Diseño de Compiladores I - 2017
Trabajo Práctico Nº 2
Fecha de entrega: 25/09/2017

Objetivo

Construir un parser que invoque al Analizador Léxico creado en el Trabajo Práctico Nº 1, y que reconozca un lenguaje que incluya:

Programa:

- Programa constituido por un conjunto de sentencias, que pueden ser declarativas o ejecutables.
  Las sentencias declarativas pueden aparecer en cualquier lugar del código fuente, exceptuando los bloques de las sentencias de control.
- El programa no tendrá ningún delimitador.
- Cada sentencia debe terminar con " **.** ".

Sentencias declarativas:

- Sentencias de declaración de datos para los tipos de datos correspondientes a cada grupo según la consigna del Trabajo Práctico 1, con la siguiente sintaxis:
  <lista_de_variables> **:** <tipo> **.**
  Donde <tipo> puede ser (Según tipos correspondientes a cada grupo):
  INT
  UINT
  LONG
  ULONG
  FLOAT
  DOUBLE
  Las variables de la lista se separan con coma ("**,**")

Sentencias ejecutables:

- Cláusula de selección (**IF**). Cada rama de la selección será un bloque de sentencias. La condición será una comparación entre expresiones aritméticas, variables o constantes, y debe escribirse entre **( )**. La estructura de la selección será, entonces:
  IF (<condicion>**) THEN** <bloque_de_sentencias> **ELSE** <bloque_de_sentencias> **END_IF**
  El bloque para el **ELSE** puede estar ausente.
- Un bloque de sentencias puede estar constituido por una sola sentencia, o un conjunto de sentencias delimitadas por **BEGIN** y **END.**
- Sentencia de control según tipo especial asignado al grupo. (Temas 7 al 10 del Trabajo práctico 1)
  Debe permitirse anidamiento de sentencias de control. Por ejemplo, puede haber una iteración dentro de una rama de una selección.
- Sentencia de salida de mensajes por pantalla. El formato será **OUT**(cadena). Las cadenas de caracteres sólo podrán ser usadas en esta sentencia, y tendrán el formato asignado al grupo en el Trabajo Práctico 1.
- Los operandos de las expresiones aritméticas pueden ser variables, constantes, u otras expresiones aritméticas.
  No se permiten anidamientos de expresiones con paréntesis.

Temas particulares

Sentencias de Control

- ***WHILE DO*** (tema 7 en TP1)
  WHILE (<condicion> **) DO** <bloque_de_sentencias> **.**
- ***DO UNTIL*** (tema 8 en TP1)
  DO <bloque_de_sentencias> **UNTIL (** <condicion> **) .**
- ***FROM TO BY*** (tema 9 en TP1)
  FROM i **:=** n **TO** m **BY** j < bloque_de_sentencias > **.**
  i debe ser una variable de tipo (1-2-3).
  m, n y j pueden ser variables, constantes o expresiones aritméticas de tipo (1-2-3).
  (1) Para los grupos que tengan asignados 2 tipos enteros, considerar el "más chico" (por ej. INT, para grupos con INT y LONG)
  (2) Para los grupos que tengan asignados 1 tipo entero, y uno de punto flotante, considerar el tipo entero
  (3) Para los grupos que tengan asignados 2 tipos de punto flotante, considerar el "más chico" (por ej. FLOAT, para grupos con FLOAT y DOUBLE)
  Nota: Las restricciones de tipo serán chequeadas en la etapa 3 del trabajo práctico.
- ***SWITCH CASE*** (tema 10 en TP1) Incorporar las llaves (**{** y **}**) como caracteres válidos.
  SWITCH (variable){
  CASE valor1**:** <bloque> **.**
  CASE valor2**:** <bloque> **.**
  ...
  CASE valorN**:** <bloque> **.**
  } .
  Nota**:** Los valores valor1, valor2, etc., sólo podrán ser constantes del mismo tipo que la variable.
  La restricción de tipo será chequeada en la etapa 3 del trabajo práctico.

Temas 11 y 13

- (Tema 11 del TP1)
  Se deberán incorporar, como posibles sentencias ejecutables, asignaciones del tipo:
  LET <asignación> **.**
  Nota: La semántica de estas sentencias, se explicará y resolverá en el trabajo práctico 3..
- (Tema 13 del TP1)
  Entre las sentencias declarativas del programa, se podrá incluir la declaración de funciones, con el siguiente formato:
  <tipo> **FUNCTION** <nombre_de_funcion> o <tipo> **MOVE FUNCTION** <nombre_de_funcion>
  {
  <Sentencias declarativas y ejecutables de la función>
  RETURN(<expresión>) **.**
  }
  Incorporar las llaves (**{** y **}**) como caracteres válidos.
  Notas:
  - No se permite anidamiento de funciones.
  - Una función se invoca utilizando su nombre seguido de paréntesis: <nombre_de_función>**(),** y una invocación puede aparecer en cualquier lugar donde pueda aparecer una variable o constante
  - La semántica de la declaración usando **MOVE**, se explicará y resolverá en el trabajo práctico 3..

Conversiones

- **Explícitas** (Temas 14 y 15 del TP1)
  En cualquier lugar donde pueda aparecer una variable o una expresión, podrá aparecer:
  <conversión>(<expresión>)
  Donde <conversión> será una palabra reservada, que se deberá incorporar a las anteriores, según nro de grupo:

Grupo	Palabra reservada
1	**I_F**
2	**F_UI**
3	**F_L**
4	**UI_UL**
5	**D_UL**
6	**D_I**
7	**L_D**
8	**L_D**
9	**UI_F**
10	
11	**F _UL**

Grupo	Palabra reservada
12	
13	**I_L**
14	
15	**UL_F**
16	
17	**I_D**
18	
19	**UL_F**
20	
21	
22	**L_I**
23	**F_I**

  La semántica de estas conversiones se explicará y resolverá en el trabajo práctico 3.
- **Sin conversiones** (Tema 16 del TP1)
  Se definirá en el Trabajo Práctico 3.

Salida del Compilador

El programa deberá leer un código fuente escrito en el lenguaje descripto, y deberá generar como salida:

- Tokens detectados por el Analizador Léxico
- Estructuras sintácticas detectadas en el código fuente. Por ejemplo:
  Asignación
  Sentencia **WHILE**
  Sentencia **IF**
  etc.
  (Indicando nro. de línea para cada estructura)
- Errores léxicos y sintácticos presentes en el código fuente, indicando: nro. de línea y descripción del error. Por ejemplo:
  Línea 24: Constante de tipo UINT fuera del rango permitido.
  Línea 43: Falta paréntesis de cierre para la condición del IF.
- Tabla de símbolos

Consignas

a) Utilizar YACC u otra herramienta similar para construir el parser.

b) Adaptar el Analizador Léxico del Trabajo Práctico 1 para convertirlo en el método o rutina ***int yylex()*** (o el nombre que el Parser generado requiera). Tener en cuenta que el léxico deberá devolver al parser, en cada invocación, un token. Para los identificadores, constantes y cadenas, deberá devolver además, la referencia a la entrada de la Tabla de Símbolos donde se ha registrado dicho símbolo.

c) Para aquellos tipos de datos que permitan valores negativos (**INT**, **LONG**, **FLOAT** y **DOUBLE**) deberán detectar **constantes negativas**, modificando la tabla de símbolos según corresponda. Será necesario volver a controlar el rango de las constantes, ya que un valor aceptado para una constante por el Analizador Léxico, que desconoce su signo, puede estar fuera de rango si la constante es positiva.

- Ejemplo 1: Las constantes de tipo **INT** pueden tomar valores desde –32768 a 32767. El Léxico aceptará la constante 32768 como válida, pero si se trata de una constante positiva, estará fuera de rango.

d) Cuando se detecte un error, la compilación debe continuar.
e) Conflictos: Eliminar **todos** los conflictos shift-reduce y reduce-reduce que se presenten al generar el parser.

Informe

Se debe presentar un informe que incluya:

- Contenidos indicados en el enunciado del Trabajo Práctico 1
- Descripción del proceso de desarrollo del Analizador Sintáctico: problemas surgidos (y soluciones adoptadas) en el proceso de construcción de la gramática, manejo de errores, solución de conflictos shift-reduce y reduce-reduce, etc.
- Lista de no terminales usados en la gramática con una breve descripción para cada uno.
- Lista de errores léxicos y sintácticos considerados por el compilador.
- Conclusiones.

Forma de entrega

Se deberá presentar:

a) Código fuente completo y ejecutable, **incluyendo librerías del lenguaje** si fuera necesario para la ejecución
b) Informe
c) Caso de prueba que contemple **todas** las estructuras válidas del lenguaje