# Estructuras de control

1. ¿Qué son las estructuras de control?

Son el medio por el cual los programadores pueden determinar el flujo de ejecución entre los componentes de un programa. Existen dos tipos:

- **A nivel de unidad**: cuando el flujo de control se pasa entre unidades. Intervienen los pasajes de parámetros.
- **A nivel de sentencia**: hay, a su vez, otros tres tipos:
    - **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, Python. Se pueden agrupar varias sentencias en una sola llamada sentencia compuesta. Por lo general, llevan delimitadores como Begin y End(ADA, Pascal, etc). Ej: Ada, { y } en C, C++, Java, etc.
      Un ejemplo de secuencia es la asignación. La asignación es una sentencia que produce cambios en los datos de la memoria. Asigna al l-valor de un dato objeto el r-valor de una expresión. Su sintaxis varía en diferentes lenguajes.
    - **Selección**: Esta estructura de control permite que el programador pueda expresar una elección entre un cierto número posible de sentencias alternativa. Es una de las estructuras de control que más ha evolucionado.
      El primer if se remonta a Fortran, el cual se ejecutaba solo si la condición era verdadera(*If(condición lógica) sentencia*).
      Luego llegó el *if then else* de Algol, sin embargo, este presentaba un problema grave de ambigüedad ya que no establecía por lenguaje cómo se asociaban los else con los if abiertos. Este problema fue arreglado con Pl/1, Pascal y C, en los que no habia ambigüedad pero si un problema de legibilidad: dado que establecían por lenguaje que cada else cierra con el último if abierto, había que indicar cuándo cerraba el último if y así aparecieron sentencias de cierre del bloque condicional, como son endif, fi, etc. Esto significaba que programas con muchos if anidados podrían quedar ilegibles.
      Python fue el que solucionó ambos problemas con el if ya que los **:** son obligatorios al final del if, else y del elif y la indentación es obligatoria al colocar las sentencias correspondientes tanto al if, else y el elif.
      Una variante de la selección es la selección múltiple que se usa para evitar tantos if anidados. PL/I fue el primero en incorporla expresando que los valores de la expresión sean ordinales y ramas con etiquetas. No importa el orden en que aparecen las ramas. Tiene la opción “else”. Sin embargo, es inseguro porque no establece qué sucede cuando un valor no cae dentro de las alternativas puestas.
      Esto fue solucionado en ADA en que había una cláusula extra, *others*, que se podía utilizar para aquellos valores que no caían en las alternativas. En ADA, además, las expresiones podían ser solamente de tipo entero o enumerativas y en las selecciones se debía estipular todos los valores posibles que podía tomar la expresión. Si no se colocaba la rama para un posible valor o si no aparecía la opción *others* en esos casos, la compilación fallaba.
      Por su parte, C y C++ también tienen una cláusula *switch* para estos fines. En este caso, cada rama es etiquetada por uno o más valores constantes. Además, cuando la opción coincide con una etiqueta del *switch*, se ejecutan las sentencias asociadas y se continúa con las sentencias de las otras entradas. Esto tiene un efecto que podía ser no deseado(el hecho de que se continúe con las sentencias de las otras entradas), el cual se soluciona con la sentencia *break*. También tiene una cláusula que sirve para los casos en que el valor no coincida con ninguna de las opciones establecidas: el *default*.
      Ruby también lo implementa y tiene una particularidad interesante: si no entra en ninguna opción, sigue la ejecución y la variable no tendrá ningún valor.
    - **Iteración**: puede ser un for o un while.

- **For**: éste tipo de instrucciones se utilizan para representar aquellas acciones que se repiten un cierto número de veces. Es también una de las sentencias que más historia tiene.
  Empezó como un *Do label var-de-control=valor1, valor2 sentencias label continue* en Fortran. En este caso, la variable de control solo podía tomar valores enteros y lo que se evaluaba era si la variable de control había llegado al límite al final del bucle, esto significa que siempre, al menos una vez, el bucle era ejecutado.
  Luego evolucionó a un *For* de Pascal y ADA. En este caso, la variable de control podía tomar cualquier valor ordinal, no solamente enteros. Además, el valor de la variable fuera del bloque se asume indefinida. En el caso de Pascal, no está permitido que se toquen ni los valores del límite inferior y superior, ni el valor de la variable de control. En Ada, no debe declararse el iterador. C y C++ también implementaron el *for* pero con algunas consideraciones: * Se compone de tres partes: una inicialización y dos expresiones. * En el primer parámetro(inicialización) se pueden colocar sentencias separadas por comas. * La primer expresión(2do. parámetro) es el testeo que se realiza antes de cada iteración. Si no se coloca, el *for* queda en loop indefinidamente. * La segunda expresión indica cómo avanza el iterador. Python también implementó ésta estructura la cual permite iterar sobre una secuencia. Las secuencias pueden ser: una lista, una tupla, etc. Se puede simular el *for* de otros lenguajes utilizando la función *range*.
- **While**: estructura que permite repetir un proceso mientras se cumpla una condición. La condición se evalúa antes de que se entre al proceso.
- **Until**: estructura que permite repetir un proceso mientras se cumpla una condición. La condición se evalúa al final del proceso, por lo que el bucle se realiza al menos una vez.
- **Loop**: estructura que solo existe en ADA. De este bucle se sale normalmente, mediante una sentencia "exit when" o con una alternativa que contenga una cláusula "exit".