


pri

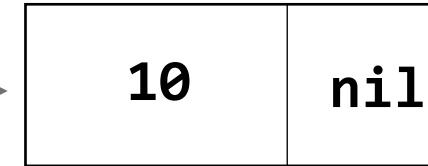
ACCA



ADBA



ACDA



```
procedure recorrerLista (pI: listaE);
```

Var

aux:listaE;

begin

aux:= pI;

while (aux^.sig <> nil) do

begin

write (aux^.elem);

aux:= aux^.sig;

end;

end;

Es correcto?



Si la lista está **vacía**, (aux^.sig) da error.

Si la lista tiene **un solo elemento** (aux^.sig <> nil) da falso.

Si la lista tiene **muchos elementos** no imprime el último

CADP – TIPOS DE DATOS - LISTA

RECORRER UNA LISTA

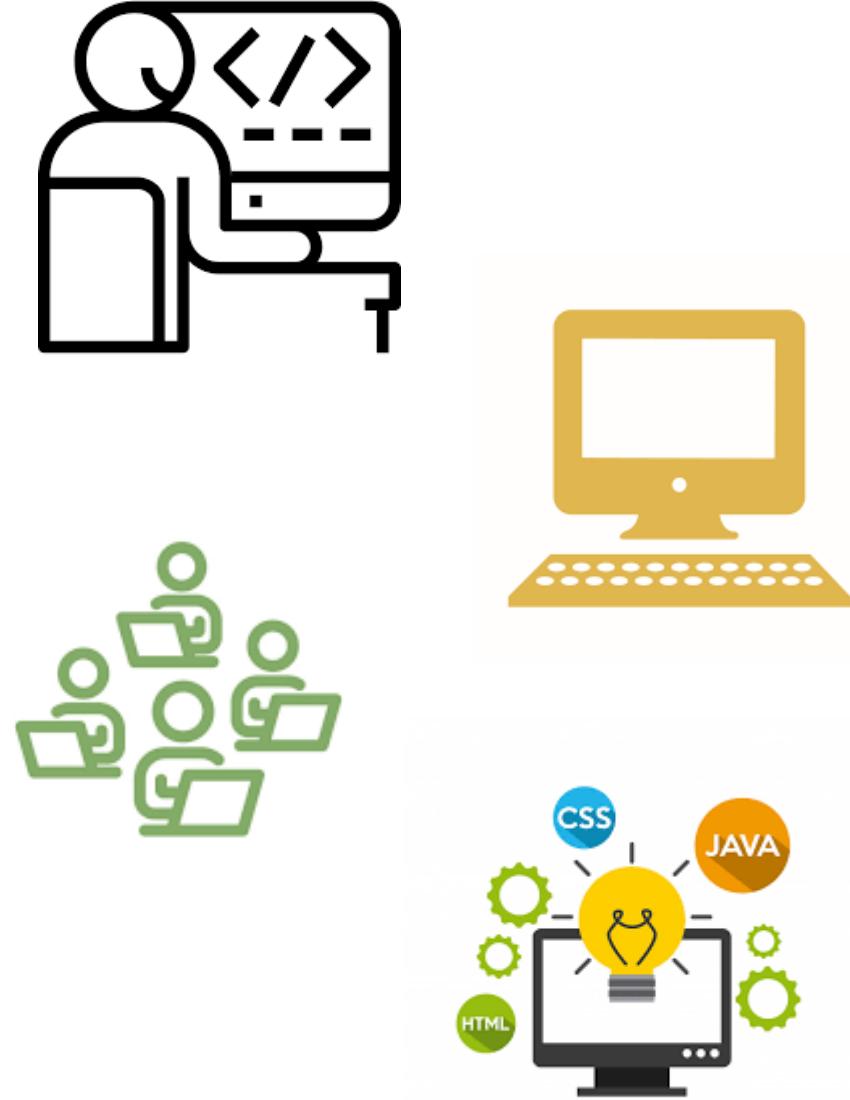


```
procedure recorrerLista (pI: listaE);  
Var  
  aux:listaE;  
begin  
  aux:= pI;  
  while (aux <> nil) do  
    begin  
      write (aux^.elem);  
      aux:= aux^.sig;  
    end;  
end;
```

Es necesaria
la variable
aux?

ALTERNATIVA

```
procedure recorrerLista (pI: listaE);  
begin  
  while (pI <> nil) do  
    begin  
      write (pI^.elem);  
      pI:= pI^.sig;  
    end;  
end;
```



Conceptos de Algoritmos Datos y Programas

CADP – TIPOS DE DATOS - LISTA

Creación de una lista.

Agregar nodos al comienzo de la lista.

Recorrido de una lista.

Agregar nodos al final de la lista.

Insertar nodos en una lista ordenada

Eliminar nodos de una lista



CADP – TEMAS



- Operación de AGREGAR ADELANTE
- Operación de AGREGAR AL FINAL

CADP – TIPOS DE DATOS - LISTA



AGREGAR ADELANTE

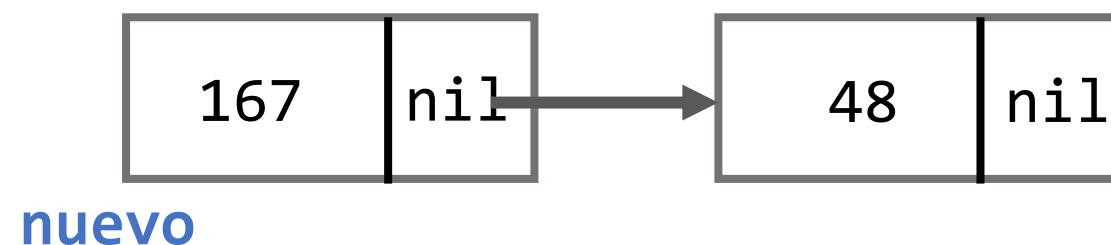
Implica generar un nuevo nodo y agregarlo como primer elemento de la lista.

➤ pri = nil

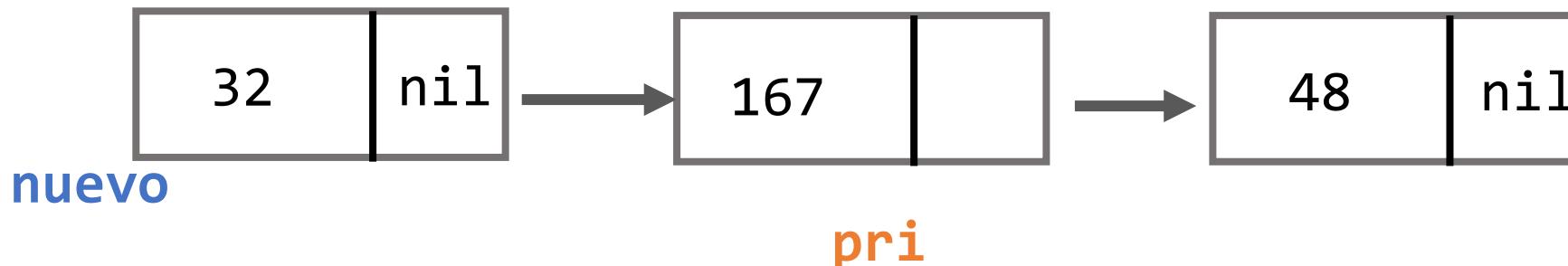


pri

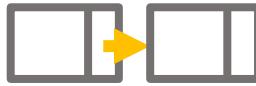
➤ pri <> nil



pri



Cómo lo
escribo?



Implica generar un nuevo nodo y agregarlo como primer elemento de la lista.

Reservo espacio en memoria **nuevo elemento**.

si (es el primer elemento a agregar)

asigno al puntero inicial la dirección del **nuevo elemento**.

sino

indico que el siguiente de **nuevo elemento** es el puntero inicial.

actualizo el puntero inicial de la lista con la dirección del **nuevo elemento**.

CADP – TIPOS DE DATOS - LISTA

AGREGAR ADELANTE



Program uno;

```
Type listaE= ^datosEnteros;
```

```
    datosEnteros= record
        elem:integer;
        sig:listaE;
    end;
```

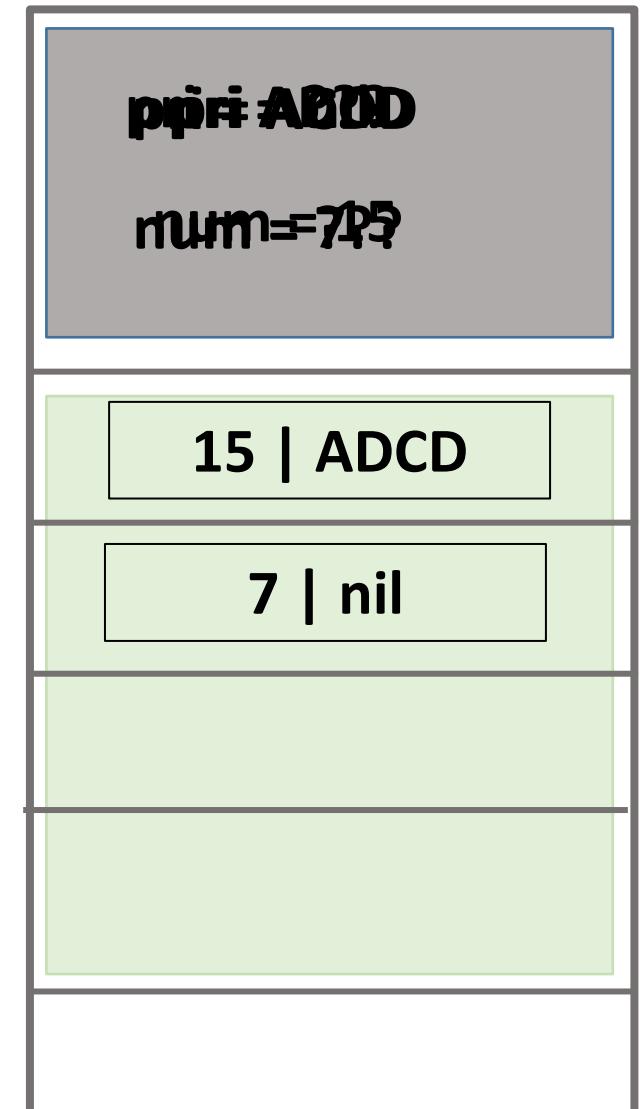
Var

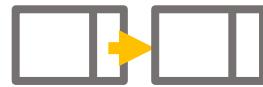
```
    pri: listaE;
    num:integer;
```

Begin

```
    crear (pri);
    read (num);
    agregarAdelante (pri,num);
    read (num);
    agregarAdelante (pri,num);
```

ACDD
ADCD
ADDA





```
procedure agregarAdelante (var pI:listaE; num:integer);  
Var  
    nuevo:listaE;   Creo espacio para el  
                    nuevo elemento  
Begin    new (nuevo); nuevo^.elem:= num; nuevo^.sig:=nil;  
        if (pI = nil) then pI:= nuevo  
        else begin  
            nuevo^.sig:= pI;  
            pI:=nuevo;  
        end;  
End;
```

Evalúo el caso y
reasigno los
punteros

CADP – TIPOS DE DATOS - LISTA



AGREGAR AL FINAL

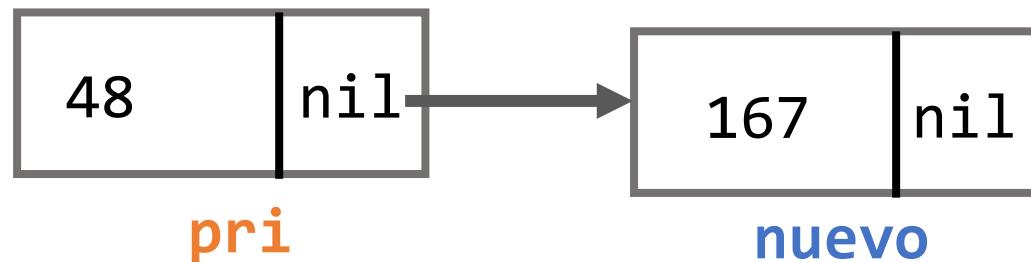
Implica generar un nuevo nodo y agregarlo como último elemento de la lista.

➤ pri = nil



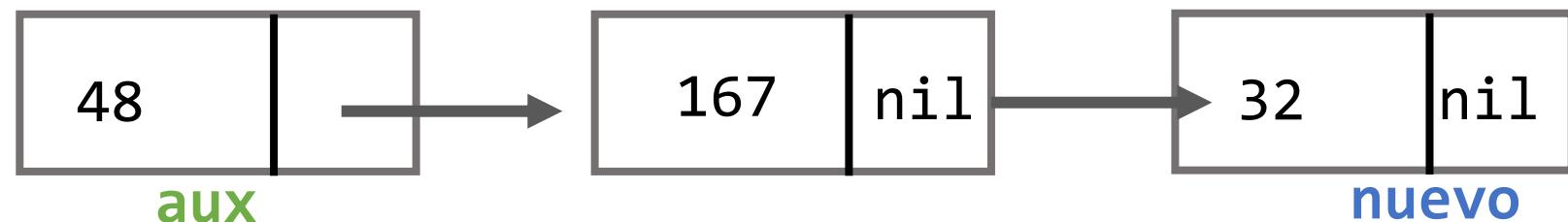
pri

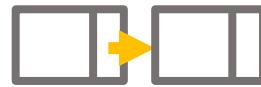
➤ pri <> nil



Cómo lo
escribo?

pri





Implica generar un nuevo nodo y agregarlo como último elemento de la lista.

Reservo espacio en memoria **nuevo elemento**.

si (es el primer elemento a agregar)
asigno al puntero inicial la dirección del **nuevo elemento**.

sino
 inicializo un puntero auxiliar **aux**
 mientras (no llegue al último elemento)
 avanzo en la lista.
 actualizo como siguiente del último nodo al **nuevo elemento**

CADP – TIPOS DE DATOS - LISTA

AGREGAR AL FINAL



Program uno;

```
Type listaE= ^datosEnteros;
```

```
    datosEnteros= record
        elem:integer;
        sig:listaE;
    end;
```

Var

```
    pri: listaE;
    num:integer;
```

Begin

```
    crear (pri);
```

```
    read (num);
```

```
    agregarAlFinal (pri,num);
```

```
    read (num);
```

```
    agregarAlFinal (pri,num);
```

ADDA

ADCD

ACDD

pri = ADD

num = 15

15 | nil

7 | ADD

CADP – TIPOS DE DATOS - LISTA AGREGAR AL FINAL



```
procedure agregarAlFinal (var pI:listaE; num:integer);
```

Var

```
nuevo,aux:listaE;
```

Begin

```
  new (nuevo); nuevo^.elem:= num; nuevo^.sig:=nil;
```

Si agrego al final por qué
paso por referencia el
puntero inicial?

Por qué en la
condición del while
se pregunta por el
aux^.sig?

```
  if (pI = nil) then pI:= nuevo
```

----- Evalúo si la lista está vacía

```
  else begin
```

```
    aux:= pI;
```

```
    while (aux ^.sig <> nil) do
```

```
      aux:= aux^.sig;
```

}

Re corro y quedo
parado en el último
elemento

```
    aux^.sig:=nuevo;
```

----- Le indico al último que ahora
su siguiente es nuevo

```
  end;
```

End;

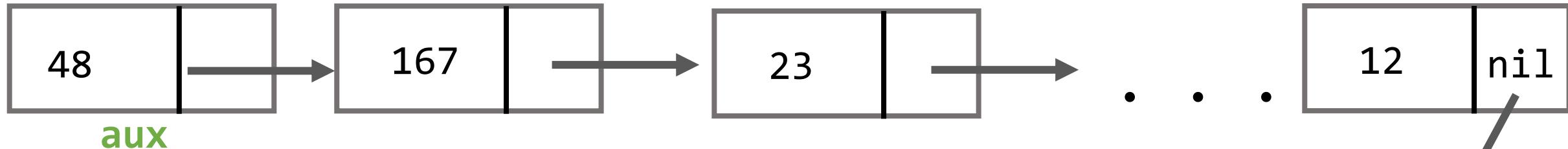


CADP – TIPOS DE DATOS - LISTA



AGREGAR AL FINAL EN UNA LISTA (OPCION 2)

pri



Cada vez que debo agregar un nuevo elemento al final de la lista, se tiene que recorrer la misma de manera completa hasta llegar al final.

SE PUEDE MEJORAR?

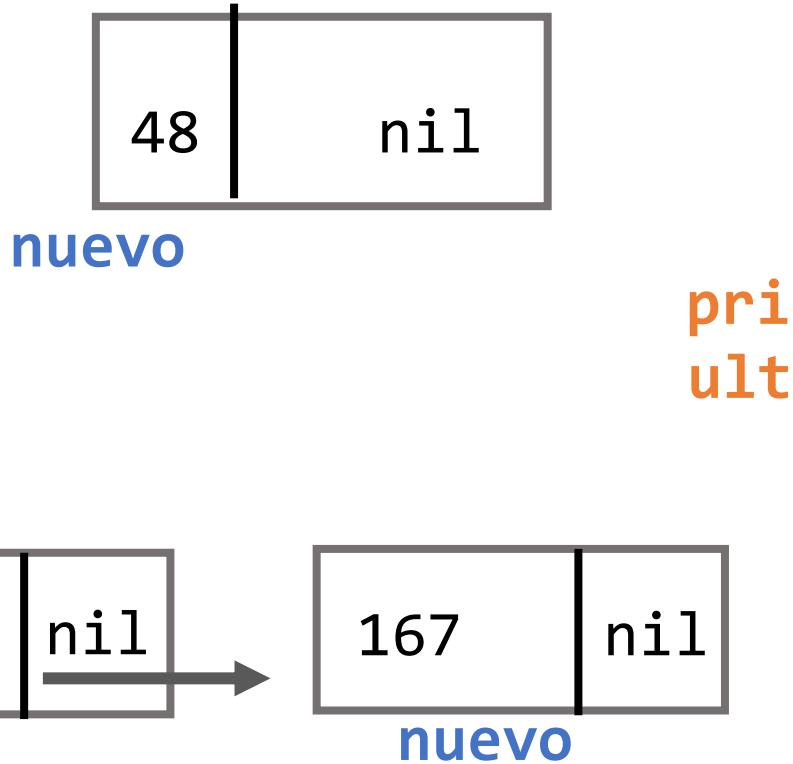
**Cómo lo
escribo?**

nuevo

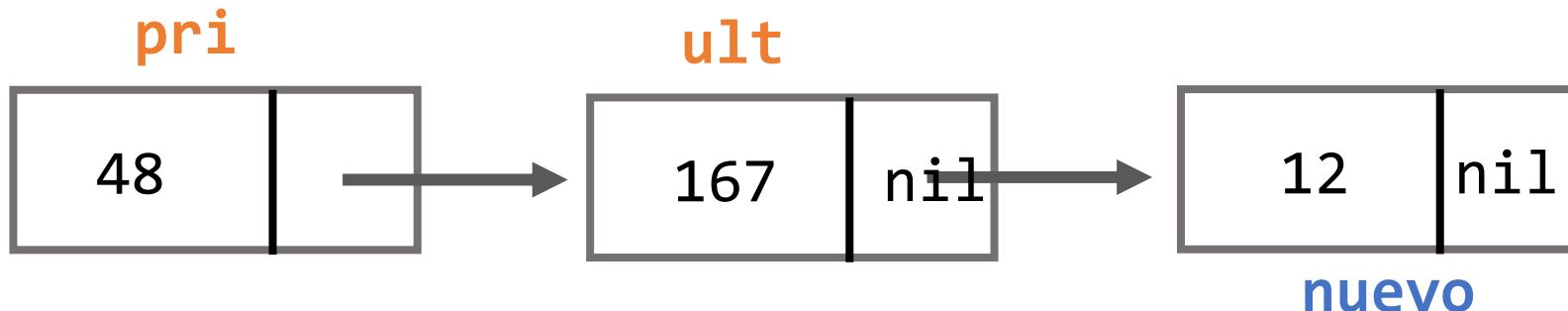
CADP – TIPOS DE DATOS - LISTA AGREGAR AL FINAL – 2



➤ pri = nil



➤ pri <> nil



Cómo lo
escribo?



AGREGAR AL FINAL EN UNA LISTA (OPCION 2)

Implica generar un nuevo nodo y agregarlo como último elemento de la lista.

Reservo espacio en memoria **nuevo elemento**.

si (es el primer elemento a agregar)

asigno al puntero inicial la dirección del **nuevo elemento**.

asigno al puntero final la dirección del **nuevo elemento**.

sino

actualizo como siguiente del puntero final al **nuevo elemento**

actualizo el la dirección del puntero final

CADP – TIPOS DE DATOS - LISTA

AGREGAR AL FINAL -2



Program uno;

```
Type listaE= ^datosEnteros;
```

```
    datosEnteros= record
        elem:integer;
        sig:listaE;
    end;
```

Var

```
    pri,ult: listaE;
    num:integer;
```

Begin

```
    crear (pri,ult);
```

```
    read (num);
```

```
    agregarAlFinal2 (pri,ult,num);
```

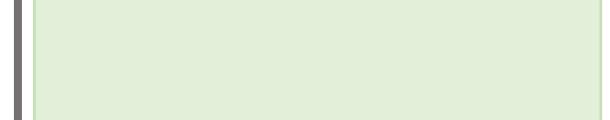
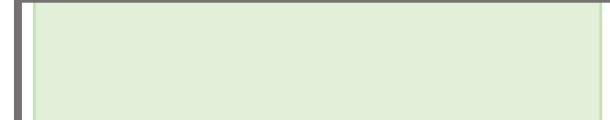
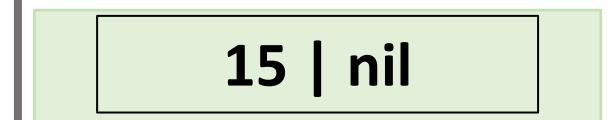
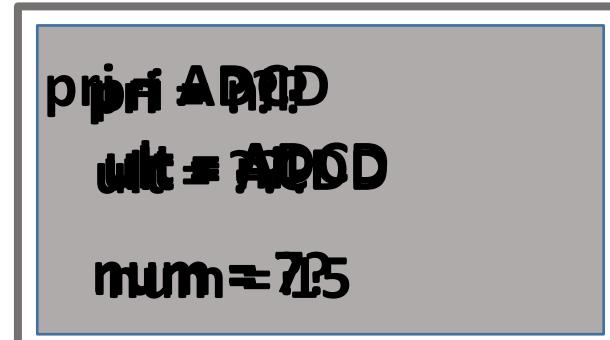
```
    read (num);
```

```
    agregarAlFinal2 (pri,,ult,num);
```

ACDD

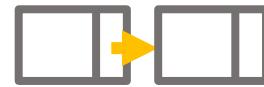
ADCD

ADDA



CADP – TIPOS DE DATOS - LISTA

AGREGAR AL FINAL-2



```
procedure agregarAlFinal2 (var pI,pU:listaE; num:integer);
Var
    nuevo:listaE;

Begin
    new (nuevo); nuevo^.elem:= num; nuevo^.sig:=nil;
    if (pI = nil) then begin
        pI:= nuevo;           ----- Evalúo si la lista está vacía
        pU:= nuevo;
    end
    else begin
        pU^.sig:=nuevo;
        pU:= nuevo;           ----- Actualizo el siguiente del
    end;                      último nodo y al último nodo
End;
```



Conceptos de Algoritmos Datos y Programas

CADP – TIPOS DE DATOS - LISTA

Creación de una lista.

Agregar nodos al comienzo de la lista.

Recorrido de una lista.

Agregar nodos al final de la lista.

Buscar un elemento en una lista

Eliminar un elemento de una lista

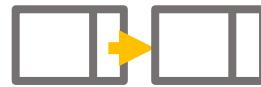
Insertar un elemento en una lista ordenada



CADP – TEMAS



Operación de BUSCAR un ELEMENTO



Significa recorrer la lista desde el primer nodo buscando un valor que puede o no estar. Se debe tener en cuenta si la lista está o no ordenada.

LISTA Desordenada

- Se debe recorrer toda la lista (en el peor de los casos), y detener la búsqueda en el momento que se encuentra el dato buscado o en el que la lista se terminó.

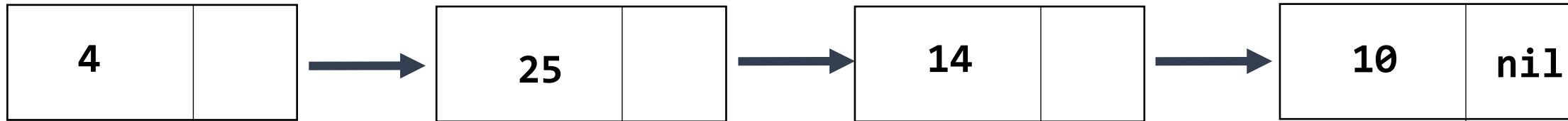
LISTA Ordenada

- Se debe recorrer la lista teniendo en cuenta el orden. La búsqueda se detiene cuando se termina la lista o el elemento buscado es mayor al elemento actual.

CADP – TIPOS DE DATOS - LISTA

BUSQUEDA LISTA DESORDENADA

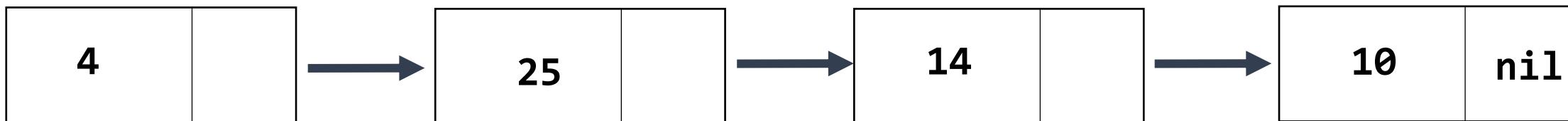
aux



Pri

num = 14

aux



Pri

num = 3



CADP – TIPOS DE DATOS - LISTA

BUSQUEDA LISTA DESORDENADA

Comienzo a recorrer la lista desde el nodo inicial.

mientras ((no sea el final de la lista)y(no encuentre el elemento))

si (es el elemento buscado) entonces

detengo la búsqueda

sino

avanzo al siguiente elemento

*Qué módulo
utilizo?*



Program uno;

Type listaE= ^datosEnteros;

```
datosEnteros= record
    elem:integer;
    sig:listaE;
end;
```

Var

```
pri: listaE;
num:integer;
```

Begin

```
crear (pri);
cargar (pri); //se dispone
read (num);
if (buscar(pri,num)) then write (“el elemento existe”);
```

End.



```
function buscar (pI: listaE; valor:integer):boolean;
Var
    aux:listaE;
    encontré:boolean;
Begin
    encontré:= false;
    aux:= pI;
    while ((aux <> nil) and (encontré = false)) do
        begin
            if (aux^.elem = valor) then
                encontré:=true
            else
                aux:= aux^.sig;
        end;
    buscar:= encontré;
end;
```

Funciona si la
lista que recibo
es vacía?

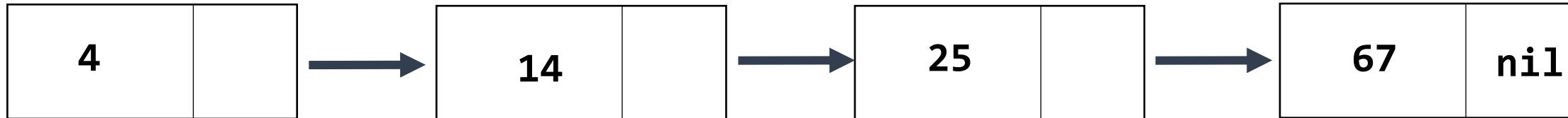
Necesito
usar aux?

Qué modiflico si la lista está
ordenada?

CADP – TIPOS DE DATOS - LISTA

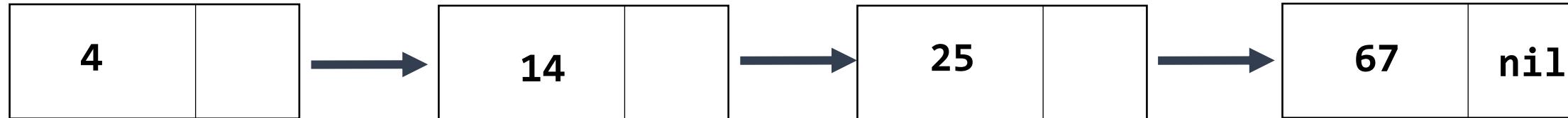
BUSQUEDA LISTA ORDENADA

aux



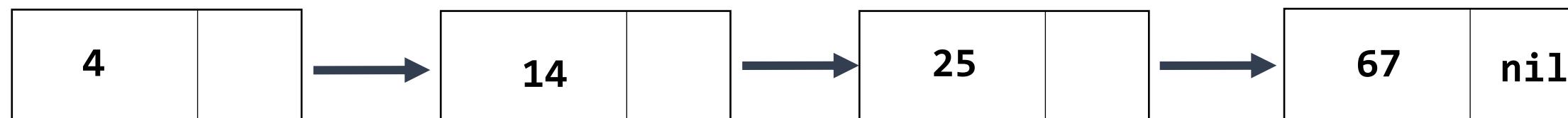
Pri num = 14

aux



Pri num = 20

aux



Pri num = 80



```
function buscar (pI: listaE; valor:integer):boolean;  
Var  
    aux:listaE;  
    encontré:boolean;  
  
Begin  
    encontré:= false;  
    aux:= pI;  
    while ((aux <> nil) and (aux^.elem < valor)) do  
        begin  
            aux:= aux^.sig;  
        end;  
  
        if (aux <> nil) and (aux^.elem = valor) then encontré:= true;  
  
    buscar:= encontré;  
end;
```

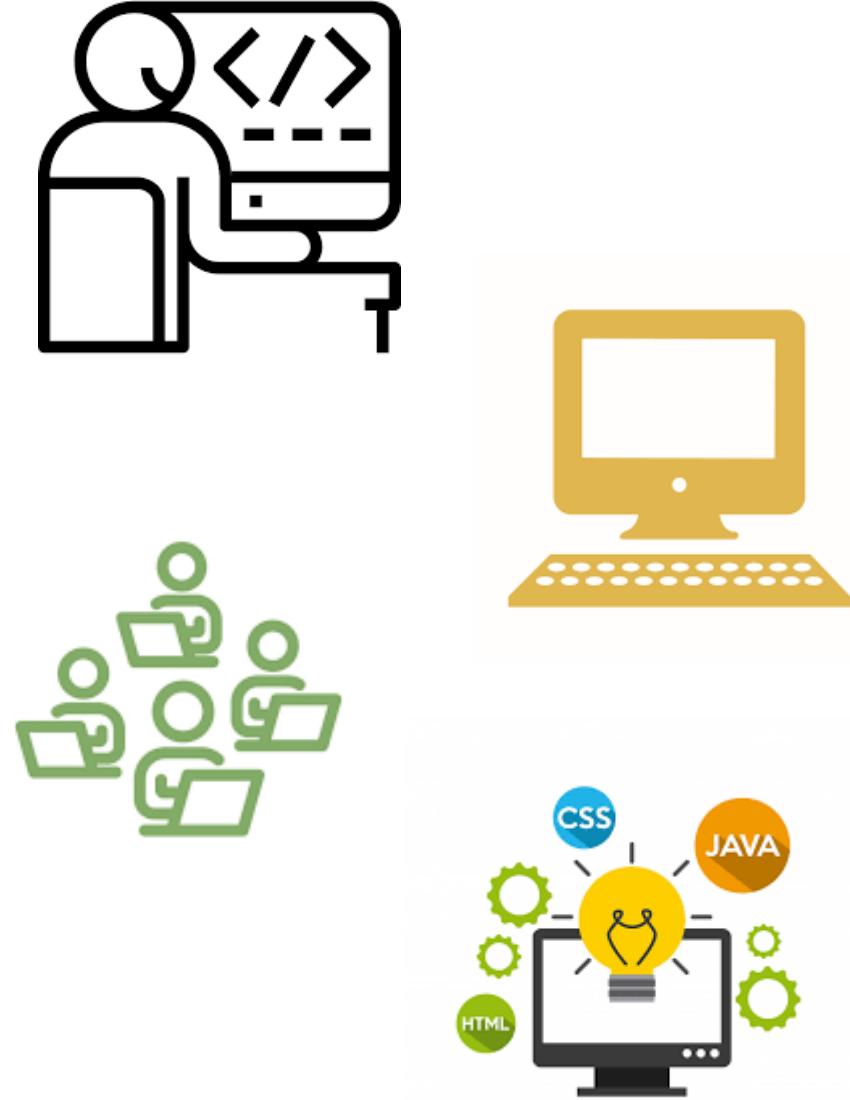
Funciona si la lista que
recibo es vacía?

Necesito usar
aux?

Es necesario respetar el
orden de las condiciones?

Necesito el chequeo
del final?

Buscar en una lista tiene las mismas
características que buscar en un vector



Conceptos de Algoritmos Datos y Programas

CADP – TIPOS DE DATOS - LISTA

Creación de una lista.

Agregar nodos al comienzo de la lista.

Recorrido de una lista.

Agregar nodos al final de la lista.

Buscar un elemento en una lista

Eliminar un elemento de una lista

Insertar un elemento en una lista ordenada



CADP – TEMAS



Operación de ELIMINAR un ELEMENTO



Implica recorrer la lista desde el comienzo pasando nodo a nodo hasta encontrar el elemento y en ese momento eliminarlo (`dispose`). El elemento puede no estar en la lista.

Si la lista está desordenada seguramente la búsqueda se realizará hasta encontrar el elemento o hasta que se termina la lista.

Si la lista está ordenada seguramente la búsqueda se realizará hasta que se termina la lista o no se encuentre un elemento mayor al buscado.

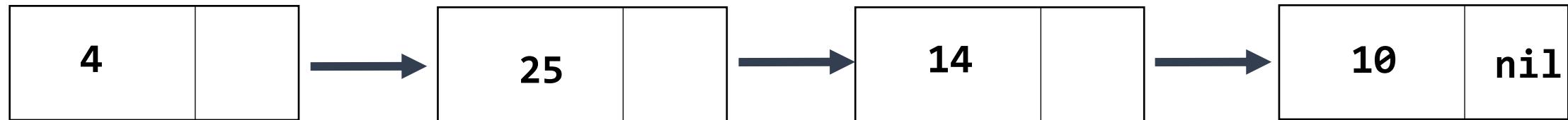
Existen 3 casos:

- que elemento a eliminar no se encuentre en la lista
- que elemento a eliminar sea el primero de la lista
- que elemento a eliminar no sea el primero en la lista



anterior

actual



Pri

num = 20

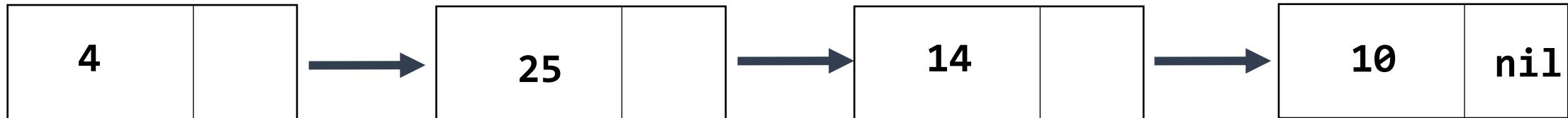
Caso 1:

Recorrió toda la lista y el elemento a eliminar no se encuentra.

OBSERVAR QUE actual QUEDÓ EN nil



anterior
actual



Pri

num = 4

Caso 2:

Empiezo a recorrer la lista.

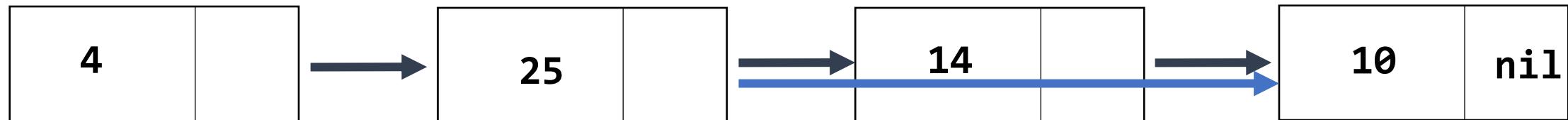
Mientras (no encuentro el elemento a borrar) y (no se termine la lista)
el puntero anterior toma la dirección del puntero actual
avanzo el puntero actual

Como (el elemento está) y (es el primer elemento)
actualizo el puntero inicial de la lista
elimino la dirección del puntero actual

OBSERVAR QUE actual HABIA QUEDADO IGUAL A pri



anterior
actual



Pri

num = 14

Caso 3:

Empiezo a recorrer la lista.

Mientras (no encuentro el elemento a borrar) y (no se termine la lista)
el puntero anterior toma la dirección del puntero actual
avanzo el puntero actual

Como (el elemento está) y (NO es el primer elemento)
actualizo el siguiente del puntero anterior con el siguiente de actual
elimino la dirección del puntero actual

OBSERVAR QUE actual HABIA QUEDADO <> nil y de pri



CADP – TIPOS DE DATOS - LISTA

ELIMINAR EN UN LISTA DESORDENADA

Comienzo a recorrer la lista desde el nodo inicial.

mientras ((no sea el final de la lista)y(no encuentre el elemento))

el puntero anterior toma la dirección del puntero actual
avanzo el puntero actual

si (encontré el elemento) entonces

 si (es el primer nodo) entonces

 actualizo el puntero inicial de la lista

 elimino la dirección del puntero actual

 sino

 actualizo el siguiente del puntero anterior con el siguiente de actual
 elimino la dirección del puntero actual





CADP – TIPOS DE DATOS - LISTA

ELIMINAR EN UN LISTA DESORDENADA

Comienzo a recorrer la lista desde el nodo inicial.

mientras ((no sea el final de la lista)y(no encuentre el elemento))

el puntero anterior toma la dirección del puntero actual
avanzo el puntero actual

si (encontré el elemento) entonces

 si (es el primer nodo) entonces

 actualizo el puntero inicial de la lista

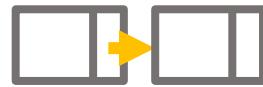
 sino

 actualizo el siguiente del puntero anterior con el siguiente de actual

 elimino la dirección del puntero actual

CADP – TIPOS DE DATOS - LISTA

ELIMINAR



Program uno;

Type listaE= ^datosEnteros;

```
datosEnteros= record
    elem:integer;
    sig:listaE;
end;
```

Var

```
pri: listaE;
num:integer;
```

Begin

```
crear (pri);
cargar (pri); //se dispone
read (num);
eliminar(pri,num);
```

End.



```
procedure eliminar (Var pI: listaE; valor:integer);
Var
    actual,ant:listaE;

Begin
    actual:=pI;
    while (actual <> nil) and (actual^.elem <> valor) do begin
        ant:=actual;
        actual:= actual^.sig;
    end;
    if (actual <> nil) then
        if (actual = pI) then
            pI:= pI^.sig;
        else
            ant^.sig:= actual^.sig;
    dispose (actual);
End;
```

Qué modiflico si el elemento
puede repetirse?

CADP – TIPOS DE DATOS - LISTA

ELIMINAR



```
procedure eliminar (Var pI: listaE; valor:integer);
```

```
Var
```

```
    actual,ant:listaE;
```

```
Begin
```

```
    actual:=pI;
```

```
    while (actual <> nil) do begin
```

```
        if (actual^.elem <> valor) then begin
```

```
            ant:=actual; actual:= actual^.sig;
```

```
        end;
```

```
        else begin
```

```
            if (actual = pI) then
```

```
                pI:= pI^.sig;
```

```
            else
```

```
                ant^.sig:= actual^.sig;
```

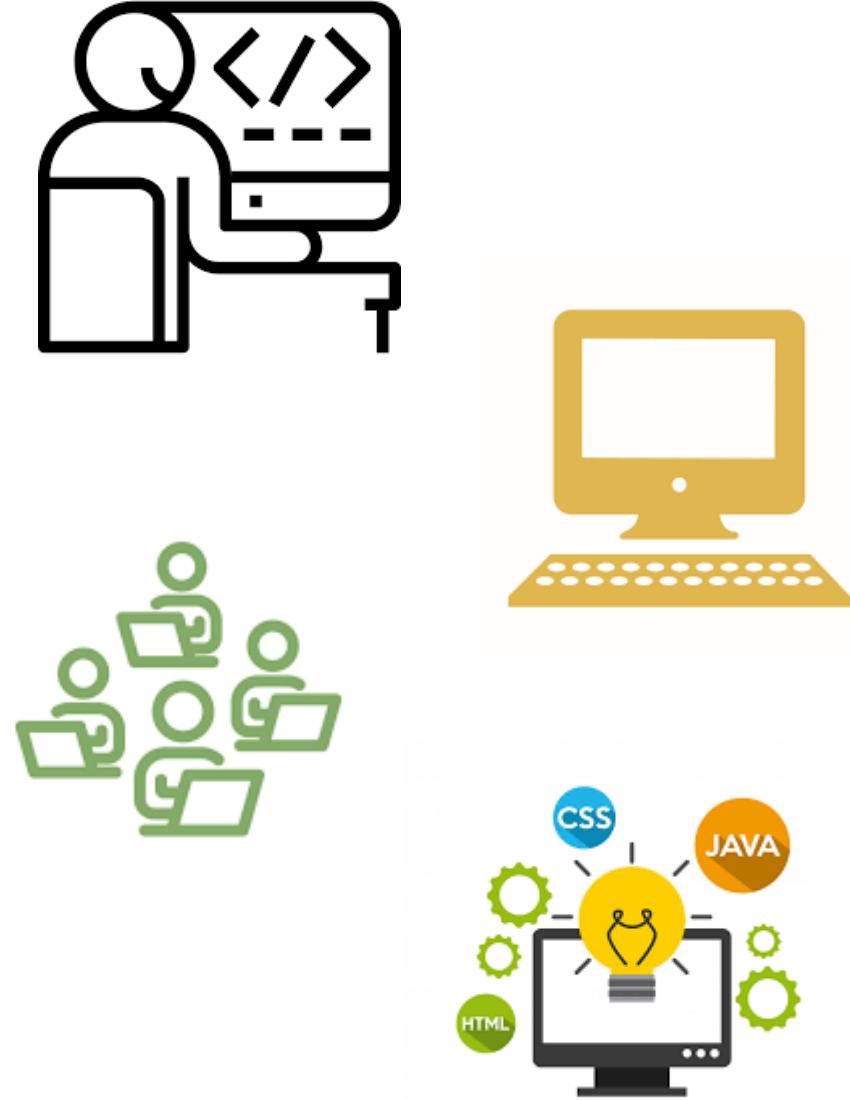
```
                dispose (actual);
```

```
                actual:= ant;
```

```
        end;
```

```
End;
```

Qué modifco si la lista está ordenada y ele elemento un única vez ?



Conceptos de Algoritmos Datos y Programas

CADP – TIPOS DE DATOS - LISTA

Creación de una lista.

Agregar nodos al comienzo de la lista.

Recorrido de una lista.

Agregar nodos al final de la lista.

Buscar un elemento en una lista

Eliminar un elemento de una lista

Insertar un elemento en una lista ordenada



CADP – TEMAS



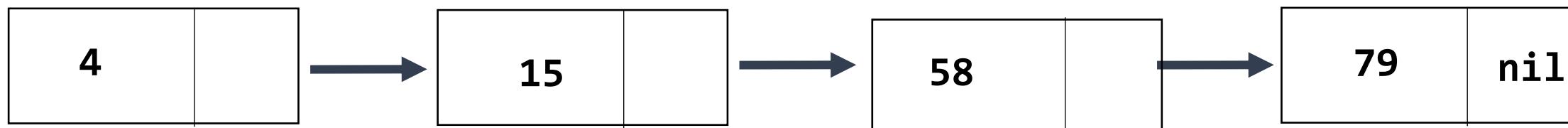
Operación de INSERTAR un ELEMENTO



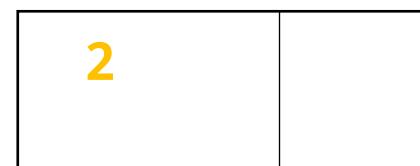
Implica agregar un nuevo nodo a una lista ordenada por algún criterio de manera que la lista siga quedando ordenada.

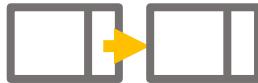
Existen 4 casos:

- que la lista esté vacía.
- que elemento vaya al comienzo de la lista (es menor al 1er nodo de la lista)
- que elemento vaya al “medio” de la lista (es menor al último nodo de la lista)
- que elemento vaya al final de la lista (es mayor al último nodo de la lista)



Pri





CASO 1: lista vacía

Pri = nil

4	nil
---	-----

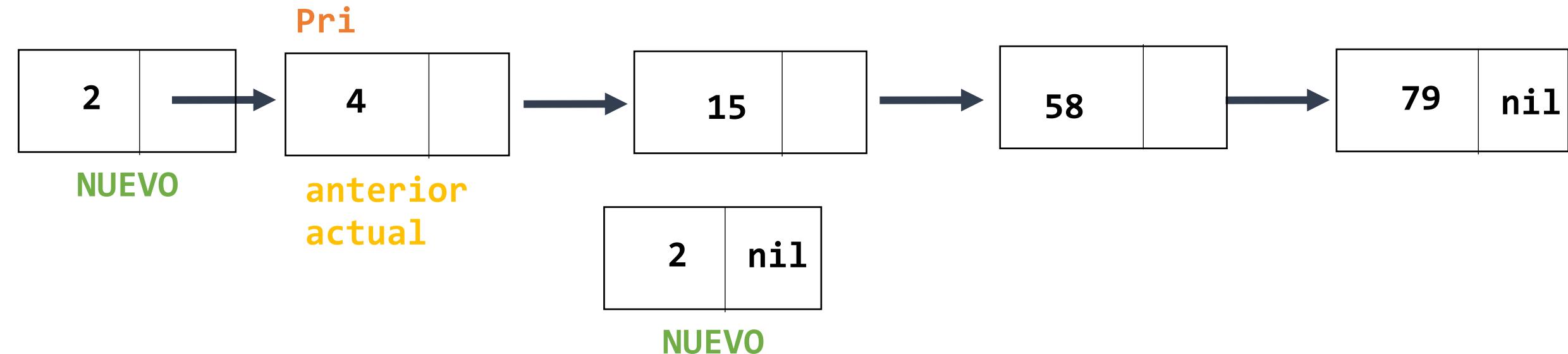
NUEVO

Generar un nuevo nodo (NUEVO).

Asignar a la dirección del puntero inicial (PI) la del nuevo nodo (NUEVO)



CASO 2: lista no vacía, va al principio



Generar un nuevo nodo (nuevo). Preparar punteros para el recorrido. Asignar a la dirección del puntero siguiente del nuevo la dirección del nodo inicial (PI).

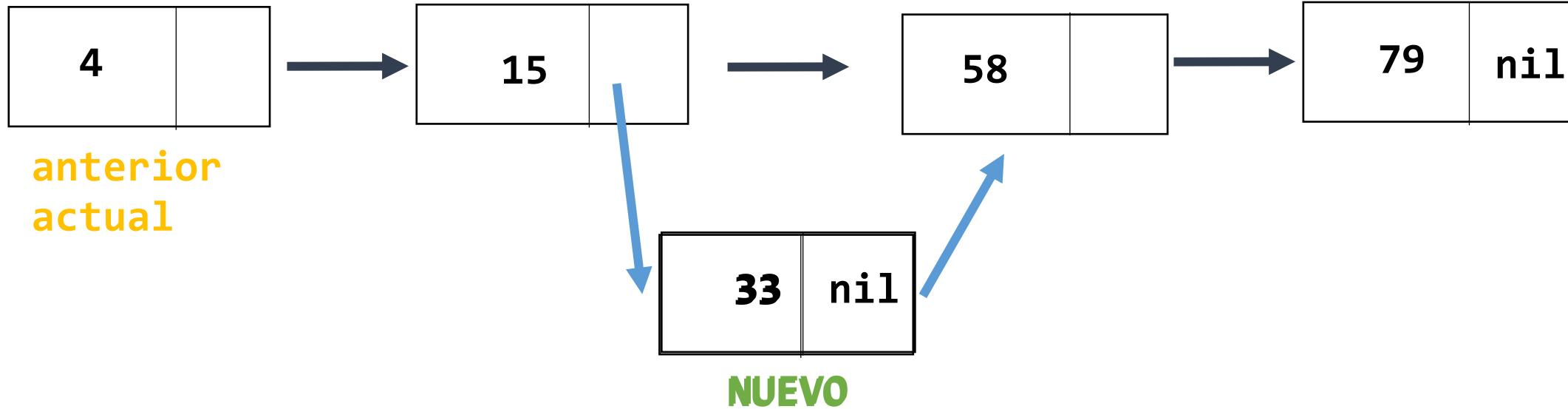
Actualizar con la dirección del nuevo nodo la dirección del puntero inicial (PI)

OBSERVAR QUE actual HABIA QUEDADO = pri



CASO 3: lista no vacía, va en el “medio”

Pri



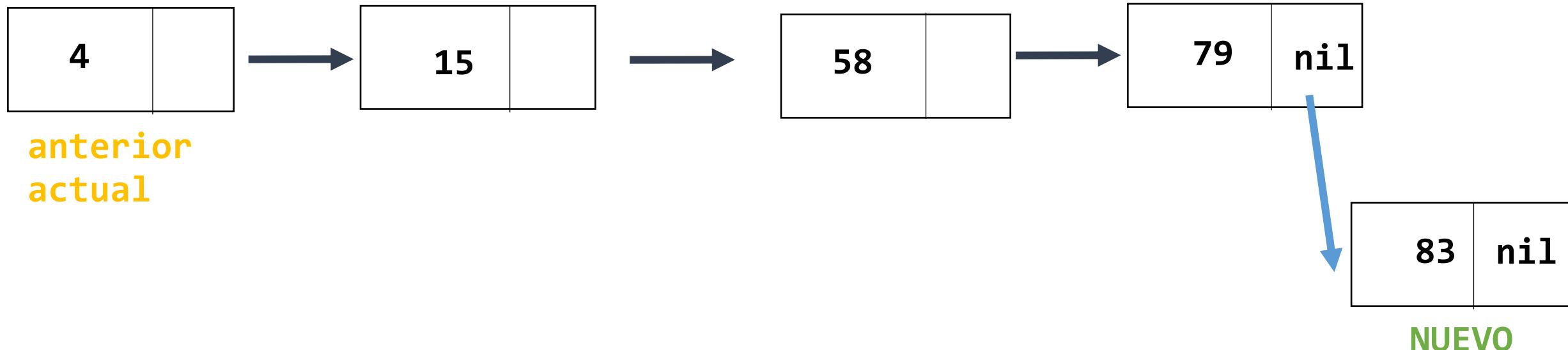
Generar un nuevo nodo (nuevo). Preparo los punteros para el recorrido
Re corro hasta encontrar la posición
Reasigno punteros, el siguiente de anterior es NUEVO y el siguiente
de NUEVO es actual.

OBSERVAR QUE actual HABIA QUEDADO <> nil



CASO 4: lista no vacía, va al final

Pri



Generar un nuevo nodo (nuevo). Preparo los punteros para el recorrido
Re corro hasta encontrar la posición
Reasigno punteros, el siguiente de anterior es NUEVO y el siguiente
de NUEVO es nil.

OBSERVAR QUE actual HABIA QUEDADO = nil

CADP – TIPOS DE DATOS - LISTA

INSERTAR



Generar un nuevo nodo (**NUEVO**).

Si la lista está vacía

Actualizo la dirección del nodo inicial (pri)

Caso 1 pri=nil

Sino

Preparo los punteros para el recorrido (anterior,actual)

Re corro hasta encontrar la posición.

Si va al principio

Asigno como siguiente del nodo nuevo al nodo inicial

Actualizo la dirección del nodo inicial (pri)

Caso 2 actual=pri

Si va en el medio

La dirección del siguiente del puntero anterior es la dirección del nodo nuevo

La dirección del siguiente del nodo nuevo es la dirección del actual

Caso 3 actual <> nil

sino

La dirección del siguiente del puntero anterior es la dirección del nodo nuevo

Clase 10-3 La dirección del siguiente del nodo nuevo es la dirección nil

Caso 4 actual <> nil

CADP – TIPOS DE DATOS - LISTA

INSERTAR



```
Program uno;

Type listaE= ^datosEnteros;

    datosEnteros= record
        elem:integer;
        sig:listaE;
    end;

Var
    pri: listaE;
    num:integer;

Begin
    crear (pri);
    cargar (pri); //se dispone
    read (num);
    insertar(pri,num);
End.
```

CADP – TIPOS DE DATOS - LISTA

INSERTAR



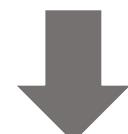
```
procedure insertar (Var pI: listaE; valor:integer);
Var
    actual,anterior,nuevo:listaE;
Begin
    new (nuevo); nuevo^.elem:= valor; nuevo^.sig:=nil;
    if (pI = nil) then
        pI:= nuevo
    else begin
        actual:= pI; ant:=pI;
        while (actual <> nil) and (actual^.elem < nuevo^.elem) do
            begin
                anterior:=actual;
                actual:= actual^.sig;
            end;
    end;
end;
```

}

Caso 1 pri=nil

}

BUSCO LA
POSICION



CADP – TIPOS DE DATOS - LISTA

INSERTAR



```
if (actual = pI) then
begin
    nuevo^.sig:= pI;
    pI:= nuevo;
end

else if (actual <> nil) then
begin
    anterior^.sigg=nuevo;
    nuevo^.sigg=actual;
end;

End;
else
begin
    anterior^.sig:= nuevo;
    nuevo^.sig:= actual;
end;
End;
```

}

Caso 2
pri=actual

}

Casos y 4
actual <> nil

}

Caso 4
actual = nil



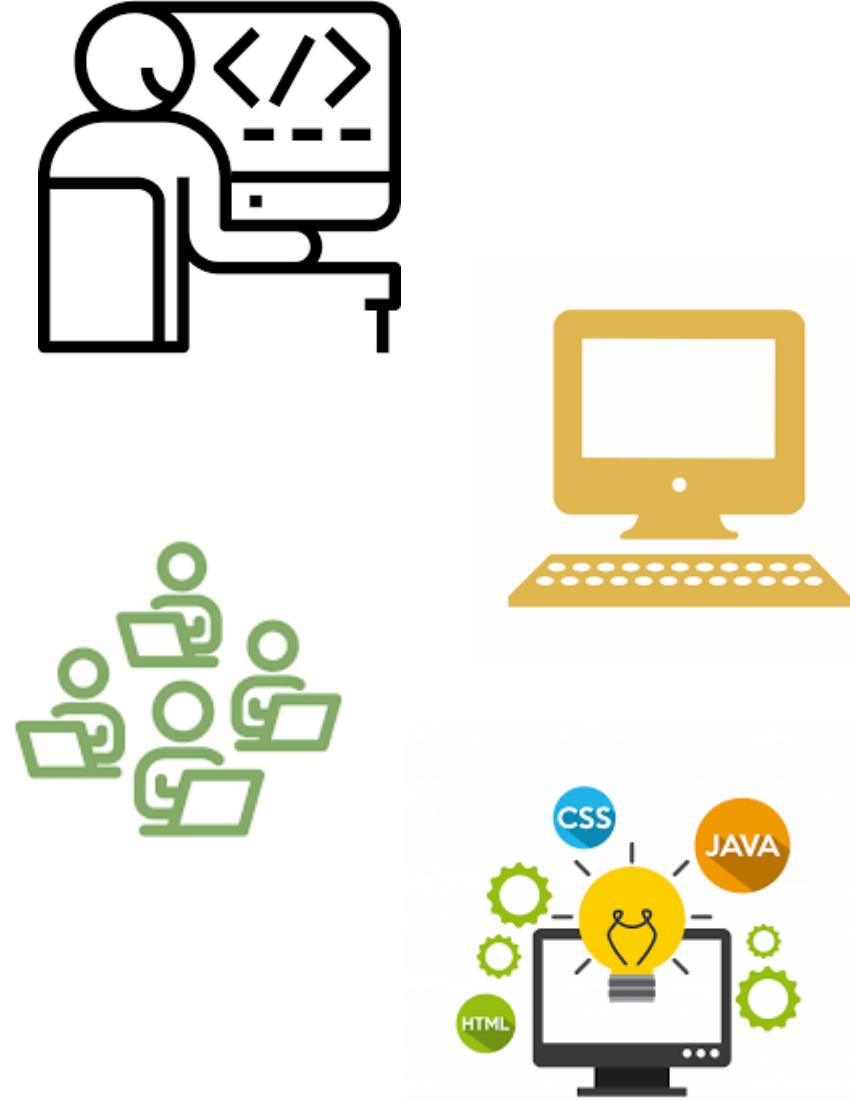
En el caso 4
cuánto vale
actual?

CADP – TIPOS DE DATOS - LISTA

INSERTAR



```
procedure insertar (Var pI: listaE; valor:integer);
Var
  actual,anterior,nuevo:listaE;
Begin
  new (nuevo); nuevo^.elem:= valor; nuevo^.sig:=nil;
  if (pI = nil) then      pI:= nuevo
  else begin
    actual:= pI; ant:=pI;
    while (actual <> nil) and (actual^.elem < nuevo^.elem) do
      begin
        anterior:=actual;
        actual:= actual^.sig;
      end;
    end;
    if (actual = pI) then
      begin
        nuevo^.sig:= pI;    pI:= nuevo;
      end
    else
      begin
        anterior^.sig:= nuevo;    nuevo^.sig:= actual;
      end;
  End;
```



Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



Corrección de Programas

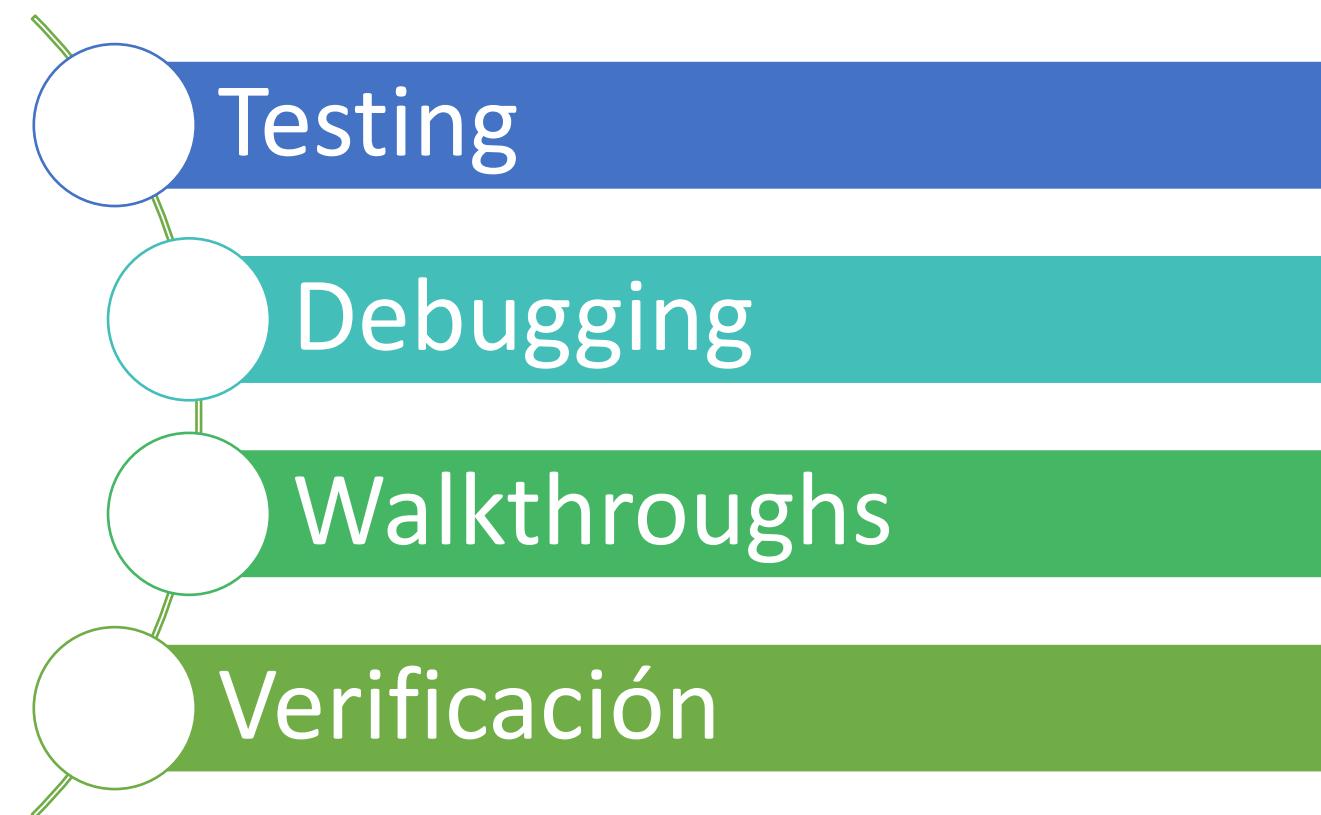
CADP – CORRECCION DE PROGRAMAS



Cuando se desarrollan los algoritmos hay dos conceptos importantes que se deben tener en cuenta: CORRECCIÓN y EFICIENCIA del programa.

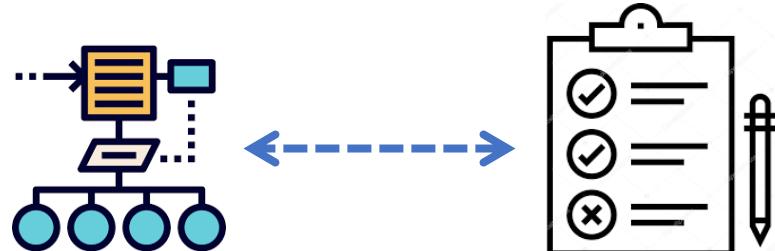
Un programa es correcto si se realiza de acuerdo a sus especificaciones.

Técnicas para
corrección de
programas





El propósito del Testing es proveer evidencias convincentes que el programa hace el trabajo esperado.

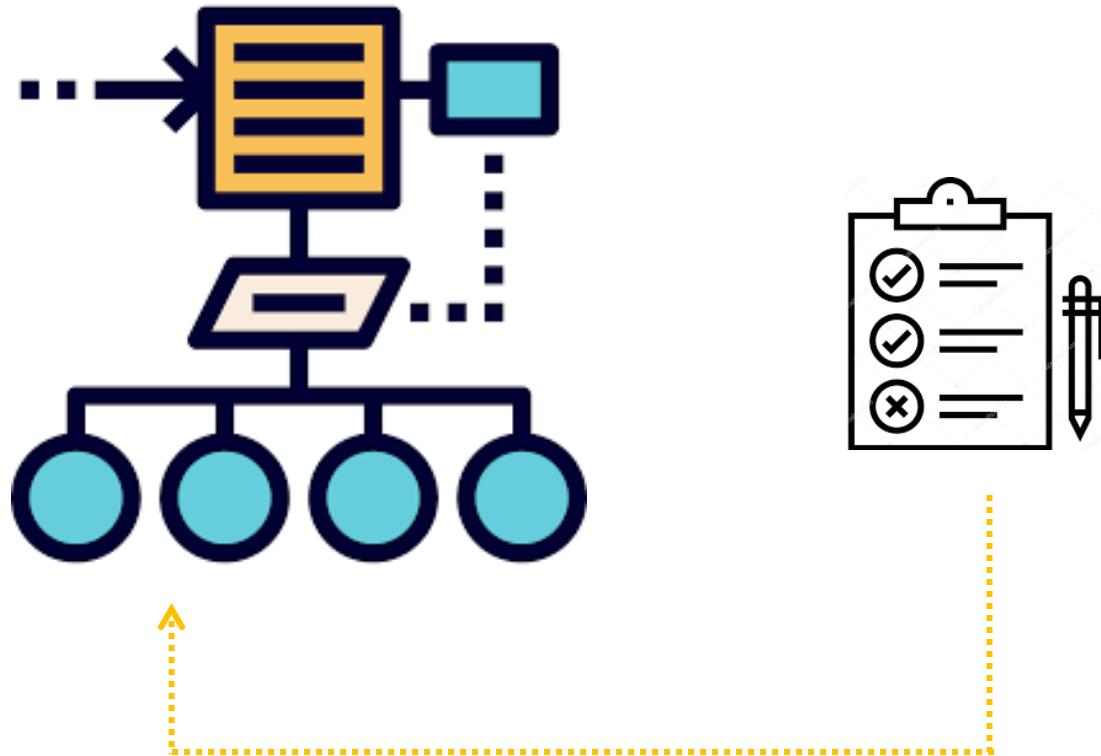


Diseñar un plan de pruebas

- Decidir cuales aspectos del programa deben ser testeados y encontrar datos de prueba para cada uno de esos aspectos.
- Determinar el resultado que se espera que el programa produzca para cada caso de prueba.
- Poner atención en los casos límite.
- Diseñar casos de prueba sobre la base de lo que hace el programa y no de lo que se escribió del programa. Lo mejor es hacerlo antes de escribir el programa.



Una vez que el programa ha sido implementado y se tiene el plan de pruebas:



- Se analiza el programa con los casos de prueba.
- Si hay errores se corrigen.

Estos dos pasos se repiten hasta que no haya errores.



Es el proceso de descubrir y reparar la causa del error.

Para esto pueden agregarse sentencias adicionales en el programa que permiten monitorear el comportamiento más cercanamente.



Los errores encontrados pueden ser de tres tipos



SINTACTICOS: Se detectan en la compilación



LOGICOS: Generalmente se detectan en la ejecución



DE SISTEMA: Son muy raros los casos en los que ocurren



Es el proceso de recorrer un programa frente a una audiencia.

La lectura de un programa a alguna otra persona provee un buen medio para detectar errores.



Esta persona no comparte preconceptos y está predisposta a descubrir errores u omisiones.

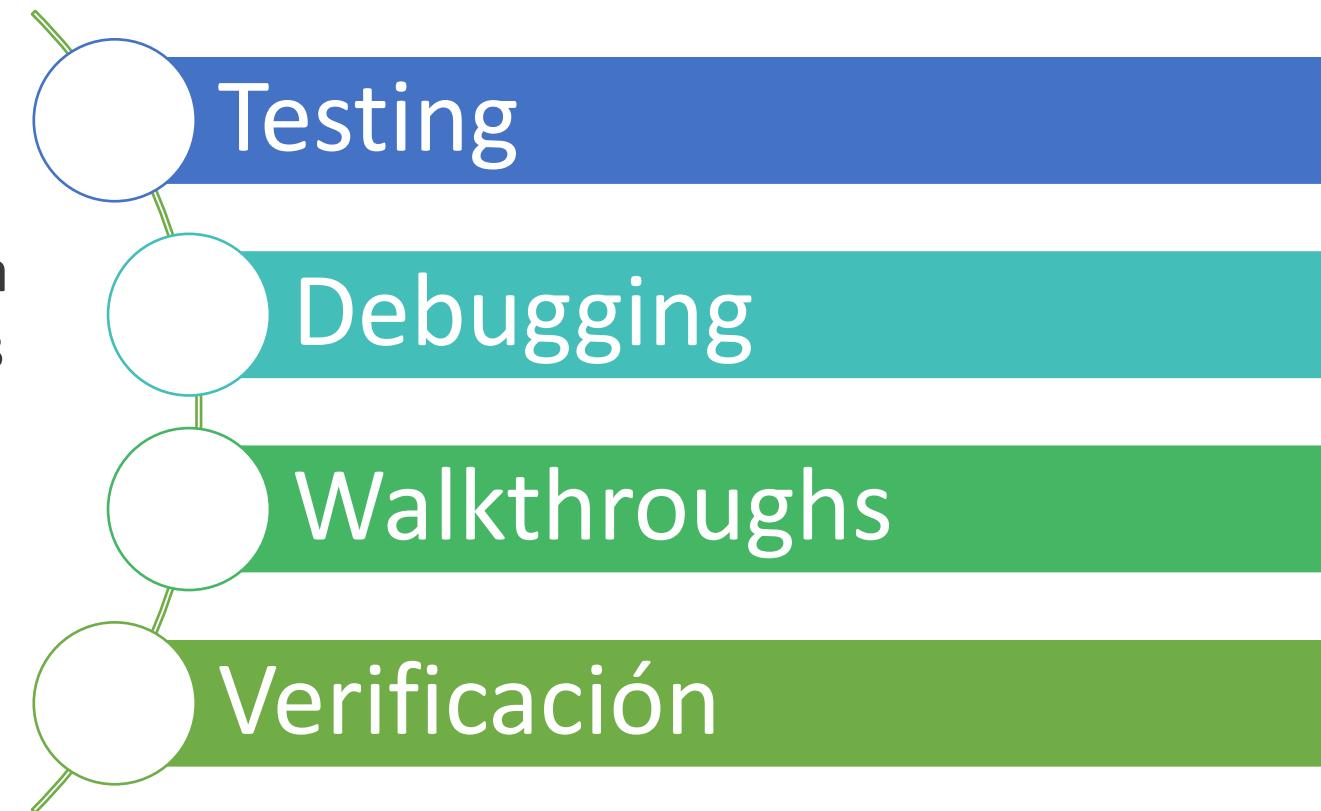


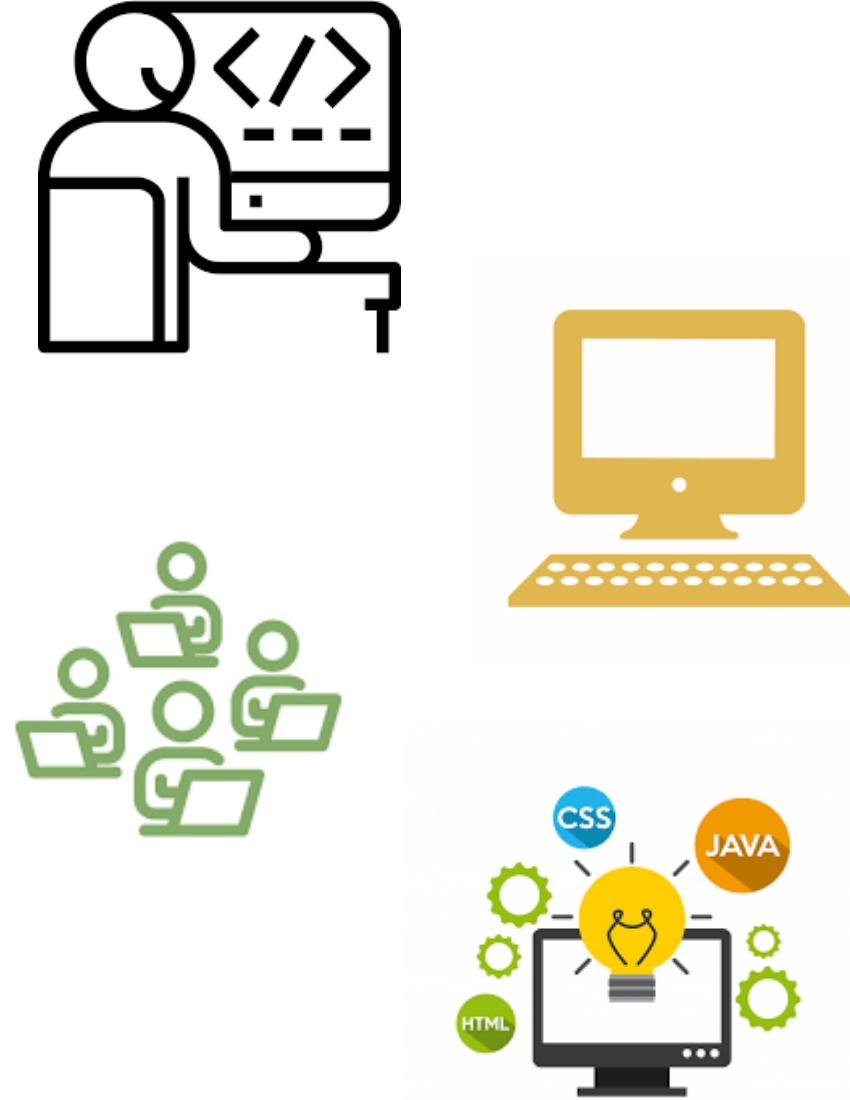
A menudo, cuando no se puede detectar un error, el programador trata de probar que no existe, pero mientras lo hace, puede detectar el error, o bien puede que el otro lo encuentre.



Es el proceso de controlar que se cumplan las pre y post condiciones del mismo.

Para determinar la corrección de un programa puedo utilizar una o varias técnicas de corrección la cantidad de veces necesarias hasta que el programa sea correcto





Conceptos de Algoritmos Datos y Programas

CADP – TEMAS



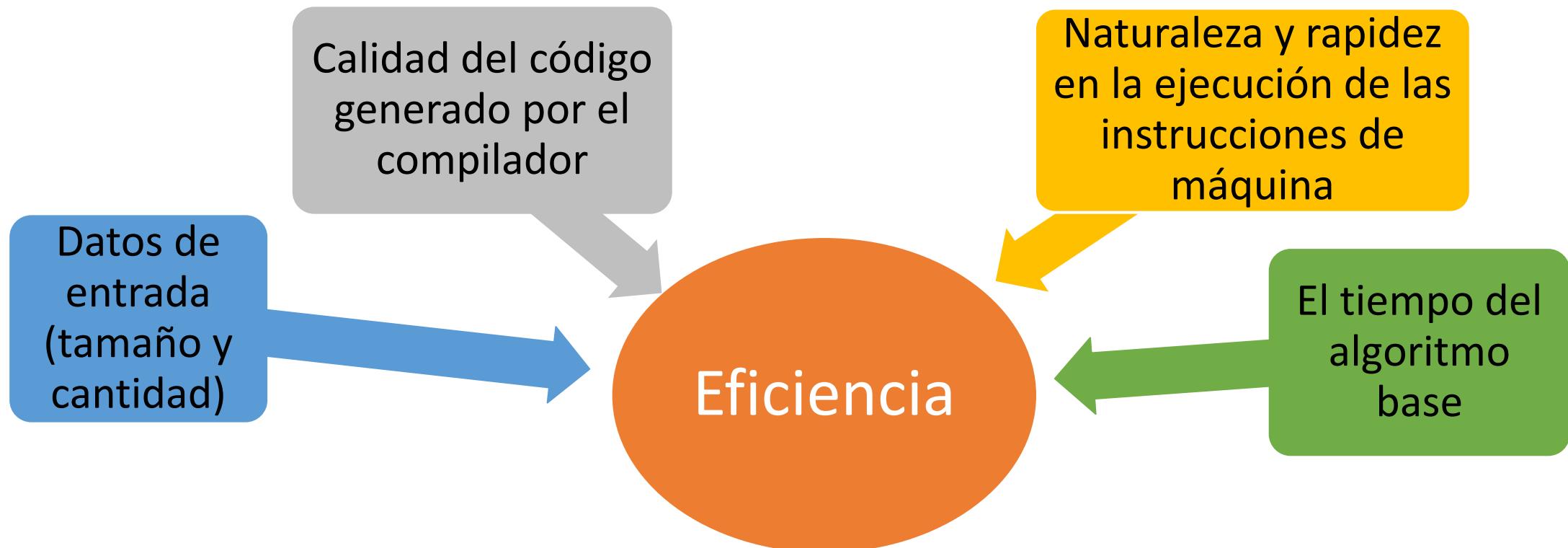
Eficiencia de Programas

CADP – EFICIENCIA DE PROGRAMAS



Una vez que se obtiene un algoritmo y se verifica que es correcto, es importante determinar la eficiencia del mismo.

El análisis de la eficiencia de un algoritmo estudia el **tiempo de ejecución** de un algoritmo y la **memoria** que requiere para su ejecución.



CADP – EFICIENCIA DE PROGRAMAS



El análisis de la eficiencia de un algoritmo estudia el **tiempo de ejecución** de un algoritmo y la **memoria** que requiere para su ejecución.

Los factores que afectan la eficiencia de un programa

Como se
miden?



MEMORIA: Se calcula (como hemos visto previamente) teniendo en cuenta la cantidad de bytes que ocupa la declaración en el programa de:



constante/s

variable/s global/es

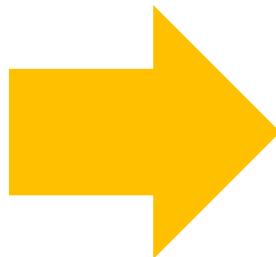
variable/s local al programa/es



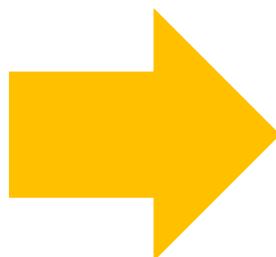
TIEMPO DE EJECUCION: puede calcularse haciendo un análisis empírico o un análisis teórico del programa.



El tiempo de un algoritmo puede definirse como una función de entrada:



Existen algoritmos que el tiempo de ejecución no depende de las características de los datos de entrada sino de la cantidad de datos de entrada o su tamaño.



Existen otros algoritmos el tiempo de ejecución es una función de la entrada “específica”, en estos casos se habla del tiempo de ejecución del “peor” caso. En estos casos, se obtiene una cota superior del tiempo de ejecución para cualquier entrada



Para medir el tiempo de ejecución se puede realizar un análisis empírico o un análisis teórico

ANALISIS EMPIRICO

Requiere la implementación del programa, luego ejecutar el programa en la máquina y medir el tiempo consumido para su ejecución.



Fácil de realizar.



Obtiene valores exactos para una máquina determinada y unos datos determinados.

Completamente dependiente de la máquina donde se ejecuta

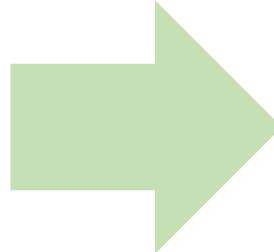
Requiere implementar el algoritmo y ejecutarlo repetidas veces (para luego calcular un promedio).



Para medir el tiempo de ejecución se puede realizar un análisis empírico o un análisis teórico

ANALISIS TEORICO

Implica encontrar una cota máxima (“peor caso”) para expresar el tiempo de nuestro algoritmo, sin necesidad de ejecutarlo.



A partir de un programa correcto, se obtiene el tiempo teórico del algoritmo y luego el orden de ejecución del mismo. Lo que se compara entre algoritmos es el orden de ejecución.

$$T(n) = 4 \text{ UT}$$

$$T(n) = (20 + N) \text{ UT}$$

$$T(n) = (20 + \log N) \text{ UT}$$

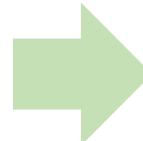
$$T(n) = (N * N) = N^2 \text{ UT}$$

$$O(n) = C \text{ (constante)}$$

$$O(n) = N$$

$$O(n) = \log N$$

$$O(n) = N^2$$





Para medir el tiempo de ejecución se puede realizar un análisis empírico o un análisis teórico

ANALISIS TEORICO

Dado un algoritmo que es correcto se calcula el tiempo de ejecución de cada una de sus instrucciones. **Para eso se va a considerar:**

Sólo las **instrucciones elementales** del algoritmo: asignación, y operaciones aritmético/lógicas.

Una instrucción elemental utiliza un tiempo constante para su ejecución, independientemente del tipo de dato con el que trabaje. **1UT.**



ANALISIS TEORICO

Program uno;

```

var
aux,temp,x: integer;
Begin
(1) aux:= 58;
(2) aux:= aux * 5;
(3) temp:= aux;
(4) read (x);
End.
```

Qué ocurre cuando
hay estructuras de
control?

El tiempo de ejecución de un algoritmo que **NO tiene estructuras de control** está dado por:

$$T(\text{alg}) = T(1) + T(2) + T(3) + T(4)$$

$$T(1) = \text{asignación} = 1\text{UT}$$

$$T(2) = \text{multiplicación} + \text{asignación} = 2\text{UT}$$

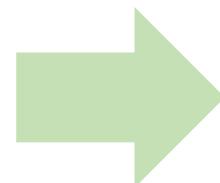
$$T(3) = \text{asignación} = 1\text{UT}$$

$$T(4) = \text{no se considera}$$

$T(\text{alg}) = 4\text{UT}$



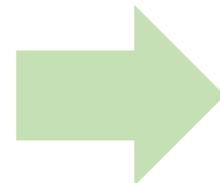
El tiempo de ejecución del **IF**



Tiempo de evaluar la condición + tiempo del cuerpo.

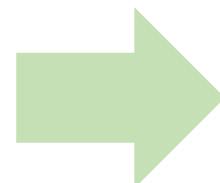
Si hay else, Tiempo de evaluar la condición + max(then,else).

El tiempo de ejecución del **FOR**



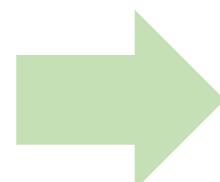
$(3N + 2) + N$ (cuerpo del for).
N representa la cantidad de veces que se ejecuta el for.

El tiempo de ejecución del **WHILE**



C(N+1) + **N**(cuerpo del while).
N representa la cantidad de veces que se ejecuta el while. ($N \geq 0$)
C cantidad de tiempo en evaluar la condición

El tiempo de ejecución del **REPEAT UNTIL**



C(N) + **N**(cuerpo del repeat).
N representa la cantidad de veces que se ejecuta el repeat. ($N > 0$)
C cantidad de tiempo en evaluar la condición



ANALISIS TEORICO

```

Program uno;
var
  aux,temp,x: integer;
Begin
  (1) aux:= 58;           -----> T(1)= asignación = 1UT
  (2) aux:= aux * 5;      -----> T(2)= multiplicación +
                                asignación = 2UT
  (3) if (aux > 45) and (aux <300) then
    begin
      temp:= aux - 5;
      x:= temp + aux + 2;
    end;                      }-----> T(3)= IF= evaluar la condición
                                + cuerpo=
                                = (1) + (1) + (conector) = 3 +
                                resta + asignación
                                suma + suma + asignación
                                = 3UT + 2UT + 3UT = 8UT
  (4) x:= x * 10;          -----> T(4)= multiplicación +
                                asignación = 2UT
end;

```

T (alg) = 13 UT



ANALISIS TEORICO

```

Program uno;
var
aux,temp,x: integer;

Begin
(1) read(aux);
(2) if (aux > 45) then
begin
temp:= aux - 5;
x:= temp;
end
else
aux:= aux + 1 * (aux MOD 2);
end;
  
```

$$T(\text{alg}) = T(1) + T(2)$$

$T(1)$ = el read no se tiene en cuenta.

$T(2)$ = IF= evaluar la condición
+ max(then,else)
cond= 1 +
then= 2 + 1 = 3
else= 4
= 1UT + max(3,4) = 5UT

T (alg) = 5 UT



ANALISIS TEORICO

```

Program uno;
var
  i,temp,x: integer;
Begin
  (1) aux:= 8;
  (2) for i:= 1 to 5 do
    begin
      x:= aux;
      aux:= aux + 5;
    end
  (3) aux:= aux + 1;
end;
  
```

$$T(\text{alg}) = T(1) + T(2) + T(3)$$

$T(1)$ = asignación 1 UT.

$T(2)$ = for= $3N+2 + N(\text{cuerpo})$

$$N= 5$$

$$(3*5) +2 + 5(\text{cuerpo})$$

$$17 + 5(\text{cuerpo})$$

$$\text{cuerpo} = 1 + 2 = 3 \text{UT}$$

$$>> 17 + 5(3) =$$

$$= 17 + 15 = 32 \text{UT}$$

$$T(3)= 1 + 1 = 2 \text{ UT.}$$

T (alg) = 35 UT



ANALISIS TEORICO

```

Program uno;
var
  i,temp,x: integer;
Begin
  (1) aux:= 8;
  (2) for i:= 4 to 9 do
    begin
      x:= aux;
      aux:= aux + 5;
    end
  (3) aux:= aux + 1;
end;
  
```

$$T(\text{alg}) = T(1) + T(2) + T(3)$$

$T(1)$ = asignación 1 UT.

$T(2)$ = for= $3N+2 + N(\text{cuerpo})$

$$N= 6 \text{ (LS-LI + 1)}$$

$$(3*6) + 2 + 6(\text{cuerpo})$$

$$20 + 6(\text{cuerpo})$$

$$\text{cuerpo} = 1 + 2 = 3 \text{ UT}$$

$$>> 20 + 6(3) =$$

$$= 20 + 18 = 38 \text{ UT}$$

$$T(3)= 1 + 1 = 2 \text{ UT.}$$

T (alg) = 41 UT



ANALISIS TEORICO

```

Program uno;
var
  i,temp,x: integer;

Begin
  (1) aux:= 0;
  (2) while (aux < 5) do
    begin
      x:= aux;
      aux:= aux + 1;
    end
  end;
  (3) aux:= aux + 1;
end;
  
```

$$T(\text{alg}) = T(1) + T(2) + T(3)$$

$T(1)$ = asignación 1 UT.

$T(2)$ = while= $C(N+1) + N(\text{cuerpo})$

$$C = 1$$

$$N = 5$$

$$1 * (6) + 5(\text{cuerpo})$$

$$\text{cuerpo} = 1 + 2 = 3 \text{UT}$$

$$>> 6 + 5(3) =$$

$$= 6 + 15 = 21 \text{UT}$$

$T(\text{alg}) = 24 \text{ UT}$

$T(3) = 1 + 1 = 2 \text{ UT.}$



ANALISIS TEORICO

Program uno;

var
i,temp,x: integer;

Begin

(1) read(aux);

(2) while (aux < 5) do

begin

x:= aux;

aux:= aux + 1;

end

(3) aux:= aux + 1;

end;

$T(\text{alg}) = T(1) + T(2) + T(3)$

$T(1) = \text{read}$ no se cuenta 0 UT.

$T(2) = \text{while} = C(N+1) + N(\text{cuerpo})$

$C = 1$

$N = ???$

$1 * (N+1) + N(\text{cuerpo})$

$\text{cuerpo} = 1 + 2 = 3 \text{UT}$

$\gg N+1 + N(3) =$

$= N + 1 + 3N = 4N + 1 \text{ UT}$

$T(\text{alg}) = 0 + 4N + 1 + 2 = (4N + 3) \text{ UT.}$



ANALISIS TEORICO

$$T(\text{alg}) = T(1) + T(2) + T(3)$$

Program uno;

var

i,temp,x: integer;

Begin

(1) aux:=0;

(2) while (aux >= 0) and (aux<5) do

begin

x:= aux;

aux:= aux + 1;

end

(3) aux:= aux + 1;

end;

$$T(1) = 1 \text{ UT.}$$

$$T(2) = \text{while} = C(N+1) + N(\text{cuerpo})$$

$$C = (1) + (1) + \text{conector} = 3$$

$$N = 5$$

$$3*(5+1) + 5(\text{cuerpo})$$

$$\text{cuerpo} = 1 + 2 = 3 \text{ UT}$$

$$>> 18 + 5(3) =$$

$$= N + 18 + 15 = 33 \text{ UT}$$

$$T(3) = 1 + 1 = 2 \text{ UT.}$$

$T(\text{alg}) = 1 + 33 + 2 = 36 \text{ UT}$



ANALISIS TEORICO

```

Program uno;
var
  i,temp,x: integer;
Begin
  (1) aux:=0;
  (2) repeat
    x:= aux;
    aux:= aux + 1;
  until (aux > 5)
  (3) aux:= aux + 1;
end;
  
```

$$T(\text{alg}) = T(1) + T(2) + T(3)$$

$$T(1) = 1 \text{ UT.}$$

$$T(2) = \text{repeat} = C(N) + N(\text{cuerpo})$$

$$C = 1$$

$$N = 5$$

$$1 * (5) + 5(\text{cuerpo})$$

$$\text{cuerpo} = 1 + 2 = 3 \text{ UT}$$

$$>> 5 + 5(3) =$$

$$= N + 5 + 15 = 20 \text{ UT}$$

$$T(3) = 1 + 1 = 2 \text{ UT.}$$

T (alg) = 1 + 20 + 2 = 23 UT