Clase 9

Estructuras de control

¿Qué son las estructuras de control?

- Medio por el cual los programadores pueden determinar el flujo de ejecución entre componentes de un programa.
    - Las estructuras de control se dividen en:
        - Estructuras a nivel de sentencia.
        - Estructuras a nivel de unidad.

Estructura a nivel de unidad

- Es cuando el flujo de control se pasa entre unidades (procesos o funciones). Acá intervienen los pasajes de parámetros.

Estructura a nivel de sentencia

- Se dividen en tres grupos.

Secuencia

- Flujo de control más simple que indica la ejecución de una sentencia a continuación de otra.
- El delimitador más usado para separar una sentencia de otra es el ";".
    - Existen lenguajes sin delimitador que establecen que por cada línea de código debe haber una instrucción -> lenguajes orientados a línea.
        - Fortran, Basic, Ruby, Python (GOD)
- Se pueden agrupar varias sentencias en una, a esto se le dice sentencia compuesta y llevan delimitadores del estilo begin y end o {}, esto es un bloque de acciones.

Asignacion

- Es una sentencia que produce cambios en los datos de la memoria de forma que al L-Valor de un objeto de dato se le asigna el R-Valor de una expresión.
- Sintaxis de 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

Si usamos un lenguaje con = tanto como para asignar como para realizar expresiones.
¿Cómo las distinguimos?

Distinción entre sentencia de asignación y expresión:

- En lenguajes convencionales existen diferencia entre sentencia de asignación y expresión.

        - Las sentencias de asignación devuelven el valor de la expresión y modifican la posición de memoria.
    - La mayoría de los lenguajes de programación requieren que sobre el lado izquierdo de la asignación aparezca un l-valor
    - Pero en C se define la sentencia de asignación como una expresión que tiene efectos colaterales
        - Las sentencias de asignación devuelven valores.
        - C evalúa de derecha a izquierda.
        -
        - C a su vez también toma como verdadero todo lo que sea distinto de 0.
            - Por si usás = y no ==.

Selección

- Permite que el programador pueda expresar una elección entre un cierto número posible de sentencias alternativas.
- Evolución de la selección
    - ❖ If lógico de Fortran -> if (condición) sentencias
        - No existía el else
    - ❖ If then else de Algol -> if (condición) then sentencias else sentencias
        - Finalmente, el else para más caminos posibles.

    Esto trajo problemas:
        - Ambiguedades: los lenguajes no establecían como se asociaban los else con los if abiertos.
            - La ambigüedad se resolvía estableciendo que el else se cerraba con el último if abierto que correspondía.
- Solución adicional: sentencia de cierre de bloque condicional, ejemplo end, if, fi, etc.
    - Desventajas de esto: ilegibilidad, si tengo muchos if anidados se complica leer el código. Para solucionar esto se utiliza Begin End o {}
- If then else de Python, no ambiguo 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!!'
```

    - Los dos puntos son obligatorios al final del if, else y del elif.
    - La indentación es obligatoria cuando se codifican las sentencias de los posibles caminos.

Sentencia condicional corta

- cond ? B: C; en C
- A if C else B en Python.
    - Devuelve A si se cumple la condición C, sino devuelve B.

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

Circuitos cortos (short-circuit):

En un lenguaje que utiliza el circuito corto, la evaluación de una expresión lógica se detiene tan pronto como se determina el resultado final sin necesidad de evaluar el resto de la expresión.

- Operador "y" (&&): Si la primera parte de la expresión es falsa, el resultado final será falso sin importar el valor de la segunda parte. Por lo tanto, en un circuito corto, si la primera parte es falsa, la segunda parte no se evalúa.
- Operador "o" (||): Si la primera parte de la expresión es verdadera, el resultado final será verdadero sin importar el valor de la segunda parte. Por lo tanto, en un circuito corto, si la primera parte es verdadera, la segunda parte no se evalúa.

Circuitos largos (non-short-circuit):

En un lenguaje que utiliza el circuito largo, la evaluación de una expresión lógica continúa evaluando todas las partes de la expresión, incluso si el resultado final ya se ha determinado.

- Operador "y" (&&): En un circuito largo, ambas partes de la expresión se evalúan independientemente del valor de la primera parte. Esto asegura que se realicen todas las comprobaciones necesarias.
- Operador "o" (||): En un circuito largo, ambas partes de la expresión se evalúan independientemente del valor de la primera parte. Esto asegura que se realicen todas las comprobaciones necesarias.

- IMPORTANTE: Python evalúa las condiciones usando circuito corto.

Selección Múltiple

La sentencia Select de PL/1 (Programming Language 1) es la que introduce la selección entre dos o más opciones de forma que se reemplacen if's anidados muy largos.

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

Selección Múltiple en Pascal

- Incorpora la restricción de que los valores de la expresión deben ser de tipo ordinal y ramas con etiquetas (esto no lo entendí). No importa el orden en el que aparecen las ramas y existe la opción else que ocurre cuando el valor ordinal no corresponde con ninguna expresión.
    - El default.
- El case de Pascal es inseguro por que no establece obligatoriamente qué sucede cuando un valor no cae dentro de las alternativas puestas.
    - Esto se debe a que el Else es OPCIONAL.

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

    - Si no estuviera el Else y yo ingresara un valor distinto a 1, 2 o 3. Entonces provoco un error.

Selección Múltiple en ADA

- Las expresiones pueden ser solamente de tipo entero o enumerativas.
- En la selección de ADA se DEBE estipular todos los valores posibles que pueda tomar la expresión.

- ADA posee la cláusula Others (que es OBLIGATORIA, sino no compila). Dicha cláusula se puede utilizar para representar los valores que no fueron especificados de forma explícita.

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

```
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 en C, C++

- Se usa Switch.
- Cada rama (camino) se etiqueta por uno o más valores constantes.
- Cuando la opción coincide con una etiqueta del Switch se ejecutan las sentencias asociadas a la rama y se continúan ejecutando las sentencias de las otras ramas.
    - Es decir, si debajo de mi caso particular tengo definidos más casos particulares, estos se ejecutan.
    - Para solucionar esto, se usa la sentencia break que provoca la salida de una rama que se está ejecutando (lo mejor es ponerlo al final de todas las sentencias de cada rama).
- La cláusula default nos sirve para los casos donde el valor no coincide con ninguna opción establecida.
    - Como el else de Pascal también es opcional.

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

Selección múltiple en Ruby

- Constructor Case expresion, 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 coincde con inguna expresión irá al bloque else igual que por defecto.
- else es opcional, esto puede traer efectos colaterales

```
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!

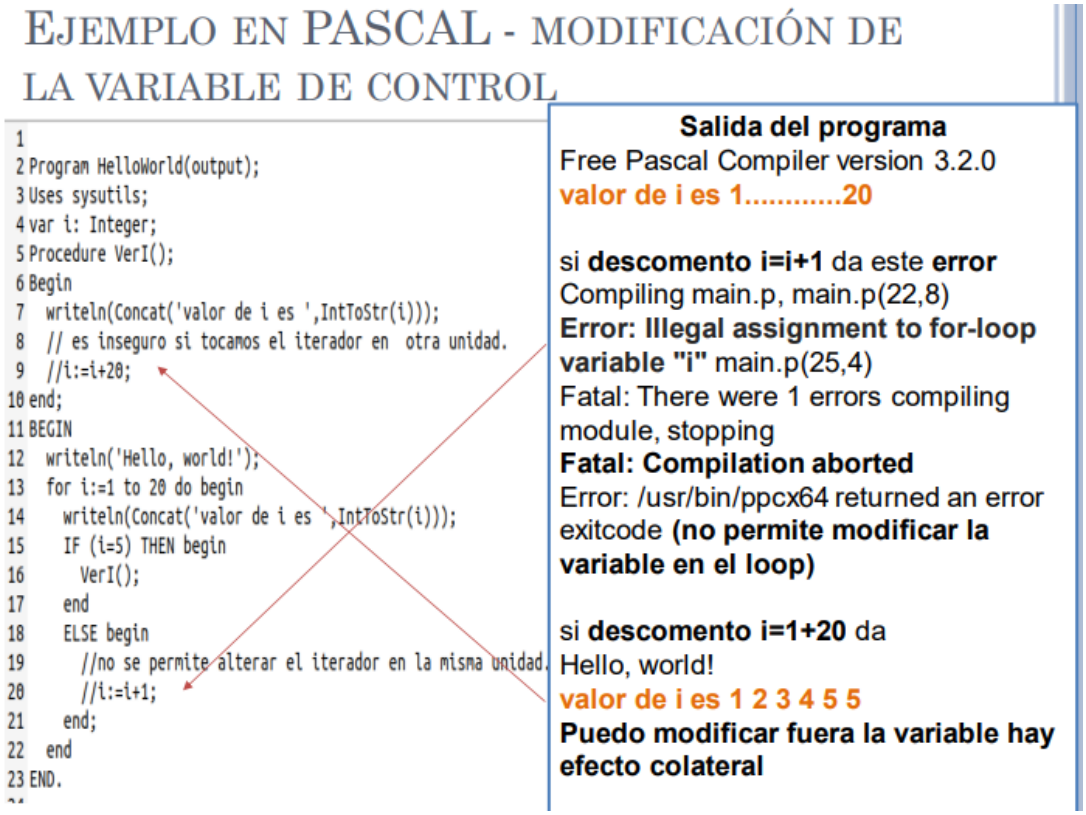
Iteración

- Estas instrucciones se utilizan para representar acciones que se repiten un cierto número de veces.
- Evolución:
  - Empieza con sentencia Do de Fortran
    - Do etiqueta variable-control=valorInicio,valorFin
      Sentencias
- La variable de control del Do de fortran SOLO puede tomar valores enteros y nunca deberá ser modificada por otras sentencias dentro del ciclo
- El Fortran original (el más viejo será) evaluaba si la variable de control había llegado al límite al final del búcle, entonces por lo menos se ejecutaba una vez.
- La iteración esta comumente agrupada 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)

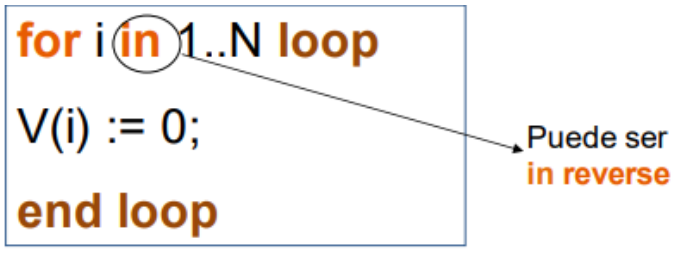
For de Pascal, ADA, C, C++

- La variable de control puede tomar cualquier valor ordinal.
- Pascal estándar NO permite que se toquen los valores del límite inferior y superior, ni el valor de la variable de control.
- La variable de control puede ser de cualquier tipo ordinal.
- Fuera del bloque el valor de la variable de control se asume indefinida.



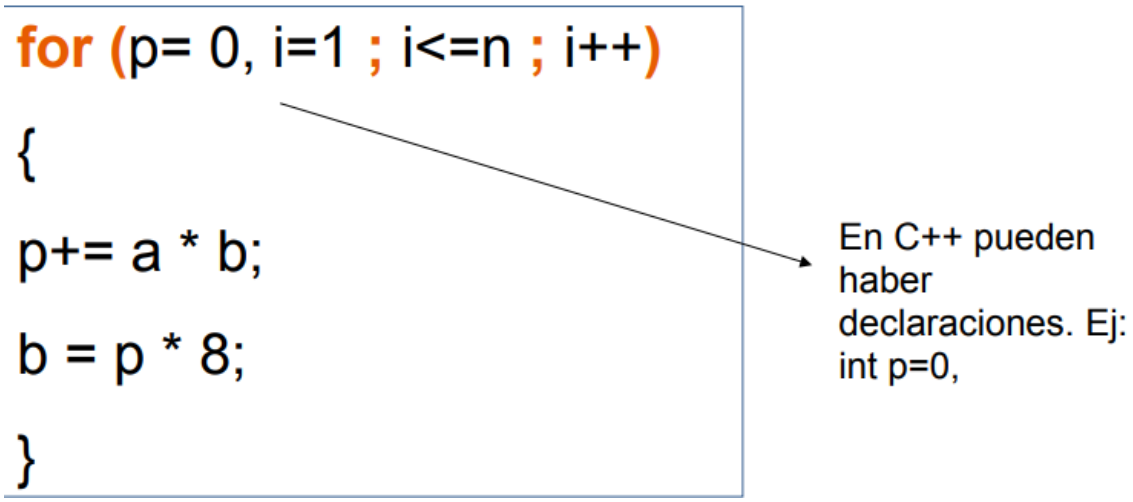
Específicamente sobre el for para ADA

- 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



Específicamente sobre el for para C y C++

- En ambos lenguajes el for se compone de tres partes: la inicialización de la variable de control, y dos expresiones, cada parte SEPARADA por ";" .
    - La primera expresión (parámetro de la mitad) es el testeo realizado antes de cada iteración (evaluación de una condición lógica).
        - Hay que colocarlo, sino quedamos en loop infinito.
        - Sale del ciclo si la expresión no se cumple
    - En el primer y segundo parámetro se pueden colocar sentencias separadas por coma.
    - La segunda expresión suele ser para incrementar/decrementar la variable de control.



Específicamente en 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 en Python

- Se itera sobre una secuencia de datos, dichas secuencias pueden ser listas, tuplas, etc.
- Para iterar sobre un rango de valores basado en una secuencia numérica se usa la función Range()

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

- El for finaliza su ejecución cuando se recorren todos los elementos.
- La función range() que puede tomar 1,2 o 3 argumentos.
- Dicha función range nos permite simular la sentencia FOR de otros lenguajes:
    - For i in range(valor-inicial, valor-final +1): acciones

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

While

- Estructura que nos permite repetir sentencia/s mientras se cumpla una condición.
- La condición es evaluada antes de que se ejecuten las sentencias.

- 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
```

Until

- Estructura que nos permite repetir sentencia/s mientras se cumpla una condición.
- PERO la condición es evaluada al final de la ejecución de la/s sentencia/s, por lo menos el bloque de sentencias se ejecuta una vez.

- En Pascal:

```
repeat
sentencia
until condición;
```

- En C, C++:

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

LOOP de Ada

- Si, Ada tiene una estructura iterativa adicional que es un bucle simple.

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

- De este bucle solo se puede salir mediante una sentencia exit when o una alternativa que contenga una cláusula exit.

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

    - Si no está ninguna de las dos sentencias entonces es loop infinito.
    - A menos que alguna instrucción dentro del bloque levante una excepción.