Principios y Técnicas de Compiladores
Trabajo Práctico Nº 2 - Fecha de entrega: 29-09-2023

Objetivo

Construir un Analizador Sintáctico 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.
  Los elementos declarados sólo serán visibles a partir de su declaración (esto será chequeado en etapas posteriores).
- El programa tendrá un nombre y estará delimitado por **begin** y **end**.
- Cada sentencia debe terminar con coma **";"**.
  Ejemplo:

```
main
begin
        integer a, b;
        …
        a:=b+1;
        ulongint x , y, z;
        …
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

  Las variables de la lista se separan con coma (",")
- Sentencias de declaración de funciones con la siguiente estructura:

  <tipo> **fun** <nombre_funcion> (<parámetro>) **begin**
  <cuerpo_ de_la_funcion>
  end

  Donde:
  - <parametro> será un identificador precedido por un tipo.:
    <tipo> ID
    Se permite hasta un parámetro, y puede no haber parámetros.
  - <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:
    return (<retorno>);
    <retorno> podrá ser cualquier expresión aritmética

  Ejemplos válidos:

```
integer fun f1 (integer y)
begin
        integer x;
        x =: y;
        ulongint f11 ()
        begin
        …
        end;
        return (x);
end
```

```
integer fun f2 (longint x)
begin
        integer x;
        x =: y;
        …
        return (x);
end
```

```
ulongint fun f3 ()
begin
        ulongint x;
        x =: 1;
        …
        if (x > 0) then
                return (x);
        else
                return (x + 2);
        end_if;
begin_end
```

Sentencias ejecutables:

- Asignaciones donde el lado izquierdo será 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 debe permitir anidamiento de expresiones con paréntesis.
- Considerar como posible operando de una expresión, la invocación a una función, con el siguiente formato:

  ID(<parametro_real>)

  Donde el parámetro real puede ser un identificador, una constante, o estar ausente.
- Los operadores aritméticos son asociativos a izquierda, y la multiplicación y división tienen mayor precedencia que la suma y la resta.
- 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, entre **"("** y **")"**. 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 iteración asignada al grupo
- Debe permitirse anidamiento de sentencias de selección e iteración. Por ejemplo, puede haber una iteración dentro de una rama de una selección.
- Sentencia de emisión 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.

Temas particulares

Iteraciones

- Grupo 1: **while do**
  while (<condicion>) **do** <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** y **end**.
- **Grupo 2: do while**
  do <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 por **begin** y **end**.

Conversiones

- **Grupo 1**
  En cualquier lugar donde pueda aparecer una variable o una expresión, podrá aparecer:
  itoul(<expresión>**)**
  La semántica de esta conversión se explicará en el trabajo práctico 3
- **Grupo 2**
  Para este grupo, se implementarán conversiones implícitas que se incorporarán en el trabajo práctico 3.

Casos de Prueba

Se debe incluir, como mínimo, ejemplos que contemplen las siguientes alternativas:
(Cuando sea posible, agregar un comentario indicando el comportamiento esperado del compilador)

- Declaración de una variable para cada tipo
- Declaración de una lista de variables
- Declaración de funciones
- Diferentes tipos de sentencias ejecutables:
  - Asignaciones
  - Sentencias de control (con y sin anidamientos)
  - Emisión de mensajes por pantalla
  - Conversiones (si corresponde)
- Para las diferentes estructuras sintácticas, considerar ejemplos válidos, y ejemplos con error.

Salida del Compilador

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

- 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 integer 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 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) Cuando se detecte un error, la compilación debe continuar.

d) 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:

- Descripción del proceso de desarrollo del Analizador Sintáctico: problemas que hayan surgido (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 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) Casos de prueba que contemplen **todas** las estructuras válidas del lenguaje, considerando también casos con error.