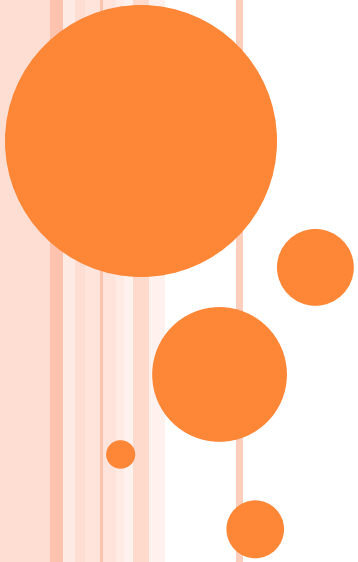
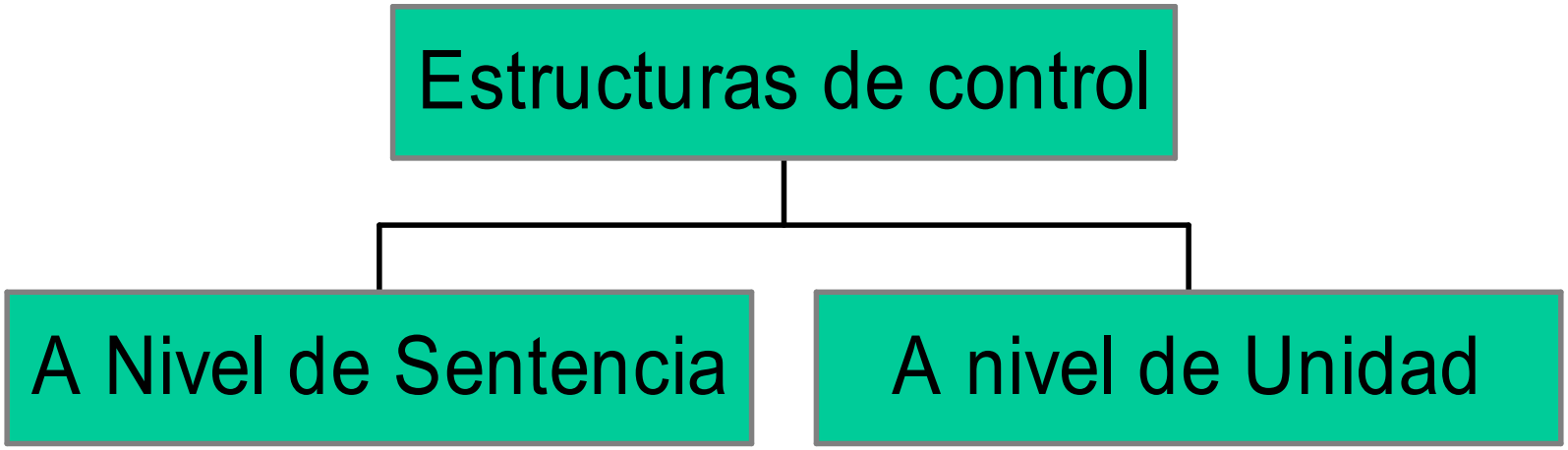


Estructuras de control

CYPLP

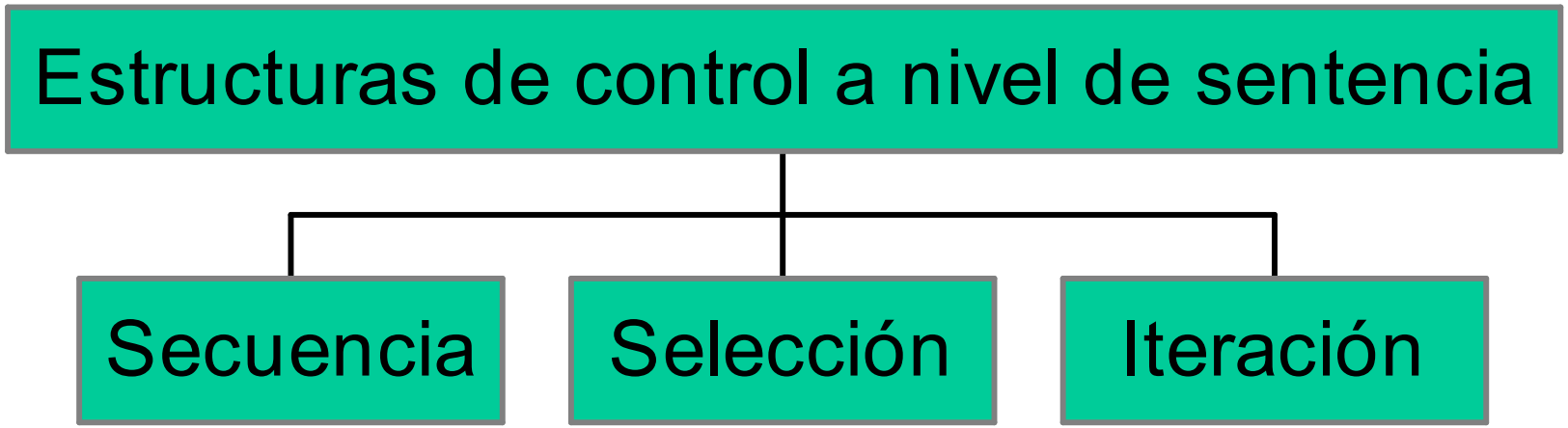
Estructuras de Control

Son el medio por el cual los programadores pueden determinar el flujo de ejecución entre los componentes de un programa



Estructuras de Control

- **A Nivel de Unidad** : Cuando el flujo de control se pasa entre unidades. Intervienen los pasajes de parámetros.
- **A Nivel de Sentencia:** Se dividen en tres grupos



SECUENCIA

- Es el flujo de control más simple.
- Indica la ejecución de una sentencia a continuación de otra.
- El delimitador más general y más usado es el ";"
- Hay lenguajes que no tienen delimitador y establecen que por cada línea vaya una instrucción. Se los llaman orientados a línea.

  Ej: Fortran, Basic, Ruby, Phyton
- Se pueden agrupar varias sentencias en una, llamada *Sentencia Compuesta*, llevan delimitadores como Begin y End. Ej: Ada, { y } en C, C++, Java, etc.

ASIGNACIÓN

- Sentencia que produce cambios en los datos de la memoria.
- Asigna al l-valor de un dato objeto el r-valor de una expresión.
- Sintaxis en diferentes lenguajes

A := B	Ej: Pascal, Ada, etc.
A = B	Ej: Fortran, C, Prolog, Python, Ruby, etc.
MOVE B TO A	COBOL
A← B	APL
(SETQ A B)	LISP

Distinción en entre sentencia y expresión

- En cualquier lenguaje convencional, existe diferencia entre sentencia de asignación y expresión
- En otros lenguajes tales como C definen la sentencia de asignación, como una expresión con efectos laterales.
- Las sentencias de asignación devuelven valores.

Distinción en entre sentencia y expresión

- Evalúa de derecha a izquierda en C
- Ejemplo a=b=c=0;
- if (i=30) printf("Es verdadero")
- La mayoría de los lenguajes de programación requieren que sobre el lado izquierdo de la asignación aparezca un l-valor. C permite cualquier expresión que denote un l-valor.

  Ej.: ++ p = *q;
  (i<j?z:y)=4;

SELECCIÓN

- Esta estructura de control permite que el programador pueda expresar una elección entre un cierto número posible de sentencias alternativa

- Evolución:

→ **If lógico de Fortran**

If (condición lógica) sentencia

Si la condición es verdadera ejecuta la sentencia

SELECCIÓN

if then else de Algol

if (condición lógica) **then** sentencia1

else sentencia2

Este permite tomar dos caminos posible

Problemas:

Ambigüedad, no establecía por lenguaje cómo se asociaban los else con los If abiertos.

→ **if then else de Pl/1, Pascal y C sin ambigüedad**

Establecen por lenguaje que **cada else cierra con el último if abierto**.

Solución: Sentencia de cierre del bloque condicional, por ejemplo end if, fi, etc.

Desventajas:

Ilegibilidad, programas con muchos **if** anidados pueden ser ilegibles. Utilizar **Begin End.**

SELECCIÓN – EJEMPLO EN C

No lleva la palabra clave **"then"**

```
int main ()
    {
    int year = 0;
    printf (" Introduzca el año : ");
    scanf (" %d", & year );
    if ((0 == year % 400) || ((0 == year % 4) && (0 != year % 100)))
        printf ("El anio %d es bisiesto \n", year );
    else
        printf ("El anio %d NO es bisiesto \n", year );
    return 0;
    }
```

Si son más de una sentencia se deben encerrar entre { }

```
(i<j?z:y)
```

Otra forma de selección

SELECCIÓN

If then else de Python, sin Ambigüedad y legible

```
if sexo == 'M':
    print 'La persona es Mujer'
else :
    print 'La persona es Varón'
```

```
if hora <= 6 and hora >=12:
    print 'Buenos días!!'
elif hora >12 and hora < 20 :
    print 'Buenas tardes!!'
else :
    print 'Buenas noches!!'
```

- : es obligatorio al final del if, else y del elif
- La indentación es obligatoria al colocar las sentencias correspondientes tanto al if, else y el elif

Sentencia condicional – A if C else B de Python

- Construcción equivalente al "?" del lenguaje C
- Devuelve A si se cumple la condición C, sinó devuelve B

Ejemplo de uso:

```
>>> altura = 1.79
>>> estatura = "Alto" if altura > 1.65 else "Bajo"
>>> estatura
'Alto'
>>>
```

IMPORTANTE: Python para evaluar las condiciones utiliza la "**Evaluación con circuito corto**"

Selección Múltiple

Sentencia Select de Pl/1

Pl/1 incorpora la sentencia de selección entre dos o más opciones

```
Select
    when(A) sentencia1;
    when(B) sentencia2
    ..........
    Otherwise sentencia n;
End;
```

Reemplaza a :

```
if (A) then sentencia1
else if (B) then sentencia2
        else if .....
            else sentencia n;
```

Selección Múltiple - Pascal

- Incorpora que los valores de la expresión sean Ordinales y ramas con etiquetas. No importan el orden en que aparecen las ramas. Tiene la opción "else".
- Es inseguro porque no establece qué sucede cuando un valor no cae dentro de las alternativas puestas

```
var opcion : char;
    begin
      readln(opcion);
  case opcion of
            '1' : nuevaEntrada;
            '2' : cambiarDatos;
            '3' : borrarEntrada
          else writeln('Opcion no valida!!')
    end
  end
```

El else es opcional, que hace si lse ingresa un 5 y no hay un else?

Selección Múltiple - ADA

- Las expresiones pueden ser solamente de tipo entero o enumerativas

- En las selecciones se debe estipular todos los valores posibles que puede tomar la expresión

- Tiene la cláusula Others que se puede utilizar para representar a aquellos valores que no se especificaron explícitamente

Si NO se coloca la rama para un posible valor o si NO aparece la opción Others en esos casos, no pasará la compilación.

Selección Múltiple - ADA

Ejemplo 1

```
Case Operador is
when  ´+´ => result:= a + b;
when  ´-´ => result:= a - b;
when  others => result:= a * b;
end case
```

La cláusula **others** se debe colocar porque las etiquetas de las ramas NO abarcan todos los posibles valores de Operador

Ejemplo 2

```
Case Hoy is
 when MIE..VIE => Entrenar_duro; -- Rango.
 when MAR | SAB => Entrenar_poco; -- Varias elecciones.
when DOM => Competir; -- Una elección.
when others => Descansar; -- Debe ser única y la última alternativa.
end case;
```

Selección Múltiple – C, C++

- Constructor Switch
- Cada rama es etiquetada por uno o más valores constantes
- Cuando la opción coincide con una etiqueta del Switch se ejecutan las sentencias asociadas y se continúa con las sentencias de las otras entradas.
- Existe la sentencia break, que provoca la salida
- Tiene una cláusula default que sirve para los casos que el valor no coincida con ninguna de las opciones establecidas

SELECCIÓN MÚLTIPLE – C, C++

```
Switch Operador {
    case  ´+´ :
        result:= a + b; break;
    case  ´-´:
        result:= a - b;  break;
    default :
        result:= a * b;
    }
```

De be ponerse la sentencia **break** para saltar las siguientes ramas

Selección Múltiple – Ruby

```ruby
raza = 'enano'
# Ahora utilizaremos el case
puts 'Utilizando case asignado a una varible :'
personaje = case raza
            when 'elfo' then 'Legolas'
            when 'enano' then 'Gimli'
            when 'mago' then 'Gandalf'
            when 'ents' then 'Barbol'
            when 'humano' then 'Aragorn'
            when 'orco' then 'Ufthak'
end
puts personaje
```

Si no entra en ninguna opción, sigue la ejecución y la variable no tendrá ningún valor!

ITERACIÓN

Este tipo de instrucciones se utilizan para representar aquellas acciones que se repiten un cierto número de veces

Su evolución:

→**Sentencia Do de Fortran**

Do label var-de-control= valor1, valor2

.....

Label **continue**

- La variable de control solo puede tomar valores enteros
- El Fortran original evaluaba si la variable de control había llegado al límite al final del bucle, o sea que siempre una vez lo ejecutaba

ITERACIÓN

Sentencia For de Pascal, ADA, C , C++

- La variable de control puede tomar cualquier valor ordinal, no solo enteros
- Pascal estandar no permite que se toquen ni los valores del límite inferior y superior, ni el valor de la variable de control.
- La variable de control puede ser de cualquier valor ordinal.
- El valor de la variable fuera del bloque se asume indefinida

Ada,
no debe declararse el iterador

```
for i in 1..N loop
V(i) := 0;
end loop
```

Puede ser **reverse**

ITERACIÓN

C, C++ se compone de tres partes: una inicialización y dos expresiones

- La primer expresión (2do. Parámetro) es el testeo que se realiza ANTES de cada iteración. Si no se coloca el for queda en LOOP.
- En el primer y último parámetro se pueden colocar sentencias separadas por comas

```
for (p= 0, i=1 ; i<=n ; i++)
{
p+= a * b;
b = p * 8;
}
```

En C++ pueden haber declaraciones. Ej: int p=0,

ITERACIÓN

Phyton, estructura que permite iterar sobre una secuencia. Las secuencias pueden ser: una lista, una tupla, etc.

```
lista = [ "el", "for", "recorre", "toda", "la", "lista"]
for variable in lista:
    print variable
```

Imprimirá:

el
for
recorre
toda
la
lista

Si se quiere que el for actúe como en los demás lenguajes, entonces hacer uso de la función "range()", ej. range(6). Hará que variable tome los valores de 0 a 5.

ITERACIÓN

Phyton, uso de la funcion "**range()**"

- La función devuelve una lista de números enteros ej. range(5), devuelve [0,1,2,3,4].
- Puede tener hasta 3 argumentos:
  - 2 argumentos: range(2,5), devuelve [2,3,4]
  - 3 argumentos: range(2,5,2), devuelve [2,4]

Esa función **range()** da la posibilidad de simular la sentencia FOR de otros lenguajes

for i **in range (**valor- incial, valor-final + 1**):** acciones

ITERACIÓN

Phyton, ejemplo uso de la funcion "**range()**"

```
for vuelta in range(1,10):
    print("Vuelta "+str(vuelta))
```

Salida

Vuelta 1
Vuelta 2
Vuelta 3
....
Vuelta 10

ITERACIÓN: WHILE

while

- Estructura que permite repetir un proceso mientras se cumpla una condición.
- La condición se evalúa antes de que se entre al proceso

- En **Pascal:**

```
while condición do sentencia;
```

- En **C, C++:**

```
while (condición) sentencia;
```

- En **ADA**:

```
while condición sentencia
end loop;
```

- En **Phyton**:

```
while condición :
        sentencia1
        sentencia2
        .....
        sentencia n
```

ITERACIÓN: UNTIL

Until

- Estructura que permite repetir un proceso mientras se cumpla una condición.
- La condición se evalúa al final del proceso, por lo que por lo menos una vez el proceso se realiza

- En Pascal:

```
repeat
sentencia
until condición;
```

- En C, C++:

```
do sentencia;
 while (condición);
```

ITERACIÓN: LOOP

Ada tiene una estructura iterativa

```
loop
    ...
end loop;
```

De este bucle se sale normalmente, mediante una sentencia "**exit when**" o con una alternativa que contenga una cláusula "**exit**".

```
loop
    ...
    exit when condición;
    ...
end loop;
```