Diseño de Compiladores I - 2024
Trabajo Práctico Nº 2
Fecha de entrega: 07-10-2024

OBJETIVO

Construir un Parser (Analizador Sintáctico) que invoque al Analizador Léxico creado en el Trabajo Práctico Nº 1, y que reconozca un lenguaje con las siguientes características:

SINTAXIS GENERAL:

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.
  Los elementos declarados sólo serán visibles a partir de su declaración (esto será chequeado en etapas posteriores).
- Cada sentencia debe terminar con punto y coma "**;**".
- El programa comenzará con un nombre, seguido por un conjunto de sentencias delimitado por **BEGIN** y **END**.
  Ejemplo de programa:

```
prog_prueba
begin
   …. // cuerpo del programa
end
```

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, uinteger, longint, ulongint, single, double.**
  Las variables de la lista se separan con coma ("**,**")

- Incluir declaración de funciones, con la siguiente sintaxis:

  <tipo> **FUN** ID (<parametro>)
  begin
  <cuerpo_ de_la_funcion>
  end

  Donde:
  - <parametro> será un identificador precedido por un tipo.:
    <tipo> ID
    La presencia del parámetro es obligatoria. **Este chequeo debe efectuarse durante el Análisis Sintáctico**
  - <cuerpo_de_la_funcion> es un conjunto de sentencias declarativas (incluyendo declaración de otras funciones) y/o ejecutables, incluyendo sentencias de retorno con la siguiente estructura:
    RET (<expresión>);

  Ejemplos válidos:

```
Integer fun f1 (integer y)
begin
       integer x ;
       x := y;
       …
       Ret (x);
end
```

```
Single fun f3 (longint w)
begin
       single x;
       x := 1.2;
       …
       IF (x > 0.0)then
              RET (x + 2);
       ELSE
          begin
              x := 2.0;
              RET (x * 2);
```

```
                end
        END_IF;
end
```

Sentencias ejecutables:

- Asignaciones donde el lado izquierdo puede ser un identificador, y el lado derecho una expresión aritmética. Los operandos de las expresiones aritméticas pueden ser variables, constantes, invocaciones a función u otras expresiones aritméticas.
  No se deben permitir anidamientos de expresiones con paréntesis.
  - Las invocaciones a función tendrán el siguiente formato:

    ID(<parametro_real>)

  El parámetro real puede ser cualquier expresión aritmética, variable o constante.

  Ejemplos de asignaciones:

```
a := b * 2;        z := f1(a) + j;                w := z / f3(a+b) + f4(5);
```

- Cláusula de selección (**IF**). Cada rama de la selección será un bloque de sentencias. La estructura de la selección será, entonces:

  IF (<condicion>) **THEN** <bloque_de_sent_ejecutables> **ELSE** <bloque_de_sent_ejecutables> **END_IF ;**

  El bloque para el **ELSE** puede estar ausente.
  La condición será una comparación entre expresiones aritméticas, variables, invocaciones a función o constantes, y debe escribirse entre **"(" ")"**.
  El bloque de sentencias ejecutables puede estar constituido por una sola sentencia, o un conjunto de sentencias ejecutables delimitadas por **begin end**.

- Sentencia de salida de mensajes por pantalla. El formato será

  OUTF(<cadena>);

  o

  OUTF(<expresion>);

  Ejemplos:

```
OUTF({Hola mundo});              //Tema 9
OUTF(  [Hola                     //Tema 10
       Mundo]);
OUTF(a+b);                       //Todos los temas
```

TEMAS PARTICULARES

Nota: La semántica de cada tema particular, se explicará y resolverá en las etapas 3 y 4 del trabajo práctico

Temas 11 y 12 -

- **Tema 11**: **Subtipos**

  Se debe incorporar la posibilidad de declarar tipos como subrangos de los tipos básicos aceptados por el lenguaje. En la declaración se incluirá el nombre del nuevo tipo, el tipo en el que se basa, y el rango de valores aceptado por ese tipo
  Por ejemplo:

```
typedef enterito := integer[-10,10]; //declara un tipo subrango de enteros
typedef flotantito := single[-1.2,3.5]; //declara un tipo subrango de single
```

  Se podrán declarar variables de los tipos definidos, que se utilizarán del mismo modo que cualquier variable. Ejemplos:

```
enterito a, b, c;
flotantito x, y, z;
a := 1;
z := x + y;
```

- **Tema 12**: **Tipos Embebidos**

  La semántica de este tema se explicará en el Trabajo Práctico 3

Temas 13 a 16: Sentencias de Control

- **Tema 13**: ***WHILE***

```
WHILE ( <condicion> ) <bloque_de_sentencias_ejecutables> ;
```

<condición> tendrá la misma definición que la condición de las sentencias de selección**.**

<bloque_de_sentencias_ejecutables> podrá contener una sentencia, o un grupo de sentencias ejecutables delimitadas por **begin end**.

- **Tema 14**: ***REPEAT UNTIL***

```
REPEAT <bloque_de_sentencias_ejecutables> UNTIL ( <condicion> );
```

<condición> tendrá la misma definición que la condición de las sentencias de selección**.**

<bloque_de_sentencias_ejecutables> podrá contener una sentencia, o un grupo de sentencias ejecutables delimitadas por **begin end**.

- **Tema 15**: ***REPEAT WHILE***

```
REPEAT <bloque_de_sentencias_ejecutables> WHILE ( <condicion> );
```

<condición> tendrá la misma definición que la condición de las sentencias de selección**.**

<bloque_de_sentencias_ejecutables> podrá contener una sentencia, o un grupo de sentencias ejecutables delimitadas por **begin end**.

- **Tema 16**: ***FOR***

FOR (i **:=** m ; <condición> ; **up**/**down** n) < bloque_de_sentencias_ejecutables > ;
i debe ser una variable entera.
m y n serán constantes enteras

<bloque_de_sentencias_ejecutables> podrá contener una sentencia, o un grupo de sentencias ejecutables delimitadas por **begin end**.

Ejemplos:

```
FOR (i := 1;  i<10 ; up 1)
   Outf ([en el for]);

FOR (j := 10;  j > 0 ; down 2)
Begin
   Outf ({en el for});
End;
```

Nota: Las restricciones de tipo de la variable de control y los componentes del rango serán chequeadas en la etapa 3 del trabajo práctico.

Temas 17 al 19

- **Tema 17 y 18**: ***Asignaciones múltiples***

  Se deben reconocer asignaciones múltiples, que permitan una lista de variables, separadas por coma (","), del lado izquierdo de la asignación, y una lista de expresiones, separadas por coma (","), del lado derecho.

  Por ejemplo:

```
a,b,c := 1+d, e, f+5;
x,y := 1, f1(3), w;
i, j , k := a*3 , b;
```

- **Tema 19**: ***Pattern Matching***

  Se debe incorporar a las condiciones, la posibilidad de comparar listas de expresiones entre paréntesis. Las expresiones se separarán con coma (",").

  Por ejemplo:

```
if ((a,c,...) = (b,2.3,...)) then …

while ((a,b)<(1,2)) begin … end
```

Temas 20 al 22

- **Tema 20**: ***struct***

  Incorporar la posibilidad de declarar tipos **struct** parametrizados en los tipos de sus componentes.
  Por ejemplo:

```
typedef Struct <integer,single,integer> {
   a,
   b,
   c,
}  ts1;
```

  Se podrán declarar variables del tipo definido (en el ejemplo ts1), y los componentes de cada variable se referenciarán con el nombre de la variable, un punto y el nombre de la componente.
  Por ejemplo:

```
ts1 s1,s2,s3;    // declaración de las variables s1 a s3 con el tipo ts1
s1.a := 8;
s2 := s1;
x := s2.b;
```

- **Tema 21**: ***pair***

  Incorporar la posibilidad de declarar tipos **pair** de un tipo determinado (considerando los tipos asignados al grupo).
  Por ejemplo:

```
typedef pair <integer> pint;  //declara un tipo par de enteros
```

  Se podrán declarar variables del tipo definido (en el ejemplo pint), y los componentes de cada variable se referenciarán con el nombre de la variable y la posición de la componente entre corchetes, considerando 1 y 2 para la primera y segunda componente, respectivamente.
  Por ejemplo:

```
pint p1,p2,p3;        // declaración de las variables p1 a p3 del tipo pint
p1[1] := 8;           // se asigna 8 a la primera componente de la variable p1
p2 := p1;             // se asigna la variable p2 con la variable p1
x := p2[1];           // se asigna x con la primera componente de la variable p2
```

- **Tema 22**: ***triple***

Incorporar la posibilidad de declarar tipos **triple** de un tipo determinado (considerando los tipos asignados al grupo).
Por ejemplo:

```
typedef triple <integer> tint;  //declara un tipo tripla de enteros
```

Se podrán declarar variables del tipo definido (en el ejemplo tsing), y los componentes de cada variable se referenciarán con el nombre de la variable y la posición de la componente entre corchetes, considerando 1, 2 y 3 para la primera, segunda y tercera componente, respectivamente.

Por ejemplo:

```
tint t1,t2,t3;        // declaración de las variables t1 a t3 del tipo tint
t1[1] := 8;           // se asigna 8 a la primera componente de la variable t1
t2 := t1;             // se asigna la variable t2 con la variable t1
x := t2[3];           // se asigna x con la tercera componente de la variable t2
```

Temas 23 AL 24; Flujo de Control

- **Tema 23**: ***goto***

Permitir, como sentencia ejecutable, la sentencia goto <etiqueta>, donde la etiqueta será un identificador seguido de dos puntos (":"). Esta etiqueta podrá aparecer en cualquier lugar del código, como una sentencia más.

Ejemplo de código para esta funcionalidad:

```
…
for (i := 1; i < 10; up 1)
begin
     if (...)
     then
          goto afuera:;
     else
          z := z + I;
     end_if;
end;
afuera:
```

El chequeo de la existencia de la etiqueta se efectuará en etapas posteriores.:

// **LÉXICO**: Incorporar a los tokens reconocidos, el token etiqueta. Este token se deberá registrar en la ¨TdeS, del mismo modo que los identificadores.

Tema 24: Iterador con condición

Incorporar a la estructura del encabezado del iterador for, la posibilidad de incluir un cuarto elemento condicional, que se escribirá entre paréntesis. Esta condición tendrá la misma estructura que las condiciones de todas las sentencias de control.

Ejemplo:

```
FOR (j := 1;  j < 10 ; up 2 ; (a > 5))
Begin
   Outf ({en el for});
   a := a + j;
End;
```

Temas 25 a 27: Conversiones

- Tema 25: **Conversiones Explícitas**
  Se debe incorporar en todo lugar donde pueda aparecer una expresión, la posibilidad de utilizar la siguiente sintaxis:

  TOS (<expresión>) // para grupos que tienen asignado el tema 5

TOD(<expresión>) // para grupos que tienen asignado el tema 6

// **LÉXICO**: Incorporar a la lista de palabras reservadas, la palabra **TOS** o **TOD** según corresponda.

- Tema 26: **Conversiones Implícitas**
  Se explicará y resolverá en trabajos prácticos 3 y 4.

- Tema 27: **Sin conversiones en expresiones / Conversiones explícitas de parámetros**
  Se debe incorporar a la sintaxis de la invocación a una función, la posibilidad de anteponer un tipo al nombre del parámetro real, con el fin de convertirlo al tipo del parámetro formal
  Ejemplos:

```
Z := b + f1(integer x);        w := f2(single(a+b)) + f2(z);
```

  Todos los chequeos asociados con compatibilidad de tipos se explicarán y resolverá en trabajos prácticos 3 y 4.

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: Error: Constante de tipo **integer** fuera del rango permitido.
  Línea 43: Error: Falta paréntesis de cierre para la condición de la sentencia **IF**.
  Línea 52: Warning: El identificador identificadormuylargo fue truncado a: identificadormu

- **Contenidos de la Tabla de símbolos**

CONSIDERACIONES GENERALES

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, single, double**) durante el Análisis Sintáctico se 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, podría 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) Casos de prueba que contemplen **todas** las estructuras válidas del lenguaje. Incluir casos con errores sintácticos.