Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

CI3725 - Traductores e Interpretadores

Diciembre - Marzo 2014-2015

Setlan - Etapa III

Tabla de Símbolos y Chequeo de Tipos (8%)

Especificación de la entrega

Hasta este momento hemos construido un analizador lexicográfico y sintáctico para el lenguaje de programación `Setlan`. Esta siguiente etapa del desarrollo corresponde a la verificación del uso correcto de tipos dentro del lenguaje además de la implementación y uso de una **Tabla de Símbolos**.

Para este punto, su interpretador de código `setlan` debe de reportar -si existen- los siguientes errores:

- Errores de redeclaraciones de variables dentro de un mismo bloque `using ... in`.
- Utilización de variables no declaradas en ese alcance.
- Modificación de variables de iteración.
- Errores de tipos, como por ejemplo: Intentar sumar una variable del tipo `int` con una del tipo `bool`.

Para ello deberán implementar y *llenar* una **Tabla de Símbolos**:

Estructura de datos que posee información de declaración y contexto de cada identificador incluido en un código en `Setlan`. Recuerden que a pesar de no ser declaradas dentro de un bloque `using ... in`, las variables de iteración son **también** tomadas en cuenta para por la tabla de símbolos. La implementación debe estar en un módulo o archivo aparte y **debe incluir** como mínimo los siguientes procedimientos (si está programando en castellano, puede colocar los equivalentes):

- Constructor de una tabla de símbolos vacía:
  - Esto depende del lenguaje utilizado.
- Insertar un elemento en una tabla de símbolos:
  - Python / Ruby: `table.insert(...)`
  - Haskell: `insert :: ... -> SymbolTable -> SymbolTable`
- Eliminar un elemento de una tabla de símbolos:
  - Python / Ruby: `table.delete(...)`
  - Haskell: `delete :: ... -> SymbolTable -> SymbolTable`
- Actualizar la información de un elemento de una tabla de símbolos:
  - Python / Ruby: `table.update(...)`
  - Haskell: `update :: ... -> SymbolTable -> SymbolTable`
- Determinar si un elemento se encuentra dentro de una tabla de símbolos:
  - Python / Ruby: `table.contains(...)`
  - Haskell: `elem :: ... -> SymbolTable -> Bool` (siguiendo estándares de Haskell)

- Obtener la información de un elemento que se encuentra en la tabla de símbolos:
  - Python / Ruby: `table.lookup(...)`
  - Haskell: `lookup :: ... -> SymbolTable -> ...`

Los parámetros de cada método se dejan a preferencia y conveniencia del equipo. La implementación de cada uno de estos métodos será tomada en cuenta al momento de la corrección.

Pueden suponer como valores por defecto para una variable recién declarada, los siguientes:

Tipo	Valor
`int`	`0`
`bool`	`false`
`set`	`{}`

Ejecución

Para la ejecución del interpretador su programa deberá llamarse `setlan` y recibirá como primer argumento el nombre del archivo con el código en `Setlan` a analizar.

Luego de realizar el análisis lexicográfico y sintáctico, de no presentarse errores relacionados con éstos, se procede a verificar errores de alcance, redeclaración y no-declaración utilizando la tabla de símbolos. Finalmente se chequea la correctitud en el uso de tipos.

En caso de encontrar errores lexicográficos y sintácticos, deben reportar **sólo** estos. Los errores relacionados a la verificación haciendo uso de la tabla de símbolos y errores de mal utilización de tipos deberán ser reportados **juntos**.

Por salida, se debe mostrar la tabla de símbolos. En caso de encontrar errores, ésta no debe ser impresa. El formato de impresión debe ser similar al ejemplo mostrado a continuación:

Programa:

```
program {
    using
        int i,j,k;
        bool b,c;
        set s;
    in
    ...
    if(true) {
        using
            set i;
            bool b;
        in
        i = {1,2,3};
        # nótese que al lado izquierdo del `min`, `i` es int
        # y al lado derecho es set
        for i min i <+> 2 do {
          using bool i; in          # una `i` más
          ...
        };
    } else {
       using bool i; in
       ...
    }
}
```

Salida correspondiente:

```
Symbol Table

SCOPE
  Variable: i | Type: int  | Value: 0
  Variable: j | Type: int  | Value: 0
  Variable: k | Type: int  | Value: 0
  Variable: b | Type: bool | Value: false
  Variable: c | Type: bool | Value: false
  Variable: s | Type: set  | Value: {}
  SCOPE
    Variable: i | Type: set  | Value: {}
    Variable: b | Type: bool | Value: false
    SCOPE
      Variable: i | Type: int | Value: 0
      SCOPE
        Variable: i | Type: bool | Value: false
      END_SCOPE
    END_SCOPE
  END_SCOPE
  SCOPE
    Variable: i | Type: int  | Value: 0
  END_SCOPE
END_SCOPE
```

Análogamente un caso con errores debe seguir el formato:

Programa:

```
program {
    using
        int i,j,k;
        bool b,c;
        set s;
    in
    ...
    if(true) {
        using
            int i,i;
            bool b;
        in
        ...
    }
}
```

Salida correspondiente:

```
Error en línea X, columna Y: La variable 'i' ya ha sido declarada en este alcance.
```

Errores de chequeo de tipos:

```
program {
    using
        int i;
    in
    i = 10;
    if (i + 2)
        for k in {1,2,3} + 1 do
            print k
    ;
}
```

Salida correspondiente:

```
Error en línea 6, columna 5: Instrucción 'if' espera expresiones de tipo 'bool', no de tipo 'int'.
Error en línea 7, columna 18: Operando '+' no sirve con operadores de tipo 'set' y 'int'.
```

Flags

Adicionalmente, para esta etapa de desarrollo su programa principal debe ser capaz de imprimir además de la tabla de símbolos del código: La lista de *tokens* encontrados y el AST de la estructura del código en `Setlan`. Para ello el usuario que

ejecute el interpretador colocará los siguientes `flags` para determinar **qué** imprimir:

- `-t`: Imprime la lista de *tokens*.
- `-a`: Imprime el AST asociado a la estructura del programa.
- `-s`: Imprime la tabla de símbolos.

Note que se puede imprimir más de una de las especificaciones de cada `flag`, por lo que introducir `-t -s` imprime por salida estándar *tanto la lista de tokens encontrados como la tabla de símbolos*.

Implementación

Para la implementación del interpretador del lenguaje `Setlan`, pueden escoger uno (1) de los tres (3) lenguajes de programación a continuación. Para cada uno de ellos se indica las herramientas disponibles para el desarrollo de un interpretador de código. Recuerden que para esta etapa de desarrollo deberán seguir empleando **el lenguaje de programación utilizado en la primera y segunda etapa** así como deberán seguir con él en etapas posteriores.

- *Python:*
  - Interpretador *python* 2.7.
  - *Python Lex-Yacc (PLY).*
- *Ruby:*
  - Interpretador *ruby* 1.9.
  - *Racc.*
- *Haskell:*
  - Compilador *ghc* 7.6.3 ó 7.8.3.
  - *Alex* y *Happy.*

Formato de Entrega:

Deben enviar un correo electrónico a **todos los preparadores** con el asunto: `[CI3725]eXgY` donde `X` corresponde al número de la entrega e `Y` al número del equipo. El correo debe incluir un archivo comprimido **.zip** que contenga:

- Código fuente debidamente documentado.
- En caso de utilizar Haskell, deben incluir un archivo Makefile o cabal. Si su proyecto no compila, el proyecto no será corregido.
- Un archivo de texto con el nombre `LEEME.txt` donde **brevemente** se expliquen:
  - Estado actual del proyecto.
  - Problemas presentes.
  - Cualquier comentario respecto al proyecto que consideren necesario.
  - Este archivo debe estar identificado con los nombres, apellidos y carné de cada miembro del equipo de trabajo.

Recuerde que no cumplir con estas especificaciones puede afectar directamente la nota final de la entrega.

Fecha de entrega:

La fecha límite de entrega del proyecto es el día **martes 24** de febrero de 2015 (semana 10) *hasta* las **11:50pm**. Entregas hechas más tarde tendrán una **penalización del 20%** de la nota. Esta penalización aplica por cada día de retraso.