



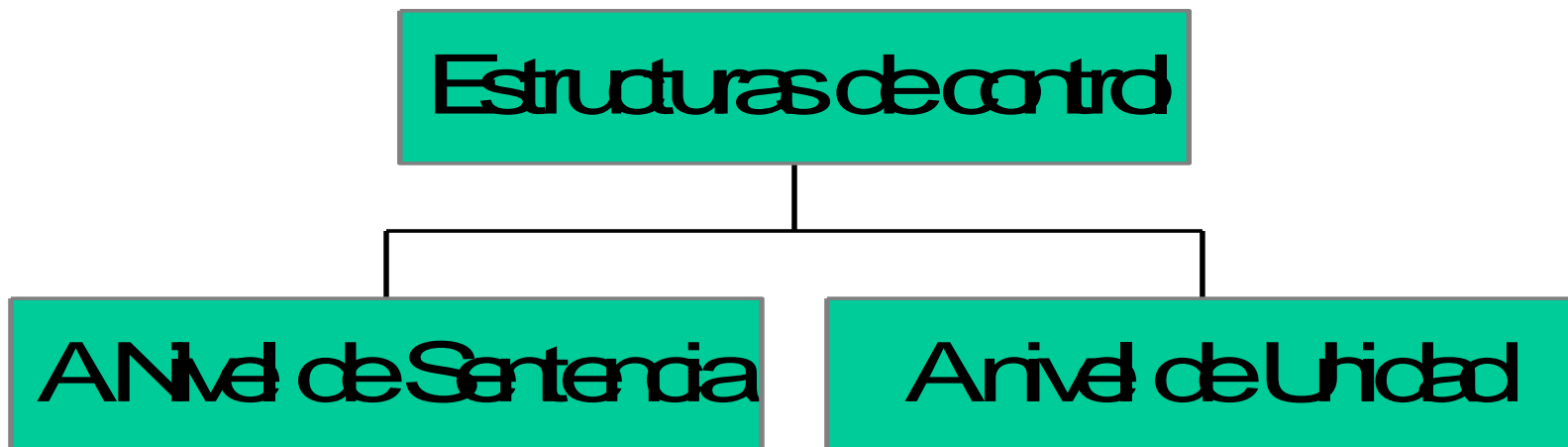
ESTRUCTURAS DE CONTROL

CYPLP

ESTRUCTURAS DE CONTROL

Los lenguajes de programación permiten estructurar al código en relación al flujo de control entre los diferentes componentes de un programa a través de 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** (rutinas, funciones, proc. Etc.)

Interviene:

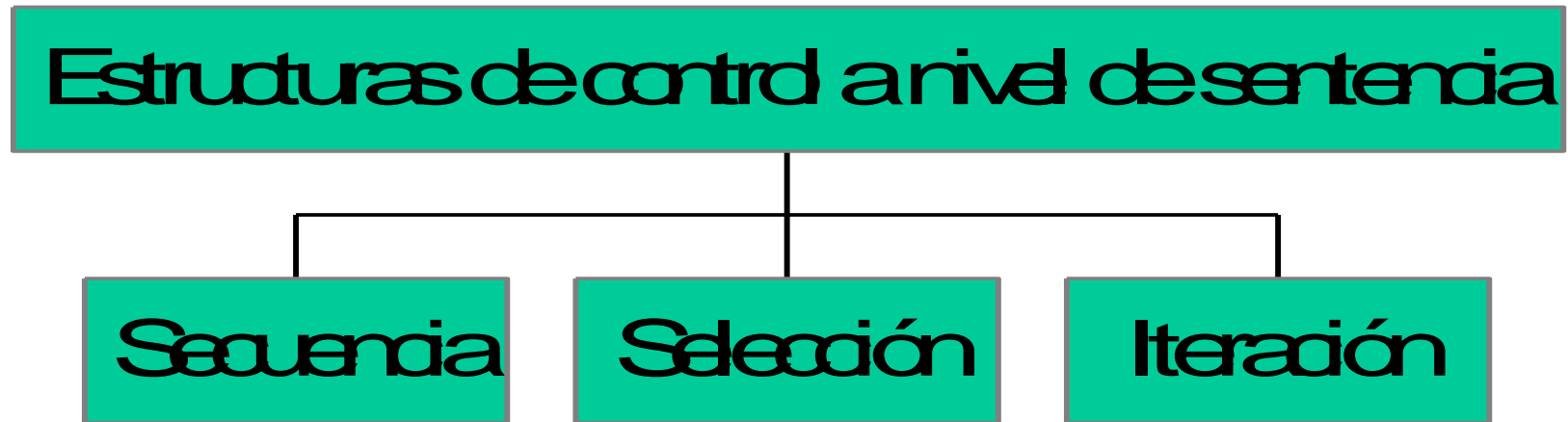
- **Pasajes de parámetros.**
- **call-return**
- **Excepciones**
- **otros**

- *¡Se verá en otras clases!*

ESTRUCTURAS DE CONTROL

○ A Nivel de Sentencia:

- Se dividen en tres grupos



SECUENCIA

- Es el **flujo** de control más simple.
- Ejecución de **una sentencia a continuación de otra**.
- El **delimitador** más general y más usado es el “;”

○ *Sentencia 1;*

○

○ *Sentencia n;*

○ Hay lenguajes que **NO** tienen delimitador

○ establecen que **por cada línea** haya sólo **1 instrucción**. Se los llaman **orientados a línea**.

○ Ej: Fortran, Basic, Ruby, Python

SECUENCIA

- Otros lenguajes permiten *Sentencias Compuestas*
- Se pueden **agrupar varias sentencias** en una con el uso **delimitadores**:
 - **Begin y End.** En Ada, Pascal
 - **{ }** en C, C++, Java, etc.

Ej. compuestas en C

```
{  
pi=3.14;  
c=x+y;  
++i;  
}
```

Ej. compuestas en Pascal:

```
Begin  
  Readln (Numero);  
  Numero:= Numero+1;  
  Write (`El número es `,Numero)  
End.
```

SENTENCIA - ASIGNACIÓN

- **Asignación:** Sentencia que **produce cambios** en los **datos de la memoria**. **($x = a + b$)**

*Asigna al l-valor de un objeto de dato (x)
el r-valor de una expresión. ($a + b$)*

- 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 ENTRE SENTENCIA DE ASIGNACIÓN Y EXPRESIÓN

- En cualquier **lenguaje convencional**, existe **diferencia** entre **sentencia de asignación** y **expresión**

- Ejemplos:

$(x + 3)^*2$ *// expresión*

$a > b \ \&\& \ c < d$ *// expresión*

$y = (x + 3)^*2;$ *// sentencia*

DISTINCIÓN ENTRE SENTENCIA DE ASIGNACIÓN Y EXPRESIÓN

- Las sentencias de asignación *devuelven* el **valor** de la expresión y *modifican* la posición de **memoria**.
- En otros lenguajes tales como C definen la **sentencia de asignación** como una **expresión** con efectos colaterales.

EJEMPLO EFECTOS COLATERALES EN C

```
1#include <stdio.h>
2
3int main()
4{
5    //ejemplo asignación múltiple
6    /*int a,b,c;
7    a=b=c=0;
8    printf("a es igual a %d, b es igual a %d, c es igual a %d \n",a,b,c );
9    */
10
11    //ejemplo de asignación que retorna un valor
12    /*int i=0;
13    if (i=30){
14    printf("Es verdadero porque i toma el valor 30 y al ser mayor a 0 se considera true. \n Valor de i es %d", i);
15    }*/
16
```

el 0 se devuelve como r-valor a todos, primero se asigna a C, luego C asigna a B, luego B asigna a A y **todos valen 0**

ejemplo de asignación que retorna un valor.
no es comparación (C usa == para comparar).
Interpreta I es distinto de 0, porque le asignamos valor 30 y al ser mayor que 0 **da verdadero**,
imprime 30

Continúa el código.....

EJEMPLO EFECTOS COLATERALES EN C

```
17 //ejemplo de lvalue en el lado izquierdo
18 /*int p=0; //Declaración de variable entera de tipo entero
19 int *puntero; //Declaración de variable puntero de tipo entero
20 puntero = &p; //Asignación de la dirección memoria de p
21
22 printf("El valor de p es: %d. \nEl valor de *puntero es: %d. \n",p,*puntero);
23 printf("La dirección de memoria de *puntero es: %p. \n",puntero);
24 ++p=*puntero;
25 printf("El valor de p es: %d. \nEl valor de *puntero es: %d. \n",p,*puntero);*/
26 //falla porque del lado izquierdo debe haber un lvalue
27
28 //otro ejemplo
29 /*int i,j,z,y;
30 i=0; j=2;
31 (i<j?z:y)=4;
32 */
```

error en línea24 ++p=*puntero;
requiere del lado izq de la asignación un **l-valor** pero lo que devuelve es un r-valor.
++p es una expresión,

Para ejecutar
sacar
comentarios
/*

error: se requiere lvalue como operando izquierdo de la asignación

? operador condicional (relacionado con la estructura if/else.)

Se escribe en la forma: **a ? b : c;**

a es una *expresión booleana*, **b** y **c** pueden ser *expresiones o sentencias*.

Si el valor de **a** es verdadero se devuelve el valor de **b**, Si el valor de **a** es falso se devuelve el valor de **c**.

Ej. Si **I** es menor que **J** devuelve el **valor de Z** caso contrario devuelve el **valor de Y** (el r-valor y no el l-valor)

DISTINCIÓN ENTRE SENTENCIA DE ASIGNACIÓN Y EXPRESIÓN

Resumen del código anterior:

- C evalúa de **derecha a izquierda** Ejemplo `a=b=c=0;`
- `if (i=30) printf("Es verdadero")` está permitido

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

Vimos en el ejemplo anterior que no

SELECCIÓN - IF

- IF Estructura de control que permite expresar una **elección entre un cierto número posible de sentencias alternativas** (*ejecución condicional*)
- Evolución:
 - If lógico de Fortran

If (condición lógica) sentencia

Si condición es verdadera ejecuta la sentencia

SELECCIÓN - IF EN ALGOL

if then else de **ALGOL**

if (condición lógica) **then** sentencia1
else sentencia2

Esto permite tomar 2 caminos posibles

SELECCIÓN - IF EN ALGOL

if then else anidados en Algol

- Con múltiples caminos posibles

Problema de Ambigüedad:

El lenguaje **no establecía cómo se asociaban**
los **else** con los **if** abiertos!

if $x > 0$ then if $x < 10$ then $x := 0$ else $x := 1000$

Si $x = 0$ lo pongo en 1000 o queda en 0?

Si $x = 10$ lo pongo en 1000 o queda en 10?

Sin Ambigüedad:

if then else de PL/1, Pascal y C

Para evitar ambigüedades, la **REGLA** es que **cada** rama **else** se empareje con la instrucción **if** solitaria **más próxima** (buscando siempre hacia atrás).

cada else se empareja para cerrar al último if abierto

Elimina la ambigüedad, pero las **declaraciones anidadas** pueden ser **difíciles de leer**, especialmente si el programa está escrito **sin respetar sangría**

SELECCIÓN - IF

Otro tipo de Solución para dar claridad:

- Usar **sentencia de cierre** del bloque condicional if

por ejemplo

- **fi** en Algol 68
- **end if** en Ada
- **end** Modula-2
- Etc.

Ej. En Modula-2

```
if i = 0
  then i := j
  else i := i + 1;
      j := j - 1
end
```

SELECCIÓN - IF

Desventajas:

- **Ilegibilidad**, programas con muchos **if anidados** pueden ser **ilegibles**.

Solución:

- En algunos lenguajes se utiliza una **instrucción** compuesta **begin** y **end**. Es un **bloque de instrucciones** acotadas entre las palabras **begin** y **end**. Depende de cada lenguaje.

```
If expr1 then  
  
    Begin  
  
        If expr2 then  
            a:=b+c  
        Else  
            a:=b-c  
  
    End;
```

SELECCIÓN - IF EN C

if then else en C en línea y en bloque

```
If (condición) Instrucción 1;  
else Instrucción A;
```

- Usa ()
- No usa then

```
if (condición) {  
    Instrucción 1;  
    Instrucción 2;  
    -  
    -  
    Instrucción n;  
}  
else {  
    Instrucción A;  
    Instrucción B;  
    -  
    -  
    Instrucción Z;  
}
```

- Usa { }
- No usa begin y end

Se recomienda **usar siempre las llaves** porque genera un **código más legible** y más **fácil de mantener**, quedando bien delimitada la intención del programador.

SELECCIÓN – EJEMPLO EN C

- **C no lleva la palabra clave “then” como en Pascal**
- **Pascal no usa () en la condición**

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

Diferencias de implementación entre los distintos lenguajes y entre las diferentes versiones

SELECCIÓN – SENTENCIA CONDICIONAL CORTA

if corto en C

Se puede representar:

cond ? b : c;

? *operador condicional*

cond es una *expresión booleana*

b y **c** pueden ser *expresiones o sentencias*.

- Si el valor de **cond** es verdadero se **devuelve el valor de b**
- Si el valor de **cond** es falso se **devuelve el valor de c**

(i<j?z:y)

Pasar de if largo a if corto

```
...  
country = ''  
if(lang == 'es') {  
    country = 'ES'  
} else {  
    country = 'EN'  
}  
...
```

A algo como esto,

```
...  
country = (lang == 'es' ? 'ES' : 'EN')  
...
```

SELECCIÓN - IF EN PYTHON

If condicion **then else** de **Python**

- **sin ambigüedad y legible** (incorpora elif y sangría)

```
if sexo == 'M':  
    print 'La persona es Mujer'  
else:  
    print 'La persona es de otro género'
```

```
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**, **elif** y del **else**
- El uso de **()** en la **condición** es **opcional**

SELECCIÓN - SENTENCIA CONDICIONAL CORTA EN PYTHON

[on_true] **if** [expresión] **else** [on_false] de **Python**

Expresión condicional (llamado "operador ternario")

Construcción equivalente al “?” del lenguaje C

C ? A : B

A if C else B

Devuelve A si se cumple la condición C, sinó devuelve B

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

CIRCUITO CORTO

Se trata de una forma de evaluar expresiones lógicas. Se implementa en muchos lenguajes de programación.

La conjunción ("y" / "and") da como resultado verdadero únicamente cuando ambos términos son verdaderos. Si el primer término es falso, no es necesario evaluar el segundo: el resultado será falso.

La disyunción ("o" / "or") da como resultado falso únicamente cuando ambos términos son falsos. Si el primer término es verdadero, no es necesario evaluar el segundo: el resultado será verdadero.

¿Qué sucede si tenemos una condición como `if (A and B)`, donde B resulta ser un objeto nulo?

Tener en cuenta el circuito corto permite evitar estos errores.

Esto permite evitar errores al evaluar expresiones indefinidas: si existe una condición **A and B** en la que B tiene un valor nulo, podría darse un error al intentar evaluarla. Entonces podemos hacer que la expresión A garantice que B no sea nulo. También se utiliza cuando es computacionalmente costoso realizar cada evaluación.

EJEMPLOS CIRCUITO CORTO Y CIRCUITO LARGO

a and b and false and c and d

El circuito corto termina cuando evalúa y da el ***primer falso***, entonces ***no evalúa más ni C ni D*** ya que la condición dio falsa

a and false and f()

El circuito corto termina cuando evalúa y da el ***primer falso***, en esta condición ***nunca llama a f()***



SELECCIÓN MÚLTIPLE

Los lenguajes incorporan distintos tipos de sentencias de selección *para poder elegir entre dos o más opciones posibles*

Para reemplazar a estructuras del tipo:

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

SELECCIÓN MÚLTIPLE - PASCAL

- La **expresión** a evaluar es llamada **“selector”**
- lleva **palabra** **case** seguida de **variable de tipo ordinal** y la **palabra reservada of**.
- Luego **lista de sentencias** de acuerdo a diferentes valores que puede adoptar la variable (los “casos”). **Llevan etiquetas**.
- No importan el orden en que aparecen
- bloque **else** para el caso que la variable adopte un valor que no coincida con ninguna de las sentencias de la lista. **(opcional)**
- Es inseguro porque no establece qué sucede cuando un valor no cae dentro de las alternativas puestas.
- Para finalizarla la estructura del case se coloca un **“end;”** (no se corresponde con ningún “begin”).

El formato es el siguiente:

```
case variable_ordinal of
    valor1: sentencia 1;
    valor2: sentencia2;
    valor3: sentencia3;
else
    sentencia4;
end;
```

Ordinal: puede obtenerse un predecesor y un sucesor (a excepción del primer y el último (expresa la idea de orden o sucesión))

SELECCIÓN MÚLTIPLE - PASCAL

el else cambió
entre versiones de
Pascal
ejemplo otherwise

Doble end por
el begin

El else es
opcional. Qué
sucede si se
ingresa un 5 y
no hay un
else?
Es inseguro

Ejemplo:

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

SELECCIÓN MÚLTIPLE - ADA

- Constructor **Case** expresión **is**, con **when** y **end case**
- Las expresiones pueden ser solamente de tipo **entero o enumerativas**
- En las selecciones del **case** se **deben estipular "todos" los valores posibles** que puede tomar la expresión
- El **when** se acompaña con **=>** para **indicar la acción a ejecutar si se cumple la condición.**
- Tiene la cláusula **Others** que se puede utilizar para **representar a aquellos valores que no se especificaron explícitamente**

No pasa la compilación si:

- **NO** se coloca la rama para un posible valor o
- **NO** aparece la opción **Others** en esos casos

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

Debe ser la última

Ejemplo 2

case Hoy **is**

when MIE..VIE => Entrenar_duro; -- *Se puede especificar Rango con ..*

when MAR | SAB => Entrenar_poco; -- *Se puede especificar varias elecciones |*

when DOM => Competir; -- *Única elección.*

when others => Descansar; -- *Debe ser única y la última alternativa.*

end case;

SELECCIÓN MÚLTIPLE – C, C++

- Constructor **Switch** seguido de **expresión**
- Cada rama (**Case**) es etiquetada por uno o más valores constantes (enteros o char)
- Si coincide con una etiqueta del **Switch** se ejecutan las sentencias asociadas y se continúa con las sentencias de las otras entradas. (salvo exista un **break**)
- Existe la sentencia **break**, que provoca la salida de cada rama, sino continúa!!
- Existe **opción default** que sirve para los casos que el valor no coincida con ninguna de las opciones establecidas, es **opcional**

SELECCIÓN MÚLTIPLE – C, C++

Switch (Operador) {

case ' + ' :

result:= a + b; **break**;

case ' - ' :

result:= a - b; **break**;

default : *//Opcional*

result:= a * b;

}

Debe ponerse la sentencia **break** para saltar las siguientes ramas, si no pasa por todas

switch(i)

{

case -1:

n++;

break;

case 0 :

z++;

break;

case 1 :

p++;

break;

}

SELECCIÓN MÚLTIPLE - RUBY

- Constructor **Case expression**, seguido de **when** y **end**
- La **expresión** es **cualquier valor** (cualquier **cadena**, **numérico** o **expresión** que **proporcione** algunos **resultados** como **("a", 1 == 1, etc.)**.
- Dentro de los bloques **when** seguirá buscando la expresión, hasta que **coincida** con la **condición**, **ingresará en ese bloque de código**.
- Si **no coincide** con **ninguna expresión** irá al bloque **else** **igual que por defecto**.
- **else** es **opcional**, esto puede traer **efectos colaterales**

Consejo de programación defensiva:

La cláusula debe **tomar la acción apropiada** o **contener un comentario adecuado** sobre **por qué no se toma ninguna acción**.

SELECCIÓN MÚLTIPLE – RUBY


Noncompliant Code Example

```
case param
  when 1
    do_something()
  when 2
    do_something_else()
end
```

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

Compliant Solution

```
case param
  when 1
    do_something()
  when 2
    do_something_else()
  else
    handle_error('error_message')
end
```



SELECCIÓN MÚLTIPLE – RUBY

```
raza = 'enano'
# Ahora utilizaremos el case
puts 'Utilizando case asignado a una variable :'  
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!**

SELECCIÓN MÚLTIPLE - PL/I

- Sentencia **SELECT** de PL/I incorpora el uso de **WHEN/OTHERWISE/END**
- Tiene 2 tipos de formatos

```
-----Formato - 1-----  
SELECT (identifier) ;  
    WHEN (value1) statement;  
    WHEN (value2) statement;  
    OTHERWISE statement ;  
END;
```

```
-----Formato - 2 -----  
SELECT ;  
    WHEN (cond1) statement;  
    WHEN (cond2) statement;  
    OTHERWISE statement ;  
END;
```

ITERACIÓN

- La **iteración** permite que una **serie de acciones se ejecuten repetidamente (loop)**.
- La mayoría de los lenguajes de programación proporcionan **diferentes tipos de construcciones de bucle** para definir la *iteración de acciones (llamado el cuerpo del bucle)*.
- Comúnmente agrupados como :
 - **Tipo bucle for:** bucles en los que *se conoce el número de repeticiones al inicio del bucle*. (se repiten un cierto número de veces)
 - **Tipo bucle while:** bucles en los que el cuerpo se *ejecuta repetidamente siempre que se cumpla una condición*. (hay 2 tipos)

ITERACIÓN - DEL TIPO FOR LOOP

□ Sentencia **Do** de **Fortran**

Do **label** **var-de-control**=valorIni, valorFin

.....

label **continue**

- La **var-de-control** solo puede tomar **valores enteros** y se **incrementa en 1** (*si no se especifica otra cosa*)
- **ValorIni** es el valor inicial
- **ValorFin** es el valor final
- La **declaración** **continue** junto con la etiqueta **label** se usa como la **última declaración de un DO** (*depende versión de FORTRAN*)

ITERACIÓN - TIPO FOR LOOP

□ Sentencia **Do** de **Fortran**

- Fortran original evaluaba si la variable de control había llegado al límite al final del bucle, o sea que *"siempre una vez lo ejecutaba" y traía problemas.*
- Desde **FORTTRAN 77** se evalúa el número de veces que se debe ejecutar el ciclo **antes de que se ingrese al ciclo** por primera vez.
- La **variable de control** *"nunca" deberá ser modificada por otras sentencias dentro del ciclo*, ya que puede generar errores de lógica.

Ejemplo:

```
DO 1 I = 1,10  
  SUM = SUM A(I)  
1  CONTINUE
```

```
DO 1 I = 10,1  
  SUM = SUM A(I)  
1  CONTINUE
```

El 2do no ejecuta

ITERACIÓN - TIPO FOR LOOP

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

for loop_ctr_var := lower_bound **to** upper_bound **do** statement

- La **variable de control** puede tomar **cualquier valor ordinal**, no solo enteros
- **Pascal estándar "no permite"** que se **modifiquen** los **valores del límite inferior, límite superior**, ni del valor de la **variable de control**.
- El **valor de la variable fuera del bloque** se **asume indefinida**

EJEMPLO EN PASCAL - MODIFICACIÓN DE LA VARIABLE DE CONTROL

```
1
2 Program HelloWorld(output);
3 Uses sysutils;
4 var i: Integer;
5 Procedure VerI();
6 Begin
7   writeln(Concat('valor de i es ',IntToStr(i)));
8   // es inseguro si tocamos el iterador en otra unidad.
9   //i:=i+20;
10 end;
11 BEGIN
12   writeln('Hello, world!');
13   for i:=1 to 20 do begin
14     writeln(Concat('valor de i es ',IntToStr(i)));
15     IF (i=5) THEN begin
16       VerI();
17     end
18     ELSE begin
19       //no se permite alterar el iterador en la misma unidad.
20       //i:=i+1;
21     end;
22   end
23 END.
```

Salida del programa

Free Pascal Compiler version 3.2.0

valor de i es 1.....20

si **descomento $i=i+1$** da este **error**

Compiling main.p, main.p(22,8)

Error: Illegal assignment to for-loop variable "i" main.p(25,4)

Fatal: There were 1 errors compiling module, stopping

Fatal: Compilation aborted

Error: /usr/bin/ppcx64 returned an error exitcode **(no permite modificar la variable en el loop)**

si **descomento $i=1+20$** da

Hello, world!

valor de i es 1 2 3 4 5 5

Puedo modificar fuera la variable hay efecto colateral

ITERACIÓN - TIPO FOR LOOP

Sentencia **For** de **ADA**

- Ada **encierra todo** proceso iterativo **entre las cláusulas **loop** y **end loop****.
- Permite el uso de la sentencia **Exit** para **salir del **loop****.
- La **variable de control** (*iterador*) **NO necesita declararse** (*se declara implícitamente al entrar al bucle y desaparece al salir*)
- El **in** indica incremento, permite decrementar con **in reverse**

```
for i in 1..N loop
```

```
V(i) := 0;
```

```
end loop
```

Puede ser
in reverse

ITERACIÓN - TIPO FOR LOOP

Sentencia **For** de **C, C++**

- Se componen de **3 partes**: 1 **inicialización** y 2 **expresiones**
- **Inicialización**: da el **estado inicial** para la ejecución del bucle.
- **1ra expresión**: especifica el **test** que es **realizado antes de cada iteración**. Sale del ciclo si la expresión no se cumple (sale del rango)
- **2da expresión**: especifica el **incremento** que se realiza **después** de cada iteración.

The diagram illustrates the components of a C/C++ for loop. It shows the code: `for(int x = 0; x < 100; x++) { println(x); // prints 0 to 99 }`. Annotations include: 'parenthesis' pointing to the opening curly brace; 'declare variable (optional)' pointing to 'int x'; 'initialize' pointing to '= 0'; 'test' pointing to '< 100'; 'increment or decrement' pointing to 'x++'; and a large curved arrow indicating the loop's execution flow from the initialization back to the test condition.

```
for(int x = 0; x < 100; x++) {  
    println(x); // prints 0 to 99  
}
```

ITERACIÓN - TIPO FOR LOOP

Sentencia **For** de **C++**

- En **C++** se puede realizar la **declaración** de una **variable** dentro del **for** (.....)
- El **alcance** de dicha variable se extiende **hasta el final del bloque** que encierra la instrucción **for {....}**
- Si se **omiten** una o ambas **expresiones** en un bucle for se puede **crear un bucle sin fin**, del que solo se puede salir con una instrucción ***break, goto o return.***

```
for ( ; ; ) { . . . }
```

ITERACIÓN - TIPO FOR LOOP

Sentencia **For** de **Python**

Python tiene 2 tipos de estructuras

- **For** para iterar sobre una secuencia tipo
 - *lista, tupla, conjunto, diccionario, etc.*
- **For** para iterar sobre un rango de valores basado en una secuencia numérica usando función **Range()**

ITERACIÓN - TIPO FOR LOOP

Sentencia **For** de **Python** para iterar en una secuencia

- El bucle **for** se utiliza para recorrer los elementos de un objeto iterable (*lista, tupla, conjunto, diccionario, ...*) y ejecutar un bloque de código.
- En cada paso de la iteración se tiene en cuenta a un único elemento del objeto iterable, sobre el cuál se pueden aplicar una serie de operaciones.

Su sintaxis es la siguiente:
for <elem> **in** <iterable>:
 <código>

- **elem** variable que toma el valor del elemento dentro/**in** de iterador **iterable** en cada paso del bucle.
- Finaliza su ejecución cuando se recorren todos los elementos.

ITERACIÓN - TIPO FOR LOOP

Sentencia **For** de **Python** para iterear en una secuencia

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

```
Imprimirá  
:  
  el  
  for  
  recorre  
  toda  
  la  
  lista
```

ITERACIÓN - TIPO FOR LOOP

Sentencia **For** de **Python** para iterear sobre un **rango de valores**

- Uso de la *funcion/clase* **range (max)**
- que **devuelve valores** van desde **0** hasta **max - 1**.
- Así el for actúa como en los demás lenguajes

1.	<code>for i in range(11):</code>
2.	<code> print(i)</code>
3.	
4.	0
5.	1
6.	2
7.	3
8.	...
9.	10

ITERACIÓN - TIPO FOR LOOP

Sentencia **For** de **Python** para iterear sobre un rango de valores

El tipo de datos `range` se puede invocar con uno, dos e incluso tres parámetros:

- `range(max)`: Un iterable de números enteros consecutivos que empieza en `0` y acaba en `max - 1`
- `range(min, max)`: Un iterable de números enteros consecutivos que empieza en `min` y acaba en `max - 1`
- `range(min, max, step)`: Un iterable de números enteros consecutivos que empieza en `min` acaba en `max - 1` y los valores se van incrementando de `step` en `step`. Este último caso simula el bucle for con variable de control.

Por ejemplo, para mostrar por pantalla los números pares del 0 al 10 podríamos usar la función `range` del siguiente modo:

```
1. for num in range(0, 11, 2):
2.     print(num)
3.
4. 0
5. 2
6. 4
7. 6
8. 8
9. 10
```

- 1 argumentos: `range(5)` devuelve `[0,1,2,3,4]`
- 2 argumentos: `range(2,5)` devuelve `[2,3,4]`
- 3 argumentos: `range(2,5,2)` devuelve `[2,4]`

ITERACIÓN - TIPO WHILE LOOP (CHEQUEO AL INICIO)

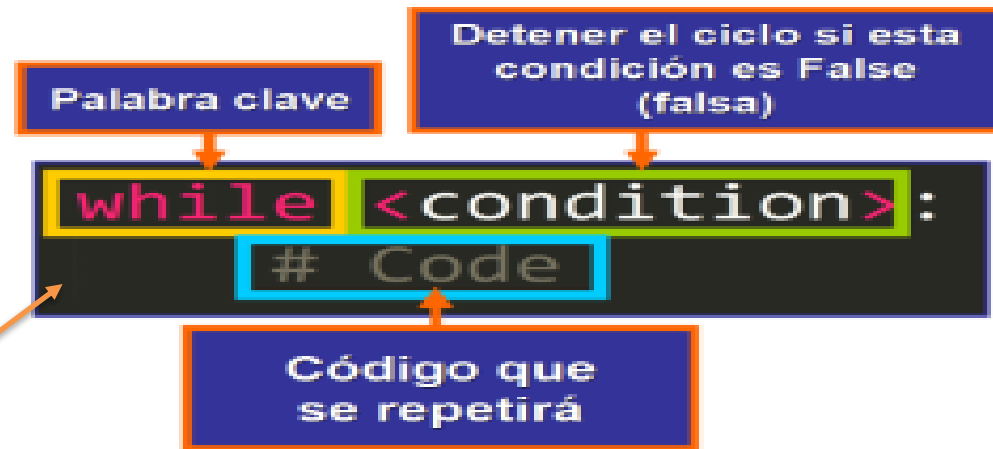
while

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

PASCAL	while condición do sentencia begin end
C, C++	while (condición) sentencia; {....}
ADA	while condición sentencia end loop;
PYTHON	while condición : Sentencia 1 Sentencia 2 sentencia n Nota: break se puede usar para salir cuando se desea

ITERACIÓN - TIPO WHILE LOOP (CHEQUEO AL INICIO)

While de PYTHON



Ciclo while (sintaxis)

La [guía de estilo de Python](#) (PEP 8) recomienda usar 4 espacios por nivel de indentación (sangría).

Estos son los elementos principales (en orden):

- La palabra clave `while` (seguida de un espacio).
- Una condición que determina si el ciclo continuará su ejecución o no en base a su valor (`True` o `False`).
- Dos puntos (`:`) al **final** de la primera línea.
- La secuencia de instrucciones o sentencias que se repetirán. A este bloque de código se le denomina el "cuerpo" del ciclo y debe estar indentado. Si una línea de código no está indentada, no se considerará parte del ciclo.

ITERACIÓN: - TIPO WHILE LOOP (CHEQUEO AL FINAL)

Until-Repeat y Do-While

- Estructuras que permite **repetir un proceso "hasta" que se cumpla una condición**. En definitiva, permite ejecutar un bloque de instrucciones *mientras no se cumpla una condición dada (sea falso)*.
- La condición/expresión se evalúa al final del proceso, por lo que *por lo menos 1 vez el proceso se realiza*

PASCAL	repeat Sentencia until condición;
C, C++	do sentencia; while (condición);

ITERACIÓN: LOOP

ADA tiene una estructura iterativa

Se puede
combinar
con
estructuras
FOR y WHILE

```
[ identificador_bucle : ] loop  
    secuencia_de_sentencias  
end loop [ identificador_bucle ] ;
```

Ejemplo:

```
Vida: loop  -- El bucle dura indefinidamente.  
    Trabajar;  
    Comer;  
    Dormir;  
end loop Vida;
```

Este bucle sólo se puede abandonar si alguno de los procedimientos levanta una excepción.

ITERACIÓN: LOOP ADA

- De este bucle se **sale** normalmente, mediante una **sentencia "exit when condition"**

```
loop
  Alimentar_Caldera;
  Monitorizar_Sensor;
  exit when Temperatura_Ideal;
end loop;
```

- O con una **alternativa** que contenga una cláusula **"exit"**
 - **loop**
 -
 - **exit when** condición;
 - ...
 - **end loop;**