Procesadores del Lenguaje

Curso 2020-2021

Práctica Final – v2

Traductor de Expresiones a un Lenguaje en Notación Prefija

En sucesivas sesiones se irán detallando las especificaciones del lenguaje que se debe traducir, con algunas explicaciones adicionales, e información técnica. Más adelante se aportarán pruebas de evaluación. La ampliación se hará de forma gradual para intentar evitar problemas técnicos como conflictos del parser.

Trabajo a realizar:

1. Leed el enunciado pasado completo antes de comenzar.
2. Revisad las especificaciones de este documento que tendrá algunas anotaciones añadidas resaltadas. Comprobad que vuestro trabajo ya realizado sigue dichas especificaciones.
3. Renombrad vuestro fichero de código en sucesivas sesiones a trad2.y, trad3.y, ...
4. Continuad con el trabajo en el punto que corresponda, desarrollando los puntos indicados en las especificaciones hasta donde os de tiempo. En futuras sesiones continuaremos con ellas.
5. Es posible intentar abordar puntos que no se han especificado, pero con la reserva de que más adelante se indiquen restricciones no contempladas.
6. Podéis evaluar los resultados de traducción usando el intérprete online https://rextester.com/l/common_lisp_online_compiler. Para ello podéis:
    a. editar un fichero (por ejemplo, `prueba.c`) con una serie de expresiones
    b. ejecutar `cat prueba.c | ./trad >prueba.l`
    c. copiar la salida y pegarla en la ventana del intérprete
    d. con F8 se compila y ejecuta.

Entrega de Sesión:

Seguid las instrucciones del entregador.

Subid el fichero `trad2.y` con el trabajo realizado hasta el momento.

Incluid en la cabecera del fichero dos líneas de comentario iniciales. La primera con vuestros nombres y número de grupo. Y la segunda con los correos electrónicos (separados con un espacio).

Entregad también un documento llamado `trad2.pdf` con una breve explicación del trabajo realizado.

Después de hacer la entrega, descargad vuestro fichero y comprobad que funciona correctamente y que cumple con las indicaciones.

Especificaciones

Proponemos una secuencia de pasos para desarrollar esta práctica. Se recomienda abordar cada uno de los pasos de forma secuencial. Pero en caso de complicarse alguno, se puede pasar al siguiente.

Con un recuadro encontraréis anotaciones para comentar preguntas frecuentes.

1. Retomando la sentencia simplificada para imprimir, el scanner actual procesa el símbolo **#** de forma especial, así que puede que no funcione. Cambiad el símbolo de impresión **#** por el símbolo **$**. Incluid también los paréntesis para englobar la función **$ (<exp>) ;** debe traducirse a **(print <exp>)** .

2. Adaptad la gramática para que traduzca la impresión con múltiples parámetros. La entrada **$ (<exp1>, <exp2>, <exp3>) ;** debe traducirse a **(print <exp1>) (print <exp2>) (print <exp3>)** , en el mismo orden original. Las dos indicaciones previas se modificarán más adelante para emplear ya la palabra printf con los parámetros que corresponden.

3. Incluid la definición de variables en la gramática. La definición en C será **int <id> ;** que debe traducirse a **(setq <id> 0)**. Es necesario incluir un segundo parámetro en Lisp para inicializar las variables. En el caso de la inicialización más simple en C, este valor será 0 por omisión. A partir de este punto comienza la traducción de C a Lisp.

4. Ampliad la gramática para contemplar la definición de una variable con asignación incluida: **int <id> = <cte> ;** que debe ser traducido a **(setq <id> <cte>)**. Utilizamos aquí **<cte>** para representar un valor numérico constante. No consideraremos por ahora expresiones en las declaraciones.

Esto corresponde a la definición de variables globales. Recordamos que en C no está permitido asignar expresiones (con variables y funciones) a una variable global en la instrucción en que es declarada puesto que el proceso tiene que hacerse en tiempo de compilación. La evaluación de expresiones se hace en tiempo de ejecución.

5. En C **main** es el procedimiento/función principal, Debe tener una palabra reservada en la tabla correspondiente, y debe enlazar con el Token **Main**. Ampliad la gramática para reconocer la función **main**. Debe permitir la inclusión de sentencias. Prestad atención al siguiente punto. Incluimos un ejemplo de traducción:

C	Lisp

`int a ;`
`main () {`
`  $ (a + 1) ;`
`}`
	`(setq a 0)`
	`(defun main ()`
	`  (print (+ a 1))`
	`)`

	`(main) ; Para ejecutar el programa`

A través de la gramática es posible obligar a que exista una función **main**.

6. Considerad que la estructura de un programa en C debe ser:

   <Decl_Variables> <Def_Funciones>. En teoría debería poder intercalarse ambos tipos de definiciones, pero eso puede producir conflictos. Por ello seguiremos una estructura fija. Las sentencias que genera la gramática sólo deben aparecer dentro del cuerpo de una función. Revisad la estructura de vuestra gramática. Debe estar diseñada de forma muy cuidadosa y estructurada, intentando que sea lo más jerárquica posible.

Aquí se indican varias cuestiones. 1) Hay que diseñar una gramática lo más jerárquica o estructurada posible. Los nombres de No Terminales deben escogerse con cuidado y deben ser representativos y significativos. Esto evitará la aparición de errores típicos de diseños "enmarañados" o excesivamente complicados. 3) Se sugiere emplear una estructura de programa que empiece con la declaración de variables y luego con la definición de funciones. 4) La recomendación es definir las funciones en orden inverso de jerarquía, empezando con las funciones más sencillas y terminando por la principal, el **main**. Esto permitirá que más adelante el traductor tenga siempre referencia de las funciones que se usan porque ya se han definido previamente. Si se definen primero las funciones principales y luego las de inferior jerarquía, un compilador necesitará hacer averiguaciones sobre el tipo de las funciones que se llaman y que aún no se han definido, o emplear dos pasadas para realizar traducciones parciales y condicionadas. El compilador de C requiere en estos casos de una declaración previa (prototipo) de las funciones. En Lisp deben definirse las funciones antes de ser usadas. Para evitarnos complicaciones con los prototipos usaremos programas en C con las funciones en orden jerárquico inverso. Se sugiere no permitir la mezcla de declaraciones de variables y de definición de funciones para simplificar la gramática y evitar conflictos. No se prohíbe esta opción. Simplemente, no se recomienda recurrir a ello en el inicio de la práctica.

7. Dentro del cuerpo de las funciones pueden declararse variables, que tanto en C como en Lisp serán variables locales. Se emplea la misma traducción de las globales, pero en este caso deben generarse dentro del cuerpo de la función:

C	Lisp

`int a ;`
`main () {`
`  int a = 4 ;`
`  $ (a + 1) ;`
`}`
	`(setq a 0)`
	`(defun main ()`
	`  (setq a 4)`
	`  (print (+ a 1))`
	`)`

	`(main) ; Para ejecutar el programa`

   Hay que tener cuidado a la hora de incluir la definición de variables locales en la gramática, para evitar conflictos con la definición de las globales. Esta traducción la ampliaremos en puntos posteriores.

8. Ampliad la gramática para contemplar la definición múltiple de variables con asignaciones opcionales: **`int <id1> = 3, <id2>, …, <idk> = 1 ;`** que debe ser traducido a una secuencia de definiciones individuales

   `(setq <id1> 3) (setq <id2> 0) … (setq <idk> 1)` . Intentad que el orden de impresión de las variables corresponda al de la declaración.

Las variables locales se pueden declarar en C alternando con sentencias. También es posible declararlas con una inicialización en la que se asigna el valor de una expresión, al contrario que en el caso de las variables globales. Para comenzar proponemos declararlas únicamente al comienzo del bloque de código y restringir la asignación en la declaración a valores numéricos. Esto no es obligatorio, es una recomendación para simplificar la gramática y evitar problemas. Más adelante veremos la posibilidad de intercalar sentencias y declaración de variables.

9. Eliminad la producción que deriva una **`Sentencia → Expresion ;`** En su momento tenían su utilidad porque las expresiones se evaluaban de forma interactiva como en una calculadora. En C están permitidas las sentencias que constan sólo de una expresión, pero no les sacaremos partido ahora mismo.

10. Para imprimir cadenas literales proponemos: **`puts("Hola mundo") ;`** que se convierte en **`(print "Hola mundo")`**. Estudiad que **`yylex`** detecta las secuencias de entrada entre dobles comillas los envía como un *token* **`String`**.

11. Sustituid el símbolo **`$`** empleado para imprimir por la palabra reservada **`printf`**. El formato de la impresión se aumenta para contemplar la cadena de formato **`printf(<string>, <expr1>, … , <exprN>) ;`** traduciéndola a **`(print <expr1>) (print <expr2>), …, (print <exprN>)`**. Tomad nota que el contenido de la cadena de formato es complejo y requiere de un procesamiento muy elaborado para interpretar los formatos. Por ello debe ser reconocido por la gramática, pero no se traducirá de ninguna forma. Se omitirá en la traducción. Para imprimir cadenas literales usaremos **`puts`**.