Diseño de Compiladores I - 2020
Trabajo Práctico Nº 2
Fecha de entrega: 30/09/2020

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:
  <tipo> <lista_de_variables>**;**
  Donde <tipo> puede ser (Según tipos correspondientes a cada grupo):
  INTEGER
  UINT
  LONGINT
  ULONGINT
  FLOAT
  DOUBLE
  Las variables de la lista se separan con coma (",")
- Incluir declaraciones de procedimientos, con la siguiente sintaxis:

```
PROC ID (<lista_de_parametros>) {
   <cuerpo_del_procedimiento>   // conjunto de sentencias declarativas y ejecutables
                                // permitir procedimientos anidados
}
```

  Donde:
  - <lista_de_parametros> será una lista de parámetros, separados por ",", con la siguiente estructura para cada parámetro:
    - <tipo> ID
    - El número máximo de parámetros permitidos es 3, y puede no haber parámetros. **Este chequeo debe efectuarse durante el Análisis Sintáctico**

  // Incorporar **PROC** a la lista de palabras reservadas.

Sentencias ejecutables:

- Asignaciones donde el lado izquierdo es un identificador, y el lado derecho una expresión aritmética.
  Los operandos de las expresiones aritméticas pueden ser variables, constantes, u otras expresiones aritméticas.
  No se deben permitir anidamientos de expresiones con paréntesis.
- 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>) <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 **'{'** y **'}'.**
- Sentencia de control según tipo especial asignado al grupo. (Temas 12 A 14 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 (temas 20 y 21).
- Incorporar como sentencia ejecutable, invocaciones a procedimientos con la siguiente estructura:

  ID(<parametros>)

  Donde <parsmetros> será una lista de identificadores separados por ",".

  Nota: La semántica de la declaración e invocación de procedimientos, se explicará y resolverá en los trabajos prácticos 3 y 4.

Temas particulares

Temas 7 a 11

La semántica de los siguientes temas se explicará y resolverá en los trabajos prácticos 3 y 4.
Los chequeos de tipos y todo otro chequeo semántico se efectuarán en el trabajo práctico 3

- Tema 7 en TP1: Control de invocaciones recibidas - Pasaje de parámetros por copia valor resultado

Sentencias declarativas:

- Modificar los encabezados de declaraciones de funciones con la siguiente estructura:

  PROC ID (<lista_de_parametros>) **NI** = n { … }

  Donde n será una constante de tipo entero (temas 1-2-3-4)
- Modificar la estructura de cada parámetro, permitiendo usar la palabra reservada **VAR**, delante de cada parámetro.
- Ejemplos de declaraciones de procedimiento válidas:

```
PROC f1(INTEGER x, VAR FLOAT y, FLOAT z) NI = 2 {…}
PROC f2(VAR DOUBLE a, VAR DOUBLE b) NI = 4 {…}
```

// Incorporar **NI** y **VAR** a la lista de palabras reservadas.

Sentencias ejecutables:

- Sin cambios

- Tema 8 en TP1: Control de invocaciones efectuadas – Pasaje de parámetros por referencia

Sentencias declarativas:

- Modificar los encabezados de declaraciones de procedimientos con la siguiente estructura:

  PROC ID (<lista_de_parametros>) **NI=**n { … }

  Donde n será una constante de tipo entero (temas 1-2-3-4)
- Modificar la estructura de cada parámetro, permitiendo usar la palabra reservada **REF**, delante de cada parámetro.
- Ejemplos de declaraciones de procedimiento válidas:

```
PROC f1(INTEGER x, REF FLOAT y, FLOAT z) NI = 2 {…}
PROC f2(REF DOUBLE a, REF DOUBLE b) NI = 4 {…}
```

// Incorporar **NI** y **REF** a la lista de palabras reservadas.

Sentencias ejecutables:

- Modificar las invocaciones a procedimientos, incorporando notación explícita (indicando los nombres de los parámetros formales) para los parámetros:
- La invocación a una procedimiento será:

  ID(<parametros>)

  Donde cada parámetro de la lista tendrá la siguiente estructura:

  ID:ID
- Ejemplos de invocaciones a procedimiento válidas:

```
f1(z:i, y:j, x:k)
f2(a:m, b:n)
```

// Incorporar **":"** a la lista de tokens aceptados por el Analizador Léxico.

- Tema 9 en TP1: Control de niveles de anidamiento - Control de niveles para el alcance de las variables

Sentencias declarativas:

- Modificar los encabezados de declaraciones de procedimientos con la siguiente estructura:

  PROC ID (<lista_de_parametros>) **NA**=m, **NS=**n { … }

  Donde m y n serán constantes de tipo entero (temas 1-2-3-4)
- Ejemplos de declaraciones de procedimiento válidas:

```
PROC f1(INTEGER x, FLOAT y, FLOAT z) NA = 2, NS = 1 {…}
```

```
PROC f2(DOUBLE a, DOUBLE b) NA = 1, NS = 0 {…}
```

// Incorporar **NA** y **NS** a la lista de palabras reservadas.

Sentencias ejecutables:

- Sin cambios.

- Tema 10 en TP1: Control de niveles de anidamiento permitidos - Ámbito prefijado al usar una variable.

Sentencias declarativas:

- Modificar los encabezados de declaraciones de procedimientos con la siguiente estructura:

  PROC ID (<lista_de_parametros>) **NA=**m { … }

  Donde m será una constante de tipo entero (temas 1-2-3-4)
- Ejemplos de declaraciones de procedimiento válidas:

```
PROC f1(INTEGER x, FLOAT y, FLOAT z) NA = 2 {…}
PROC f2(DOUBLE a, DOUBLE b) NA = 1 {…}
```

// Incorporar **NA** a la lista de palabras reservadas.

Sentencias ejecutables:

- Los identificadores utilizados en cualquier lugar de una sentencia ejecutable, pueden ser antecedidos por otro identificador seguido de "::".
- Ejemplos válidos:

```
x = f1::y + f2::z;
f2::z = w + 3 * f2::k;
```

// Incorporar **"::"** a la lista de tokens aceptados por el Analizador Léxico.

- Tema 11 en TP1: Control de niveles de anidamiento permitidos - Shadowing

Sentencias declarativas:

- Modificar los encabezados de declaraciones de procedimientos con la siguiente estructura:

  PROC ID (<lista_de_parametros>) **NA**=m, **SHADOWING**=<true_false> { … }

  Donde m será una constante de tipo entero (temas 1-2-3-4)
  y <true_false> puede ser **TRUE** o **FALSE**
- Ejemplos de declaraciones de procedimiento válidas:

```
PROC f1(INTEGER x, FLOAT y, FLOAT z) NA = 2, SHADOWING = TRUE {…}
PROC f2(DOUBLE a, DOUBLE b) NA = 1, SHADOWING = FALSE {…}
```

// Incorporar **NA**, **SHADOWING, TRUE y FALSE** a la lista de palabras reservadas.

Sentencias ejecutables:

- Sin cambios

Temas 12 a 14: Sentencias de Control

- ***WHILE LOOP*** (tema 12 en TP1)

  WHILE (<condicion>) **LOOP** <bloque_de_sentencias>
- ***LOOP UNTIL*** (tema 13 en TP1)

  LOOP <bloque_de_sentencias> **UNTIL** (<condicion>)
- ***FOR*** (tema 14 en TP1)

  FOR (i = n; <condición>; <incr_decr> j) < bloque_de_sentencias >

  i debe ser una variable de tipo entero (1-2-3-4).
  n y j serán constantes de tipo entero (1-2-3-4).
  <condición> será una comparación de i con m. Por ejemplo: i < m
  Donde m puede ser una variable, constante o expresión aritmética de tipo entero (1-2-3-4).
  <incr_decr> puede ser **UP** o **DOWN**

// Incorporar **UP** y **DOWN** a la lista de palabras reservadas.
Nota: Las restricciones de tipo serán chequeadas en la etapa 3 del trabajo práctico.

Temas 15 a 17: Conversiones

- Tema 15: **Sin conversiones:** Se explicará y resolverá en trabajos prácticos 3/4. (Grupos 2, 3, 6, 10, 11, 12, 15, 17,18, 19)
- Tema 16: **Conversiones Explícitas**: . Se debe incorporar en todo lugar donde pueda aparecer una expresión, la siguiente sintaxis:

  <tipo>(<expresión>)

  donde <tipo> será:

Grupo	tipo
7	DOUBLE
8	DOUBLE
9	FLOAT
13	DOUBLE
20	DOUBLE
21	DOUBLE
22	FLOAT

- Tema 17: **Conversiones Implícitas**: Se explicará y resolverá en trabajos prácticos 3/4. (Grupos 1, 4, 5, 14, 16)

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 **LOOP UNTIL**
  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 **INTEGER** fuera del rango permitido.
  Línea 43: Falta paréntesis de cierre para la condición de la sentencia **IF**.
- Contenidos de la 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 función ***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, utilizando **yylval** para hacerlo.

c) Para aquellos tipos de datos que permitan valores negativos (**INTEGER**, **LONGINT**, **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: Las constantes de tipo **INTEGER** 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.

Forma de entrega

Se deberá presentar:

a) Código fuente completo y ejecutable, **incluyendo librerías del lenguaje** si fuera necesario para la ejecución

b) Informe
- 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.

c) Caso de prueba que contemple **todas** las estructuras válidas del lenguaje