Memoria Final

Entrega parcial

Analizador sintáctico BSL

Realizado por:
Marcelino Tena Blanco; NIA: 100383266

Índice

Contenido

1. Introducción

El siguiente documento trata sobre la realización y el procedimiento para llegar a la creación de un analizador sintáctico para el lenguaje BSL, el cual está especificado su formato en aula global. El analizador es capaz de reconocer el lenguaje BSL según la documentación al completo. A continuación, explico la gramática escogida para su lectura:

2. Gramática

La gramática llevada a cabo en el ejercicio tiene la siguiente estructura:

Programa y bloques de sentencias

Es la regla inicial de la gramática y reconoce si es un programa BSL, además de que también permite un archivo vacío.

```
programa ::= blq_sentencias
           | /*lambda*/
           ;
```

Bloque de sentencias

Permite tener varias sentencias en un mismo documento. Lo que realiza en una recursión que va analizando sentencia por sentencia por separado, para que un fichero pueda tener tantas sentencias como quiera el usuario.

```
blq_sentencias ::= blq_sentencias sentencia
                 | sentencia
                 ;
```

Sentencias

Describe todas las sentencias que puede haber en un lenguaje BSL.

```
sentencia ::= sent_decl //SENTENCIAS DE DECLARACION:
            | sent_uso //SENTENCIAS DE INICIALIZACION
            | sent_flujo //SENTENCIA DE FLUJO
            ;
```

Las diferentes sentencias tienen la siguiente estructura:

1. Sentencias de declaraciones

Las sentencias de declaraciones son aquellas que otorgan un tipo a una variable, ya sea uno de los cuatro tipos del lenguaje (entero, real, char o buleano) o un tipo estructura, es decir, declarar un identificador mediante un identificador. En esta parte declaramos variables, estructuras y funciones.

```
sent_decl ::= decl_variable SEMI
            | decl_struct
            | decl_funcion
            ;
```

1.1 Declarar un variable

Una variable se declara mediante un declarador que puede ser un tipo clave que será comentado más abajo. Se puede declarar la variable de dos formas: una es dándole un valor inicial y la otra es solo declarándola, donde entonces puedes añadir varios identificadores entre comas declarándolos todos.

```
decl_variable ::= keytipo ID DPTOS IGUAL dec_exp_n1
                | keytipo identificado
                ;
```

1.1.1 Declarar varias variables

Esta regla se utiliza para declarar varias variables a la vez de un tipo. Es una recursión, donde el valor de identificadores mínimo debe ser 1 y no existe máximo.

```
identificado ::= identificado COMA ID
               | ID
               ;
```

1.2 Declarar una estructura

Un struct se detecta mediante la palabra clave struct y se otorga el nombre del identificador. Luego, entre corchetes, tendrá las variables declaradas que solicite el desarrollador.

```
decl_struct ::= STRUCT ID LCORCH lista_struct RCORCH
              ;
```

1.2.1 Lista de variables de la estructura

Lista de variables que puede tener un struct. Un struct tiene como mínimo una variable declarada y no tiene máximo.

```
lista_struct ::= decl_variable SEMI lista_struct
               | decl_variable SEMI
               ;
```

1.3 Declarar una función

Una función solo se puede declarar de una forma: debe tener la palabra clave función, luego el id de la función, entre paréntesis los parámetros de la función, la palabra clave return y por último el bloque de sentencias entre llaves

```
decl_funcion ::= FUNCION ID LPAREN identificado_funcion RPAREN
RETURN keytipo LCORCH blq_sentencias RCORCH
                ;
```

1.3.1 Parámetros de la función

Los parámetros de la función se declaran mediante comas y sin ningún valor. También puede ocurrir que no tenga ningún parámetro la función por lo que para ello se utiliza el lambda.

```
identificado_funcion ::= declaradores
                        |/*lambda*/
                        ;
declaradores ::= declaradores COMA keytipo ID
               | keytipo ID
               ;
```

2. Sentencias de uso o inicialización

Mediante estas sentencias se pueden cambiar el valor de una variable o de un struct. También se reconocer expresiones aritmeticológicas.

```
sent_uso ::= asignacion SEMI
           | dec_exp_n1 SEMI
           ;
```

2.1 Asignar un valor a una variable

Para asignarle el valor de una variable lo que realizo es obtener el id que quiero cambiar y luego lo igualo a la expresión aritmeticológica que quiero.

```
asignacion ::= type_struct DPTOS IGUAL dec_exp_n1
                        ;
```

2.1.1 Variable tipo estructura o normal

Para obtener una variable, lo que realizo es una recursividad donde el mínimo es obtener 1 ID y no existe máximo al obtener varias variables seguidos de puntos.

```
type_struct     ::= type_struct PUNTO ID
                  | ID
                  ;
```

2.2 Una calculadora sin variable

La calculadora es aceptar expresiones aritmeticológicas que explicaré más tarde en el documento.

3. Sentencias de flujo

Las sentencias de flujo son estructuras que realizan saltos de línea según una condición. Existe dos flujos, el condicional donde se realiza lo que existe dentro si la condición es verdadera y el bucle, donde se realiza lo que tiene dentro mientras la condición sea verdadera.

```
sent_flujo ::= condicional
             | bucle
             ;
```

3.1 Condicional

El condicional está realizado como lo realiza el manual de BSL. Empiezo con un si, una expresión aritmeticológica, entonces, un bloque de sentencias y finsi. En el caso en el que exista un sino, entonces se realizará el sino y su contenido después del bloque de sentencias

```
condicional ::= SI dec_exp_n1 ENTONCES blq_sentencias FINSI
              | SI dec_exp_n1 ENTONCES blq_sentencias SINO blq_sentencias FINSI
                    ;
```

3.2 Bucle

El bucle es igual que el condicional pero en vez de realizar el contenido de este solo una vez, se realiza hasta que la expresión de la condición sea falsa.

```
bucle ::= MIENTRAS dec_exp_n1 blq_sentencias FINMIENTRAS
      ;
```

Tipos de variables

Las variables pueden ser de cuatro tipos esenciales (entero, real, buleano o carácter) y de tipo ID, solo para las estructuras.

```
keytipo ::= DENTERO
          | DREAL
          | DBOOLEAN
          | DCARACTER
          | ID
          ;
```

Reglas aritmeticológicas

Las reglas siguientes están según la prioridad de las expresiones, es decir, las expresiones aritméticas son más prioritarias que las funciones lógicas. Esto significa que las expresiones más prioritarias se llaman al final y cuanto menos prioritaria sea una expresión, más arriba en las reglas estará. En estas reglas se incluyen desde las expresiones aritmeticológicas, hasta los id de tipo estructura o sin tipo estructura y la llamada a las funciones.

```
dec_exp_n1 ::= dec_exp_n1 MENOR dec_exp_n1
                | dec_exp_n1 MAYOR dec_exp_n1
                | dec_exp_n1 MAYORIGUAL dec_exp_n1
                | dec_exp_n1 MENORIGUAL dec_exp_n1
                | dec_exp_n1 IGUALIGUAL dec_exp_n1
                | dec_exp_n1 NOTIGUAL dec_exp_n1
                | dec_exp_n2
                ;

dec_exp_n2 ::= dec_exp_n2 PLUS dec_exp_n2
                 | dec_exp_n2 MINUS dec_exp_n2
                 | dec_exp_n2 OR dec_exp_n2
                 | dec_exp_n2 AND dec_exp_n2
                 | dec_exp_n3
                 ;

dec_exp_n3 ::= dec_exp_n3 TIMES dec_exp_n3
                 | dec_exp_n3 DIV dec_exp_n3
                 | dec_exp_n4
                 ;

dec_exp_n4 ::= MINUS dec_exp_n5
             | NOT dec_exp_n5
             | PLUS dec_exp_n5
             | dec_exp_n5
             ;

dec_exp_n5 ::= NUMBER
             | LPAREN dec_exp_n1 RPAREN
             | UMINUS
             | BOOLEAN
             | CHAR
             | type_struct
             | ID LPAREN lexp RPAREN
             ;

lexp ::= lexp COMA dec_exp_n1
       | dec_exp_n1
        ;
```

3. Pruebas realizadas

Las pruebas realizadas al igual que en las prácticas anteriores son las que dan de ejemplo en los apuntes debido a que no he conseguido suficiente tiempo para realizar más comprobaciones.
Se ha comprobado los siguientes elementos:
•Operadores Aritmeticológicos
Este es el apartado más comprobado debido a que existe bastantes valores en el fichero.
•Declaración de variables e inicialización de variables
También muy comprobado, ya que está definido de varias formas las variables en el fichero.
•Declaración de funciones
Realizado las funciones que vienen de ejemplo, además de algunas añadidas por mi parte para comprobar los parámetros de entrada.
•Bucles y sentencias
Con los ejemplos otorgado se obtienen bastantes formas de estas estructuras por lo que creo que está bien probado.
•Estructuras structs
Los structs también están bien probados con los que vienen en los ejemplos otorgados por lo que no creo que haga falta más pruebas.

4. Cambios en el léxico

Debido a algunos problemas de reducción en el analizador sintáctico, he decido agregar nuevos terminales a la gramática para evitar los problemas anteriores. He agregado los siguiente terminales:

- MAYORIGUAL = “>=”
- MENORIGUAL = “<=”
- NOTIGUAL = “!=”
- IGUALIGUAL = “==”

Solo esto he cambiado en el analizador léxico.

5. Conclusiones

El analizador sintáctico debería funcionar correctamente, aunque existe ocasiones en los que no sé si es culpa del programa en el que no ocurre nada, entonces tengo que borrar los documentos java creados automáticamente y ejecutar.
En mi opinión la parte sintáctica no ha sido complicada ya que hemos tenido la ayuda de la documentación extensa de aulaglobal y creo que es una buena forma de ayudar a los alumnos ya que se ofrece una amplia explicación de lo que hay que realizar, aunque en ocasiones faltan caracteres y hace parecer como que faltan partes del documento.

Como ya comprenderá, no tengo mucho tiempo por lo que cosas como realizar comprobaciones más selectas igual que realizaba en la anterior práctica está complicado. Por lo demás, creo que me ha servido la práctica para conocer más a fondo como funciona un analizador sintáctico de un lenguaje.