Diseño de Compiladores I - 2019
Trabajo Práctico Nº 2
Fecha de entrega: 16/10/2019

Objetivo

Construir un parser que invoque al Analizador Léxico creado en el Trabajo Práctico Nº 1, y que reconozca un lenguaje que incluya las estructuras que se indican a continuación:

Programa:

- Programa constituido por un conjunto de sentencias, que pueden ser declarativas o ejecutables.
  Las sentencias declarativas estarán ubicadas al comienzo del código fuente.
- Las sentencias ejecutables, ubicadas a continuación de las declaraciones, estarán delimitadas por **begin** y **end**.
- 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>; **//incluyendo las declaraciones indicadas en temas especiales 9/10/11/12**
  Donde <tipo> puede ser (Según tipos correspondientes a cada grupo):
  int
  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>**)**<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 tema especial asignado al grupo. (Temas 5 al 8 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 **print**(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.

Notas generales:

- Si fuera necesario, incorporar al Análisis Léxico el reconocimiento de símbolos / palabras reservadas no considerados en el Trabajo Práctico 1
- La semántica de todas las sentencias que el parser debe reconocer, se explicará y resolverá en los trabajos prácticos 3 y 4.
- Todas las comprobaciones relacionadas con la semántica del lenguaje, serán efectuados en las etapas 3 y 4. Ejemplos de dichas comprobaciones son: chequeo de tipos, control de declaración / redeclaración de variables, chequeo de subíndices dentro del rango cuando corresponda, etc.

Temas particulares

Temas 5 a 8: Sentencias de Control

- Tema 5 en TP1: ***while do*** ()
  while (<condicion> **) do** <bloque_de_sentencias>**;**
- Tema 6 en TP1: ***do until***
  do<bloque_de_sentencias>**until (**<condicion>**);**
- Tema 7 en TP1:
  for(i **:=** n; j; m)<bloque_de_sentencias>**;**
  i debe ser una variable de tipo **int**
  n, j y m pueden ser variables, constantes o expresiones aritméticas de tipo **int**.
- Tema 8 en TP1: ***Iterador para colecciones***
  foreach<nombre_variable> **in** <nombre_coleccion><bloque_de_sentencias> **;**

Temas 9 a 12

La semántica de los siguientes temas se explicará y resolverá en los trabajos prácticos 3 y 4.

- Tema 9 en TP1: ***Colecciones con inferencia***

Sentencias declarativas:

- Incluir declaración de colecciones, con alguna de las siguientes opciones de sintaxis:
  1. <tipo> <nombre_coleccion>[<tamaño>] ;
     donde<tamaño> debe ser una constante entera, y <tipo> puede ser cualquiera de los dos tipos de datos asignados al grupo.
     Por ejemplo: int A[4];
  2. <tipo> <nombre_coleccion>[<lista_de_valores_iniciales>] ;
     donde<lista_de_valores_iniciales>podrá ser:
     - una lista de constantes separadas por ‘,’
       Por ejemplo: int A**[**1**,**2**,**3**,**4**]** ; o float x**[**1.2**,** 0.0**]** ;
     - una lista de ‘_’ separados por comas ‘,’.
       Por ejemplo: int A[_ , _ , _ , _] ;
     - o una combinación de los anteriores.
       Por ejemplo: int A[_ , 2 , 5 , _] ;

Sentencias ejecutables:

- Incorporar a la sintaxis del lenguaje, la posibilidad de usar referencias a elementos de colecciones, en el lado izquierdo de asignaciones, o en cualquier lugar donde se puedan usar expresiones aritméticas. La sintaxis para referirse a un elemento de una colección será:
  <nombre_coleccion>[<subíndice>]
  Donde <subíndice> sólo podrá ser un identificador o constante entera.

  Ejemplos:
  A[b]:= j*c; x[3]:=A[0] + w;

- Tema 10 en TP1: Colecciones con métodos predefinidos y rowing

Sentencias declarativas:

- Incluir declaración de colecciones, con la sintaxis:
  <tipo> <nombre_coleccion>**[**<tamaño>**] ;**
  donde<tamaño> debe ser una constante entera, y <tipo> puede ser cualquiera de los dos tipos de datos asignados al grupo.
  Por ejemplo: int A[4];

Sentencias ejecutables:

- Incorporar a la sintaxis del lenguaje:
  1. Uso de referencias a elementos de colecciones, en el lado izquierdo de asignaciones, o en cualquier lugar donde se puedan usar expresiones aritméticas. La sintaxis para referirse a un elemento de una colección será:
     <nombre_coleccion>[<subíndice>]
     Donde <subíndice> sólo podrá ser un identificador o constante entera.

     Ejemplos:
     A[b]:= j*c; x[3]:=A[0] + w;
  2. Invocación a métodos predefinidos para las colecciones. Estos métodos serán **first**, **last** y **lenght**. Y se invocarán con la siguiente sintaxis:
     <nombre_coleccion>.<nombre_metodo>();
     Por ejemplo:
     z := A.first(); w := x.lenght() * 2;
     La invocación de estos métodos se podrá efectuar en cualquier lugar donde pueda aparecer una expresión aritmética.
  3. Uso del nombre de una colección, del lado izquierdo de una asignación.
     Por ejemplo:
     A := 25; x := j; A := w * z; // A y x son nombres de colecciones

- Tema 11 en TP1: Objetos con herencia simple y control de visibilidad

Sentencias declarativas:

1. Incluir declaración de clases, con la siguiente sintaxis:

```
class <nombre_clase>
begin
  <declaraciones>
end
```

Donde declaraciones pueden ser declaraciones de atributos o métodos

- Las declaraciones de atributos serán sentencias declarativas como las indicadas en la sección general, precedidas por las palabras reservadas **public** o **private**
- Las declaraciones de métodos tendrán la siguiente sintaxis:
  (**public** o **private**) **void**<nombre_metodo>**()begin**<cuerpo_metodo>**end**

En el encabezado de la declaración de una clase, considerar también la siguiente sintaxis:

class <nombre_clase> **extends** <nombre_clase>

2. Incluir declaración de objetos de clases declaradas, con la siguiente sintaxis:

<nombre_clase> <lista_nombres_objetos>**;**

Sentencias ejecutables:

- Incorporar la referencia a los atributos de las clases con la sintaxis:
  <nombre_objeto>.<nombre_atributo>
  Estas referencias pueden aparecer en cualquier lugar donde pueda usarse un identificador o constante. Por ejemplo:
  o1.x **:=** 25 **;**
  o2.y **:=** o1.x + j **;**
- Incorporar la invocación de métodos, con la siguiente sintaxis:
  <nombre_objeto>**.**<nombre_metodo>**();**
  La invocación puede aparecer en cualquier lugar donde pueda aparecer una expresión aritmética.

- Tema 12 en TP1: Objetos con herencia múltiple

Sentencias declarativas:

1. Incluir declaración de clases, con la siguiente sintaxis:

class <nombre_clase>
begin
<declaraciones>
end
Donde declaraciones pueden ser declaraciones de atributos o métodos

- Las declaraciones de atributos serán sentencias declarativas como las indicadas en la sección general.
- Las declaraciones de métodos tendrán la siguiente sintaxis:
  void <nombre_metodo>**() begin** <cuerpo_metodo> **end**

En el encabezado de la declaración de una clase, considerar también la siguiente sintaxis:

class <nombre_clase> **extends** <lista_de_nombres_de_clase>

2. Incluir declaración de objetos de clases declaradas, con la siguiente sintaxis:

<nombre_clase> <lista_nombres_objetos>**;**

Sentencias ejecutables:

- Incorporar la referencia a los atributos de las clases con la sintaxis:
  <nombre_objeto>.<nombre_atributo>
  Estas referencias pueden aparecer en cualquier lugar donde pueda usarse un identificador o constante. Por ejemplo:
  o1.x **:=** 25 **;**
  o2.y **:=** o1.x + j **;**
- Incorporar la invocación de métodos, con la siguiente sintaxis:
  <nombre_objeto>**.**<nombre_metodo>**();**
  La invocación puede aparecer en cualquier lugar donde pueda aparecer una expresión aritmética.

Temas 13 a 15: Conversiones

La semántica de conversiones se explicará y resolverá en los trabajos prácticos 3 y 4.

- Tema 13: **Sin conversiones.**
- Tema 14: **Conversiones Explícitas**:
  Se debe incorporar, en todo lugar donde pueda aparecer una expresión, la siguiente sintaxis:
  <conversion>(<expresión>) Donde <conversión> será:

Grupos	conversion
2 y 19	to_ulong
10 y 13	to_long

- Tema 15: **Conversiones Implícitas**: Se explicará y resolverá en trabajos prácticos 3/4.

Salidas 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 **int** fuera del rango permitido.
  Línea 43: Falta paréntesis de cierre para la condición del **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.

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: 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.

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 y soluciones 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) Casos de prueba que contemplen **todas** las estructuras válidas del lenguaje, así como errores sintácticos en cada una de ellas.