Tabla de símbolos

Diseño de Compiladores

Dr. Víctor de la Cueva
vcueva@itesm.mx

Tabla de símbolos (ST, por sus siglas en inglés)

- El parser de un compilador construye y mantiene la tabla de símbolos.
- La tabla de símbolos almacena información acerca de los tokens del programa fuente, principalmente de los identificadores.
- La tabla de símbolos es un componente fundamental en la interface entre el análisis semántico y sintáctico.

ST como una estructura de datos

- Mantener una ST bien organizada es una de las tareas más importantes de un compilador.
- Conforme un compilador traduce programa fuente, debe ser capaz de colocar nueva información o actualizar la existente en la ST en forma eficiente.
- En enfoque es:
  - Crear el diseño conceptual de una ST.
  - Desarrollar una implementación en un lenguaje de programación que represente el diseño.

Información en la ST

- Durante el proceso de traducción, el compilador crea y actualiza entradas en la ST para que guarde información importante de ciertos tokens del programa fuente.
- Cada entrada tiene un nombre, el cual es el string del token.
  - La entrada también contiene otra información acerca del identificador.
  - A medida que se traduce el programa fuente, el compilador busca y actualiza dicha información.

Operaciones básicas de la ST

- ¿Qué información debe contener la ST?
  - Cualquier información que sea útil
- Por ejemplo, una entrada de la ST para un identificador contiene típicamente:
  - Su tipo
  - Estructura
  - Y cómo está definido
- Sin importar cuál información se guarda en la ST, las operaciones básicas que debe soportar son:
  - Introducir nueva información (altas)
  - Buscar información existente (consultas)
  - Actualizar información existente (actualizaciones)

Bloques en los lenguajes

- La mayor parte de los lenguajes modernos están formados por módulos:
  - {…}
  - begin … end
  - Funciones, procedimientos, métodos, etc.
  - Tipos: estructuras, registros, etc.
  - Clases
- Cada bloques:
  - Tiene declarados variables, constantes, otros bloques, etc.
  - Define un alcance (scope) local para los identificadores.

ST: bloques, declaraciones y AST

- En el AST:
  - Los nodos que inician un bloque están bien definidos (function-return, program-end, {…}, begin-end, class, etc.).
  - Los nodos que inician declaraciones también (var, int, float, type, lista de parámetros, etc.)
  - Si no hay parte declarativa, cada vez que se encuentre un identificador se considera que se está declarando (revisar reglas del lenguaje)
- Cada vez que se encuentre un bloque:
  - Se debe crear una nueva ST (local), a la que se le deberán agregar los identificadores que se encuentren dentro del bloque, si es que estos están declarados ahí.
  - Se insertarán TODOS los elementos que estén declarados en dicho bloque.

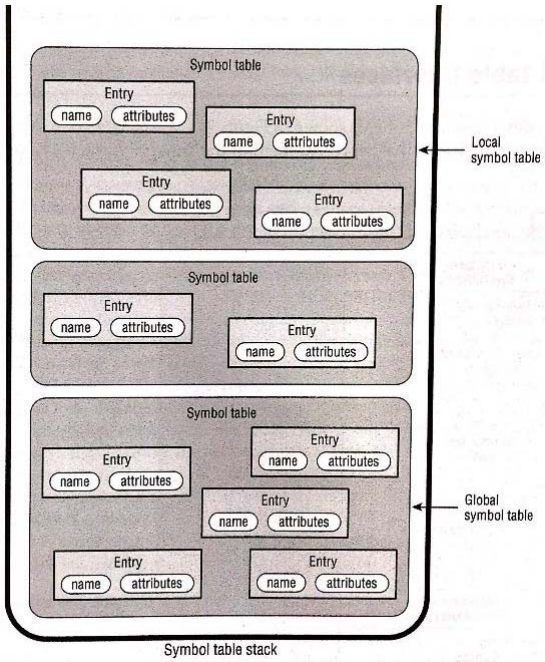
El stack de STs

- Si un lenguaje está compuesto de sólo una secuencia de instrucciones (statements o estatutos, e.g. Basic, Fortran o TINY), bastará sólo una ST.
- Si el programa es estructurado en bloques (e.g. Pascal, C, C++, Java o C-), se requerirán múltiples STs:
  - Una ST global, para el programa principal
  - Una ST local, para cada procedimiento, función, estructura o clase.
- Debido a que estas estructuras pueden estar anidadas, se requerirá un STACK de ST:
  - La ST en el *top* contendrá información sobre el programa o función que se esté parsenado en ese momento.

Una entrada de la ST

- Cada entrada de la ST contiene información acerca de un token:
  - Típicamente un identificador
    - Nombre de la entrada
    - Información del token en forma de atributos
- Una ST busca las entradas usando los nombres como claves de búsqueda.

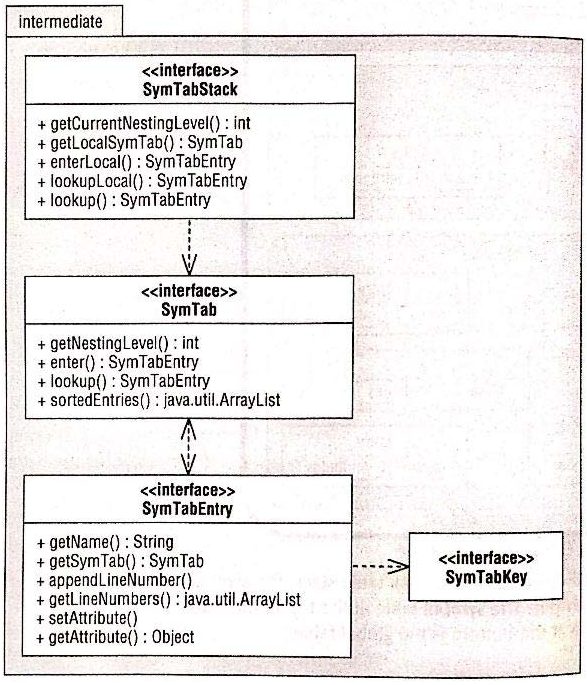
Diseño conceptual del stack de STs



Pág 99 Mak

Diseño conceptual y DS

- Del diseño conceptual sólo se debe entender:
  - Cuáles son los principales componente de una ST
  - Cuáles son sus roles
  - Cómo se relacionan unos con otros
- Es importante entender los conceptos anteriores para decidir cuál es la mejor estructura de datos (DS) que se puede utilizar en su implementación.



API recomendada para la ST

- De los componente de la ST del modelo conceptual, se puede derivar un diagrama UML para su API.

Pág 100 Mak

Generalización de la ST

- Aún y cuando haya sólo una ST en el programa, es conveniente implementarla como un stack con las siguientes operaciones:
  - ***Enter***: introduce una nueva entrada en la ST local, la cual se encuentra en el top del stack
  - ***Local look up***: buscar una entrada sólo en la tabla local
  - ***Global look up***: buscar una entrada en todas las ST del stack
- Una vez que la entrada ha sido encontrada se puede actualizar (***update***) su contenido.

Ejercicio

- Usando el AST entregado por su parser haga una función que, cada vez que se encuentre un identificador, le agrega a su ST global, su número de línea.
  - Por el momento no importa si el identificador aparece dentro de una función (en cuyo caso se tendría que crear una nueva ST local)
- Al final, imprimir la ST global creada.
  - E.g.

Identificador	Números de línea
abc	31 33
epsilon	4 33
newton	7 14 16 17 19

Comentario sobre la creación de la ST

- Es posible crear las ST desde el léxico y el parseo.
  - Es más simple si se hace después.
- Si alguien quiere modificar su léxico o parser para ir creando la ST lo puede hacer.

Referencias

- A.V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. 2nd 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*. 3rd ed, Wiley (2009).