Diseño de Compiladores I - 2022
Trabajo Práctico Nº 2
Fecha de entrega: 06/10/2022

ATENCIÓN

Antes de comenzar el desarrollo de esta etapa, revisar las asignaciones de temas particulares que se publicaron con el Trabajo Práctico 1. Para algunos grupos, se efectuaron modificaciones que afectan la lista de palabras reservadas a considerar, y los temas a desarrollar en la segunda etapa.

Para indicar cuáles son esas modificaciones, se puso entre paréntesis el tema de la asignación original, y con un * el tema que lo debe reemplazar.

Por ejemplo:

Para la asignación: 3 7 9 **(13) 14*** 18 20 **21* (23)** 24 27 se debe reemplazar el tema 13 por el 14, y el 23 por el 21

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 nombre de programa, seguido de 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 conjunto de sentencias estará delimitado por llaves '**{**' y '**}**'.
- 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): **i8, i16, i32, ui8, ui16, ui32, f32, f64**
  Las variables de la lista se separan con coma ("**,**")
- Incluir declaración de funciones, con la siguiente sintaxis:

  fun ID (<lista_de_parametros>) : <tipo>

  {

  <cuerpo_ de_la_funcion>

  }

  Donde:
  - <lista_de_parametros> será una lista de parámetros formales separados por ",", con la siguiente estructura para cada parámetro:

    <tipo> ID

    El número máximo de parámetros permitidos es 2, y puede no haber parámetros. **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:

    return (<retorno>);

    <retorno> podrá ser cualquier expresión aritmética

// **LÉXICO**: Incorporar el reconocimiento del símbolo '**:**'

Ejemplos válidos:

```
fun f1 (i16 y) : i16
{
        i16 x;
        x =: y;
        …
        return (x);
}
```

```
fun f2 (i32 x, i16 y) : i16
{
         i16 x;
        x =: y;
        …
        return (x);
}
```

```
fun f3 () : f32
{
        f32 x;
        x =: 1.2;
        …
        if (x > 0.0) then
                return (x);
        else
                return (x + 1.2);
        end_if;
}
```

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, u otras expresiones aritméticas.
  No se deben permitir anidamientos de expresiones con paréntesis.
- Considerar como posible operando de una expresión, la invocación a una función, con el siguiente formato:

  ID(<lista_de_parametros_reales>)

  Donde cada parámetro real puede ser un identificador o una constante.
- 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 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 llaves.

// **LÉXICO**: La palabra reservada **end_if** debe utilizar guión bajo (en el tp1 se indicó end-if)

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

  out(<cadena>);

  Ejemplos:

```
out('Hola mundo');                //Tema 26
out('Hola /                       //Tema 27
        mundo');
```

TEMAS PARTICULARES

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

Temas 9 y 10

- Tema 9: **discard**
  Incorporar a la gramática, la o las reglas que permitan reconocer invocaciones a funciones como si fueran procedimientos. Es decir:

  ID(<lista_de_parametros>);

  Si el compilador detecta una invocación como la indicada, debe informar Error Sintáctico.
  Sin embargo, si la invocación es precedida por la palabra reservada **discard**, la sentencia no generará error, y la invocación podrá considerarse como válida.
  Ejemplo:

```
discard f1(2);
```

- Tema 10: **Cláusulas de compilación**
  Incorporar, a las sentencias declarativas, la posibilidad de definir constantes, con la siguiente estructura:

  const <lista_de_constantes>;

  donde la lista incluirá una lista de constantes separadas por ','. Cada constante se definirá con la siguiente sintaxis:

  <nombre_de_constante> =: <valor_de_constante>

Ejemplo:

```
const c1 =:10, x=:5;
```

Incorporar, a la sentencias ejecutables, la siguiente estructura:

when (<condición>) **then** <bloque_de_sentencias>

donde:

<condición> tendrá la misma estructura que las condiciones de las sentencias de control
<bloque_de_sentencias> podrá contener tanto sentencias ejecutables como declarativas.

// **LÉXICO**: Incorporar a la lista de palabras reservadas, la palabra **const**.

Temas 11 a 16: Sentencias de Control

- ***While con continue*** (tema 11 en TP1)

  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 llaves. Además, podrá contener sentencias **break** y sentencias **continue**. Ambas se escriben con la palabra reservada seguida de ';'

- ***While con expresión*** (tema 12 en TP1)

  while (<condicion>) **:** (<asignación>) <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 llaves. Además, podrá contener sentencias break (se escribe con la palabra reservada seguida de ';')

  Ejemplo:

```
while (i < 10) : (i =: i + 1) {
out('ciclo');
};
```

- ***Do until con continue*** (tema 13 en TP1)

  do <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 llaves. Además, podrá contener sentencias **break** y sentencias **continue**. Ambas se escriben con la palabra reservada seguida de ';'

- ***Do until con expresión*** (tema 14 en TP1)

  do <bloque_de_sentencias_ejecutables> **until** (<condicion>) **:** (<asignación>) **;**

  <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 llaves. Además, podrá contener sentencias break (se escribe con la palabra reservada seguida de ';')

  Ejemplo**:**

```
do {
        out('ciclo');
} until (i > 10) : (i =: i + 1);
```

- ***for con continue*** (tema 15 en TP1)

  for (i =: n; <condición_for>; +/- j) < bloque_de_sentencias_ejecutables > **;**

  i debe ser una variable de tipo entero (1-2-3-4-5-6).
  n y j serán constantes de tipo entero (1-2-3-4-5-6).
  <condición_for> será una comparación de i con m. Por ejemplo: i < m
  Donde m puede ser una variable, constante o expresión aritmética de tipo entero (1-2-3-4-5-6).
  El signo '+' o '-' al final del encabezado es obligatorio..

  <bloque_de_sentencias_ejecutables> podrá contener una sentencia, o un grupo de sentencias ejecutables delimitadas por llaves. Además, podrá contener sentencias **break** y sentencias **continue**. Ambas se escriben con la palabra reservada seguida de ';'

Nota: Las restricciones de tipo serán chequeadas en la etapa 3 del trabajo práctico.

- ***for*** (tema 16 en TP1)

  for (i =: n; <condición_for>; +/- j) < bloque_de_sentencias_ejecutables > **;**

  i debe ser una variable de tipo entero (1-2-3-4-5-6).
  n y j serán constantes de tipo entero (1-2-3-4-5-6).
  <condición_for> será una comparación de i con m. Por ejemplo: i < m
  Donde m puede ser una variable, constante o expresión aritmética de tipo entero (1-2-3-4-5-6).
  El signo '+' o '-' al final del encabezado es opcional. La ausencia de signo implica incremento.

  <bloque_de_sentencias_ejecutables> podrá contener una sentencia, o un grupo de sentencias ejecutables delimitadas por llaves. Además, podrá contener sentencias **break, que s**e escriben con la palabra reservada seguida de ';'

  Nota: Las restricciones de tipo serán chequeadas en la etapa 3 del trabajo práctico.

Temas 17 y 18: Sentencias de Control con etiquetado

- ***Break con etiquetado*** (tema 17 en TP1)

  Incorporar, a la sentencia de control asignada (temas 11 al 16), la posibilidad de antecederla con una etiqueta.

  Incorporar, a la sentencia **break**, la posibilidad de incluir una etiqueta.

  Por ejemplo:

```
outer: while (i < 10) {
        while (j < 5) {
            break :outer;
        };
    };
```

- ***Continue con etiquetado*** (tema 18 en TP1)

  Incorporar, a la sentencia de control asignada (temas 11 al 16), la posibilidad de antecederla con una etiqueta.

  Incorporar, a la sentencia **continue**, la posibilidad de incluir una etiqueta.

  Por ejemplo:

```
outer: while (i < 10) : (i =: i + 1) {
        while (j < 5) : (j := j + 1) {
            continue :outer;
        };
    };
```

Temas 19 y 20

- Tema 19: **Diferimiento**
  Incorporar, a las sentencias ejecutables, la posibilidad de antecederlas por la palabra reservada **defer**. Esta funcionalidad podrá aplicarse a estructuras sintácticas de una sola sentencia, como asignación o emisión de mensajes, pero también a estructuras de bloque como sentencias de selección o iteración.
  Ejemplos:

```
defer  out(a);
```

```
defer while(i<10)
{
     out ('ciclo');
     i=:i+1;
}
```

  // **LÉXICO**: Incorporar a la lista de palabras reservadas, la palabra **defer**.

- Tema 20: **Sentencia de control como expresión**
  Incorporar, a la sentencia de control asignada en los temas 11 a 16, la posibilidad de devolver un valor utilizando la sentencia break, o un valor por defecto en la finalización normal de la sentencia.

Ejemplo para while: (11)

```
a =: while (i < end){
     if (i = number) {
        break 1;
        }
     } else 3;
```

Ejemplo para do until: (13)

```
a =: do {
     if (i = number) {
        break 1;
        }
     }until (i> end) else 3 ;
```

Ejemplo para for: (15/16)

```
a =: for (i =: 0 ;i < end ; +1){
     if (i = number) {
        break 1;
        }
     } else 3;
```

Ejemplo para while: (12)

```
a =: while (i < end) : (i=:i+1)
{
     if (i = number) {
        break 1;
        }
     } else 3;
```

Ejemplo para do until: (14)

```
a =: do {
     if (i = number) {
        break 1;
        }
     } until (i> end) : (i=:i+1) else 3 ;
```

Temas 21 a 23: Conversiones

- Tema 21: **Conversiones Explícitas**: . Se debe incorporar en todo lugar donde pueda aparecer una expresión, la posibilidad de utilizar la siguiente sintaxis:

  tof32(<expresión>) // para grupos que tienen asignado el tema 7

  tof64(<expresión>) // para grupos que tienen asignado el tema 8

// **LÉXICO**: Incorporar a la lista de palabras reservadas, la palabra **tof32** o **tof64** según corresponda.

- Tema 22: **Conversiones Implícitas**: Se explicará y resolverá en trabajos prácticos 3 y 4.
- Tema 23: **Sin conversiones:** Se explicará 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: Constante de tipo **i8** fuera del rango permitido.
  Línea 43: Falta paréntesis de cierre para la condición de la sentencia **if**.
- 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 (**i8, i16, i32, f32, f64**) 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 **i16** 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.