Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-3725 - Traductores e Interpretadores
Abril - Julio 2018

Proyecto
Intérprete BasicTran

Los programadores de Fortran (nombre derivado de: “Formula Translation”) necesitan la colaboración de programadores para crear un nuevo lenguaje, que sea más sencillo. Este lenguaje, se llama BasicTran (nombre derivado de: “Basic Translation”). Se necesita crear su front-end y además desarrollar una implementación.

A continuación se describe el lenguaje, para el que Ud. debe crear un intérprete. Hay 4 etapas: i) Análisis lexicográfico; ii) análisis sintáctico iii) análisis de contexto iv) Intérprete del lenguaje y máquinas de turing.

1) Cómo se estructura un programa

```
[ with ⟨Lista de Declaraciones⟩ ] begin
      ⟨Instrucción⟩
end
```

`begin` y `end` son palabras reservadas que indican el principio y fin del programa. Los corchetes no son parte del programa, ellos sólo indican que su contenido es opcional. En este caso es la declaración de variables precedida de la palabra reservada `with`.

La ⟨*Lista de Declaraciones*⟩ indica las declaraciones de variables, junto a sus tipos respectivos.

La declaración de variables tiene la forma siguiente:

`var ⟨Lista de Identificadores⟩ : ⟨Tipo⟩`

En donde la ⟨`Lista de Identificadores`⟩ es una lista no vacía de identificadores de variables. Los identificadores están separados por comas (“,”) en la lista en cuestión. No se aceptan identificadores de variables que correspondan a palabras reservadas.

Cada variable tiene asociado un tipo, especificado por ⟨Tipo⟩ y cada variable puede almacenar un valor del tipo que tiene asociado y el cuerpo del programa podrá utilizarlo en cualquier momento.

El lenguaje manejará valores de tipo *entero* (representados por la palabra clave int), *booleano* (representados por la palabra clave bool), *caracteres* (representados por la palabra clase char) y *arreglos* (representadas por la palabra clave array). En el caso de los array, se debe expresar el tipo de sus elementos y su dimensión (> 0).

En general, un arreglo de tipo T, con dimensión n:

```
var m : array [n] of T
```

Las variables toman valores únicamente al momento de asignación (salvando variables asociadas a una instrucción de repetición determinada) e inicialmente no tienen valor. Se debe llegar a un error cuando se intente usar una variable que no haya sido inicializada.

A continuación tenemos dos variables enteras `a` y `b`, inicializando la variable `a` con el valor 42. Además, declara una variable booleana `c`, inicializada con el valor True:

```
var a <- 42, b : int
var c <- True : bool
```

2) Instrucciones

Todas las instrucciones deben terminar en (";"). Las instrucciones básicas son instrucciones que no están compuestas por otras instrucciones, tales como la asignación. Las instrucciones compuestas contienen otras instrucciones, como la secuenciación o repetición.

Las instrucciones:

Asignación: Son del tipo: "⟨Ident⟩ <- ⟨Expr⟩" Y se encarga de evaluar la expresión `Expr` y almacenar el resultado en la variable con el identificador `Ident`. Dicha variable debe estar previamente declarada; de lo contrario se tiene que dar un mensaje de error. Las variables dentro de la expresión también deben estar previamente declaradas; de lo contrario se dará un mensaje de error. Las expresiones deben tener el mismo tipo que la variable a asignar; de lo contrario deben reportar un error.

Secuenciación: La composición secuencial de instrucciones son la ejecución de dos instrucciones de forma continua. Por ejemplo, "⟨Instr0⟩ ⟨Instr1⟩" corresponde a ejecutar la instrucción 0 "⟨Instr0⟩ y luego la instrucción 1 ⟨Instr1⟩".

Note que “⟨Instr0⟩ ⟨Instr1⟩” es una instrucción compuesta. La secuenciacion permite combinar varias instrucciones en una sola que puede entonces ser, por ejemplo, la instrucción del cuerpo del programa principal.

Condicionales: Son nuestros famosos “if”

“if ⟨Bool⟩ -> ⟨Instr0⟩ [otherwise -> ⟨Instr1⟩] end”

Bool es una expresión booleana, los corchetes indican que su contenido es opcional. otherwise es una palabra reservada. Instr0 e Instr1 son instrucciones, Instr1 se ejecuta en caso de que la expresión booleana sea falsa; de ser verdadera, se ejecuta la Instr0.

Iteración Indeterminada: Las instrucciones de iteracion indeterminada (esto es, con condi- ciones generales de salida) son de la forma

“while ⟨Bool⟩ -> ⟨Instr⟩ end”

con ⟨Bool⟩ una expresion booleana e ⟨Instr⟩ una instrucción cualquiera.

La semántica para esta instruccion es la convencional: Se evalua la expresión ⟨Bool⟩; si es verdadera, se ejecuta el cuerpo ⟨Instr⟩ y se vuelve al inicio de la ejecución (preguntando nuevamente por la condicion anterior) o, en caso contrario, se abandona la ejecucion de la iteracion.

Por ejemplo:

```
with
        var n, f1 <- 1, f2 <- 0 : int

Begin
      read n;
      while n > 0 ->
            with
               var tmp : int
            begin
               tmp <- f2;
                f2 <- f1;
                f1 <- f1 + tmp;
                n <- n - 1;
            end
      end
      print f2;
end
```

Iteración Determinada: Las instrucciones de iteración determinada tienen una cantidad prefijada de iteraciones y son de esta forma:

“for ⟨Ident⟩ from ⟨AritmInf⟩ to ⟨AritmSup⟩ [step ⟨Aritm-Paso⟩]-> ⟨Instr⟩ end”

con ⟨Ident⟩ un identificador, ⟨AritmInf⟩ (límite inferior), ⟨AritmSup⟩ (límite superior), ⟨Instr⟩ es una instrucción cualquiera y *opcionalmente*, el paso del valor de ⟨Ident⟩ de iteración en iteración, que es 1 por defecto, o ⟨Aritm-Paso⟩ cuando se coloca explícitamente.

La ejecución de esta instrucción consiste en, inicialmente, evaluar las expresiones aritméticas ⟨AritmInf⟩ y ⟨AritmSup⟩ lo cual determina la cantidad de veces que a continuación se ejecuta el cuerpo ⟨Instr⟩.

En cada iteración, la variable que corresponde a ⟨Ident⟩ cumplirá la función de contador del ciclo obteniendo como valor.

Nótese que dentro de ⟨Instr⟩ estarán prohibidas asignaciones a la variable representada por ⟨Ident⟩. Esto, ya que si el valor de dicha variable pudiese modificarse dentro de ⟨Instr⟩, entonces la misma podría perder su rol como contador de la repetición original. Nótese también que la evaluación de ⟨Aritm-Paso⟩ debe ser distinta de cero (0); en caso contrario se dará un mensaje de error.

Si el identificador ⟨Ident⟩ ya corresponde a una variable declarada y visible desde el punto del programa en la que se encuentra la instrucción de repetición determinada, la variable externa queda temporalmente oculta durante el transcurso de la repetición. Esto es, se incorpora un alcance nuevo que contiene una nueva variable identificada por ⟨Ident⟩, de tipo int.

Incorporación de alcance: Una instrucción de incorporación de alcance en Neo tiene la siguiente estructura:

```
[ with ⟨Lista de Declaraciones⟩ ] begin
      ⟨Instrucción⟩
end
```

Esta es la misma estructura que la de un programa. Esta instrucción incorpora las nuevas declaraciones de variables (de existir) y las hace visibles/usables únicamente en la ⟨Instrucción⟩.

Entrada y Salida: Se cuenta con instrucciones que le permiten interactuar con un usuario a través de la entrada/salida estándar del sistema de operación (indistinto para muchos sistemas de operación conocidos). Para leer un valor de la entrada las instrucciones serán de la forma:

“read ⟨Ident⟩”

donde ⟨Ident⟩ es un identificador para una de las variables del programa. Esta variable puede ser solamente de tipo entero, booleano o carácter. La instrucción debe saber manejar la entrada en todos estos casos. Para escribir en la salida las instrucciones serán de la forma:

"print ⟨Expr⟩"

donde ⟨Expr⟩ es una expresión de cualquier tipo.

3) Expresiones

1) Literales

Los tipos básicos son: enteros, booleanos y caracteres. Los literales enteros son todos los números naturales, precedidos con una cantidad arbitraria de ceros. Los números negativos se construyen como expresiones compuestas. Los booleanos son `True` y `False`. Los literales para caracteres van entre comillas simples. Además, se deben soportar los caracteres especiales:

- `\n': Salto de linea.
- `\t': Tabulador horizontal.
- `\'': Comilla simple.
- `\\': Barra inversa.

2) Variables

Las variables pueden llevar por nombre cualquier cadena, tal que empiece con un carácter alfabético (mayuscula o minuscula) y el resto esté formado por caracteres alfanumericos (mayusculas, minusculas o digitos) o el caracter de underscore. El alcance de las variables estará definido por la instrucción de incorporación de alcance donde fue declarada, durante la totalidad de dicho bloque. Aunque solo puede existir una variable de un nombre determinado en cada alcance, es posible crear variables con nombres iguales en alcances interiores. En este caso, la visibilidad de la variable externa se esconde temporalmente durante la totalidad del bloque interno antes mencionado.

3) Expresiones Aritméticas

Una expresión aritmética estará formada por números naturales, identificadores de variables, y operadores convencionales de aritmética entera. Los operadores a ser considerados serán suma (+), resta (- binario), multiplicación (*), división entera (/), resto de división entera o módulo (%), e inverso (- unario). Tal como se acostumbra, las expresiones serán construidas con notación infija para los operadores binarios, por ejemplo `1+2`, y con notación prefija para el operador unario, por ejemplo `-3`. La tabla de precedencia es también la convencional (donde los operadores con mayor precedencia están hacia abajo):

```
+, - binario
*, /, %
- unario
```

y, por supuesto, se puede utilizar paréntesis para forzar un determinado orden de evaluación. Por tanto, evaluar `2+3/2` da como resultado `3`, mientras que evaluar `(2+3)/2` da `2`. Los operadores con igual precedencia se evalúan de izquierda a derecha. Por tanto, evaluar `60/2*3` da `90`, mientras que evaluar `60/(2*3)` da `10`.

Expresiones con variables serán evaluadas de acuerdo al valor que éstas tengan en ese momento, para lo cual se requiere que tales variables hayan sido declaradas y previamente inicializadas. Por ejemplo, la evaluación de `x+2` da `5`, si `x` fue declarada y en su última asignación tomó valor `3`. Si `x` no fue declarada o es de un tipo no compatible para la operación, se da un mensaje de error; a este tipo de errores se les llama estáticos, pues pueden ser detectados antes de la ejecución del programa. Si `x` fue declarada pero no ha sido inicializada previamente, también debe darse un mensaje de error; este error sería dinámico, pues sólo puede ser detectado durante la ejecución del programa.

4) Expresiones Booleanas

Análogamente a las expresiones aritméticas, una expresión booleana estará formada por las constantes `True` y `False`, identificadores de variables, y operadores convencionales de lógica booleana. Los operadores a ser considerados serán conjunción (“/\”), disyunción (“\/”), y negación (“not”).

Tal como se acostumbra, las expresiones serán construidas con notación infija para los operadores binarios, por ejemplo “`true /\ false`”. Sin embargo, el operador unario (negación) será construido con notación prefija, por ejemplo “not true”. La tabla de precedencia es también la convencional (donde las operadores con mayor precedencia están hacia abajo):

```
\/
/\
Not
```

y, por supuesto, se puede utilizar paréntesis para forzar un determinado orden de evaluación. Por tanto, evaluar `true \/ true /\ false` resulta en `true`, mientras que la evaluación de `(true \/ true) /\ false` resulta en `false`. Los operadores con igual precedencia se evalúan de izquierda a derecha. Sin embargo, note que en este caso, en realidad dicho orden es irrelevante.

Expresiones con variables serán evaluadas de acuerdo al valor que éstas tengan en ese mo- mento, de manera análoga a las expresiones aritméticas.

5) Expresiones de Caracteres

Una expresión de caracteres estara formada por los caracteres literales, identificadores de variables, y los operadores de caracter siguiente (“++”), caracter anterior (“−−”), y valor

ASCII (“#”). Las expresiones seran construidas con notación sufija para los operadores de caracter siguiente y caracter anterior, por ejemplo 'c'++. Sin embargo, el operador de valor ASCII será construido con notación prefija, por ejemplo #'g'.

La tabla de precedencia es la siguiente (donde los operadores con mayor precedencia están hacia abajo):

```
++,
--,
#
```

Las expresiones de caracter siguiente y anterior se basan en el orden de la codificación ASCII y ademas seran ciclicos. Esto es, si se pide el caracter siguiente del caracter 127, se pasará a tener el caracter 0 y viceversa.

Expresiones con variables serán evaluadas de acuerdo al valor que éstas tengan en ese momento, de manera análoga a las expresiones anteriores.

6) Expresiones con Arreglos

Una expresión sobre arreglos estará formada por identificadores de variables, y los operadores de concatenación (“`::`”), shift (“`$`”) e indexación (“`[]`”). Las expresiones serán construidas con notación infija para los operadores binarios; por ejemplo, la concatenación de los arreglos a y b se escribiría “`a :: b`”. El operador unario `shift` será construido con notación infija, por ejemplo “`$a`”.

La tabla de precedencia es la siguiente (los operadores con mayor precedencia están hacia abajo):

```
::
$
[]
```

y, por supuesto, se puede utilizar paréntesis para forzar un determinado orden de evaluación. Los operadores con igual precedencia se evalúan de izquierda a derecha.

El operador de indexación provee la forma de acceder y modificar los elementos individuales de un arreglo. Se debe proveer entre los corchetes un valor. Cada valor debe ser no-negativo y menor o igual que el tamaño de la dimensión que le corresponde. En caso contrario, se debe reportar un error. Es posible usar el operador de indexación sobre un arreglo del lado izquierdo de una asignación. Esto es, el operador de indexación tendrá dos comportamientos dependiendo del contexto: devolverá valor almacenado o dirección de memoria correspondiente.

Por ejemplo si “a” es un arreglo de longitud 5, entonces la expresión “a[3]” se puede interpretar de dos formas:

- Si aparece del lado izquierdo de una asignación, será la cuarta posición del arreglo (posición en donde va a ser asignado el valor de la expresión del lado derecho de la asignación).
- De lo contrario, será el entero almacenado en el arreglo en la cuarta posición.

El operador de concatenación de arreglos toman dos arreglos y produce un nuevo arreglo, que es la concatenación de los mismos. Para que dicha concatenación sea válida, se deben cumplir que ambos arreglos tengan el mismo tipo. Por ejemplo si “a” y “b” son arreglos de enteros de longitudes 3 y 2 respectivamente tales que a[0]=1, a[1]=2, a[2]=3 y b[0]=7, b[1]=5, entonces el arreglo “a::b” tiene longitud 5 y satisface que a::b[0]=1, a::b[1]=2, a::b[2]=3, a::b[3]=7 y a::b[4]=5

La operación shift, toma un arreglo y mueve todos los valores de sus casillas hacia la siguiente casilla, salvo la ultima que su valor se mueve a la primera casilla del arreglo. Por ejemplo si “a” es un arreglo de enteros de longitud 3, tal que a[0]=2, a[1]=3 y a[2]=4, entonces el arreglo “$a” satisface que $a[0]=4, $a[1]=2 y $a[2]=3.

7) Expresiones Relacionales

Las expresiones relacionales consisten en aquellas expresiones en las que se comparan dos expresiones aritméticas con una relación. Estas serán de la forma “⟨Aritm⟩⟨Rel⟩⟨Aritm⟩”, en donde “⟨Aritm⟩” son expresiones aritméticas y “⟨Rel⟩” es un operador relacional. Los operadores relacionales a considerar son: menor (“<”), menor o igual (“<=”), mayor (“>”), mayor o igual (“>=”), igualdad (“=”) y desigualdad (“/=”).

Expresiones con variables serán evaluadas de acuerdo al valor que éstas tengan en ese momento, de manera análoga a las expresiones anteriores.