

Metodología empleada y manual de operación

Diseño de compiladores

Dr. Edgar E. Vallejo

Saúl de Nova Caballero (A01165957)

Diego Galíndez Barreda (A01370815)

18 de febrero de 2016

1. Metodología empleada

A continuación viene un resumen de los cambios que se hicieron a la especificación para poder cumplir todos los requerimientos.

1.1. Expresiones regulares

Para las expresiones regulares, se convirtieron todos los operadores, tipos y palabras claves del lenguaje en tokens. También se verificaron constantes de strings, enteros en hexadecimal y enteros, flotantes en notación científica y sin la notación. Se consideraron todas las cadenas que no tengan saltos de línea como válidas. Se verificó para los identificadores que fueran a lo más de 31 caracteres. En el caso de que un caracter no sea ' ', \n, \r ó \t.

Para acomodar comentarios, se utilizó la instrucción **BEGIN** para determinar los estados de flex. Se creó el estado **C_COMMENT** para determinar cuando te encuentras dentro de un comentario.

1.2. Modificaciones a la gramática

Para poder tener las expresiones con $^+$ y * se descompusieron las expresiones de la siguiente manera:

$$\begin{aligned} Expr &::= Term^+ \\ Expr &::= Term \mid Expr Term \end{aligned}$$

se cambió a:

$$\begin{aligned} Expr &::= Term^* \\ Expr &::= \epsilon \mid Expr Term \end{aligned}$$

En el caso del valorL se modificó la gramática ya que el tipo era:

$$ValorL ::= identificador \mid expr [expr]$$

se cambió a:

$ValorL ::= identificador \mid identificador[expr]$

Para poder hacer los reales y los formales, se separó la gramática en 3 para poder acomodar comas, vacíos y final sin comas.

$reals ::= expr^+, \mid \epsilon$

se cambió a:

$reals ::= \epsilon \mid realsDefEnd$
 $realsDefEnd ::= expr \mid realsDef expr$
 $realsDef ::= expr, \mid realsDef expr,$

1.3. Tabla de símbolos

Para poder manejar adecuadamente los scopes del programa fue necesario utilizar una tabla recursiva de símbolos. Se muestra el concepto en la figura 1.

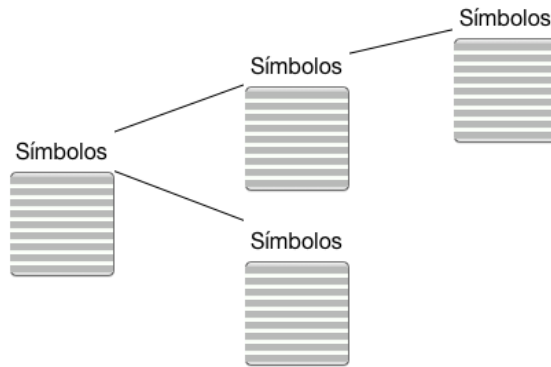


Figura 1: Tabla recursiva de símbolos

En el momento en que se cambia el scope (**instrIf**, **instrWhile**, **instrFor**, **declFunction**), se crea una nueva tabla de símbolos. Todas las tablas tienen una tabla padre, por lo que se puede regresar al scope padre.

Las tablas utilizan los hashes de los símbolos. Si hay colisiones utiliza una lista para agregar varios símbolos al mismo hash.

Los símbolos incluyen un nombre, un tipo y un contador de las veces que se accesa la variable. Tenemos una función lookup que busca recursivamente los símbolos, por lo tanto tenemos overshadowing.

2. Manual de usuario

2.1. Compilación

Para compilar el programa tenemos un Makefile. Lo único que es necesario hacer es entrar al Makefile y modificar las siguientes variables: (**LEX**, **YACC**, **C**) y que apunten a los programas adecuados. Generalmente en sistemas unix van a ser: (**flex**, **bison**, **gcc**).

Una vez modificadas las variables, para compilar el programa utilizamos el comando: **make**.

Para correr las pruebas del sistema utiliza el comando: **make test**.

Para limpiar los ejecutables utiliza el comando: **make clean**.

2.2. Corrimiento programa

Lo anterior te va a generar un ejecutable llamado **cmm**. El comando recibe un sólo argumento, el nombre del archivo a verificar. Por ejemplo, si queremos verificar el archivo **sampleProgram.cmm** se utilizaría el comando: **./cmm sampleProgram.cmm**. El programa puede imprimir dos posibles opciones:

Expression accepted.
Printing symbols table

Después de esto se va a imprimir la tabla de símbolos. El otro caso es:

Error: 'error aquí'

Se va a imprimir toda la lista de errores que se encontraron en el programa.