



### **AST** Revisited

Diseño de Compiladores

Dr. Víctor de la Cueva

vcueva@itesm.mx



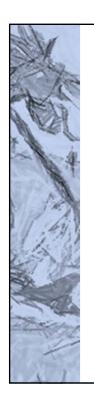
## Objetivos del Parser

- Una vez implementado un parser debe quedar claro que dos de sus principales objetivos son:
  - Verificar la sintaxis de nuestro programa
  - Crear el AST
- El árbol que regresa podría ser el Árbol de Parseo (o de análisis sintático), en el cual, cada No-terminal tiene un nodo interno y cada terminal tiene un nodo hoja.
  - Sin embargo, este árbol es muy grande y tiene información que no se usa en los pasos siguientes.
  - Si se elimina esta información obtenemos el AST.



#### Obtención del AST

- El AST se puede obtener a partir del PT (parser tree) mediante su recorrido y eliminación de los nodos que no hacen falta en las siguientes etapas.
- Sin embargo, para evitar crear y borrar nodos sin información relevante, la forma más común de obtenerlo es irlo formado durante el proceso de parseo:
  - En los procedimientos del parser verificamos si tenemos que crear un nodo, y de ser así, el procedimiento lo volvemos función para que regrese el apuntador a la raíz del nodo creado (el cual tendrá uno o más hijos)
  - El AST se va formando conforme se hace el parseo, es decir, conforme se llaman a las funciones adecuadas y éstas regresan la raíces de los subárboles que crean, y se asignan a los hijos de un nodo superior.



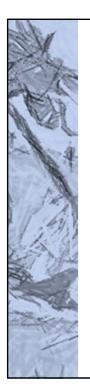
### Estructura del AST

- No existe una estructura estándar para el AST.
- Cada persona lo puede implementar de una forma diferente:
  - En cuanto a la estructura misma o la información que contienen los nodos, de acuerdo a lo que le haga falta para cumplir con las dos partes que faltan del compilador.
  - Desde luego que la estructura también puede cambiar debido al lenguaje que se está compilando
    - Por ejemplo, no todos inician un programa con una palabra específica como program o class.



# Ejemplo en TINY

- Un programa en TINY es una secuencia de sentencias.
- Un AST en TINY se puede representar, al menos de tres formas:
  - Una secuencia de nodos de sentencia (donde el primero se puede considerar la raíz).
  - Un nodo programa con muchos hijos que son nodos de sentencia.
  - Un nodo programa con un solo hijo, que es un arreglo de nodos de sentencia.
- Para un nodo de sentencia:
  - Todas las sentencias se pueden ver como un operador con un número diferente de hijos (operandos).
    - Entonces, podemos tener un solo tipo de nodo para cualquier sentencia, donde la raíz es el operador con un número variable de hijos.
    - O, pensar en diferentes tipos de nodos, uno para cada tipo de sentencia.



# Estructura anidada en los programas

- Como los programas, prácticamente de cualquier lenguaje, están compuestos de bloques, que contienen sentencias u otros bloques:
  - Los bloques son estructuras anidadas
- Los programas crean árboles que tienen como hijos otros árboles correspondientes a las sentencias del lenguaje o a un nuevo bloque, es decir:
  - Los árboles también tienen una estructura de anidación (o recurrencia)



#### Recomendaciones

- Deje su árbol lo más simple que pueda (reglas KISS), siempre y cuando siga funcionando para usarse en las siguientes etapas del compilador.
- Diseñe su árbol en forma dinámica, es decir:
  - · Haga un diseño inicial e impleméntelo
  - Si al probarlo encuentra que le hacen falta cosas, regrese al diseño y agrégueselo
  - Repita este proceso hasta que el compilador funcione correctamente



#### Recordatorio

- Hasta este momento del curso, no sabe a ciencia cierta cuál será la función del AST que creó.
- Por esta razón, su diseño e implementación deben ser dinámicos y esperar hasta implementar la semántica y la generación de código.
- Sin embargo, una vez implementado un AST los cambios en el mismo debido a requerimientos de las etapas siguientes:
  - Deben ser mínimos y
  - No en la estructura del software sino en su contenido



## Referencias

- A.V.Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. 2<sup>nd</sup> Pearson (2012).
- K.C. Louden. Contrucción de Compiladores: principios y práctica. Thomson (2004).
- Alex Aiken. Compilers. Stanford Online (2018).
  - https://lagunita.stanford.edu/courses/Engineering/Compilers/Fall2014/about
- R. Mak, Writing Compilers and Interpreters: A Software Engineering Aproach. 3<sup>rd</sup> ed, Wiley (2009).