Memoria Final

Analizador sintáctico BSL

Realizado por:
Marcelino Tena Blanco; NIA: 100383266
Año 2019-2020

Índice

Contenido

1. Introducción

El presente documento trata sobre la elaboración de la entrega final de la práctica 2. En el documento se especifica la gramática que ha sido utilizada para llevar a cabo el procesador del lenguaje BSL, la cual es igual a la entrega anterior ya que no se han realizado cambios en ella. Este documento tiene de nuevo la especificación de la solución del apartado semántico y una renovada capa de pruebas para verificar el óptimo funcionamiento del procesador.

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 sí, 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. Descripción de la solución

La solución semántica otorgada presenta las siguientes características:

- Declaración de símbolos o de varios símbolos según el tipo de variable:
  - REAL
  - ENTERO
  - BULEANO
  - CHAR
  - ID: este se busca en la tabla de registros y es un struct. En el caso de que se utilice este ID, no es posible ninguna inicialización.

  En este caso, como la variable no se inicializa, existe una función en FuncionesAyuda (una clase java) donde se otorga el valor predeterminado de la variable. En el caso de los struct, se copia el registro con los valores predeterminados de cada uno de los elementos de este.
- Declaración de símbolos con inicialización para todos los tipos comentados antes menos para ID. También se ha agregado un sistema anti duplicados, que lo que haces es que antes de crear la variables nueva busca en la tabla si existe alguna que se llame igual.
- Cambio de valor para todas las variables de los tipos anteriores, además del óptimo funcionamiento de las variables con struct.
- Las sentencias de flujo, es decir, los condicionales y los bucles son capaces de conocer si lo que tienen de condición es un buleano. No son capaces de solo reconocer si el contenido se debe reconocer o no, por lo tanto lo reconocen.
- Existe la posibilidad de calculadora, para ello lo que hay que realizar es agregar una expresión al programa y este imprimirá el resultado, por ejemplo:
  - 5+7; (impresión por pantalla 12)
  - idCualquiera; (imprime por pantalla el valor del id)
- Declaración de struct, de forma que esta se graba con valores predeterminados en la tabla de registros. Estos valores solo sirven para inicializar los valores de los símbolos struct que se declararán.
- Declaración de funciones donde se guarda en la tabla de registros la entrada (tipos de variable) y salida (solo un tipo). Tiene el problema de que no es capaz de utilizar las funciones de entrada en el programa, por lo que no está finalizado.
- Uso de funciones: identifica si la función existe y si los parámetros pasado por parámetro son del tipo que se necesita.
- Sistema de error con parada completa del procesador en caso de que se detecte que:
  - el valor que se está asignando a una variable es incorrecto
  - la variable no existe en la tabla de símbolos
  - la función o el struct no existe en la tabla de registros

    - el tipo al que se quiere declarar no es compatible con inicializaciones (struct) o que no existe ese tipo
    - la variable ya existe en la tabla de símbolos
    - el struct o la función ya existe en la tabla de registros
    - no sea capaz de agregar una nueva función
    - le pases un valor que no es una expresión condicional a las funciones de flujo (condicional y bucle). Hay que realizar un buen uso de los paréntesis en condiciones como {(a >= b) AND (a >= c)} porque si no este error saltará ya que realizará la acción de b AND a y dará error en el caso de que las variables sean números
    - le pases por parámetro a una función valores que no se pueden convertir o que no son los que necesita la función
- Funcionamiento completo de todas las operaciones aritmeticológicas, de devolución de boléanos, de números o de caracteres, además de dar los valores de struct y de símbolos. No es capaz de dar valores de retorno de funciones.
- Impresión completa y detallada en archivo y terminal de la tabla de símbolos, dando valores en los tipos primitivos y dando todos los valores de los símbolos struct
- Impresión completa y detallada de todos los struct en archivo y terminal.

En todas las características de asignar un valor dichas anteriormente está habilitado el casteo automático, es decir, si el tipo que le pasas es un entero y se necesita un real, entonces se castea a real. Esto pasa de entero a real, de real a entero, de char a real y de char a entero.

A continuación explico de forma detallada cada una de las características anteriores que contengan alguna complejidad:

- Variables globales utilizadas

Las variables globales utilizadas son 3 arraylists:

- La tabla de símbolos denominada symbolTable
- La tabla de registros denominada registerTable
- Una tabla de ayuda que guarda DatosVar

Además de las variables, están dos clases objeto que son DatosVar que almacena todos los datos de las variables:

- Tipo: el tipo de variables que es.
- ID: el nombre de la variable
- Valor: tipo objeto donde se almacena el valor de la variable
- Estructura: es de tipo DatosVar donde se almacena las variables de tipo estructura. En caso de ser un struct, el Valor solo se obtendrá de los valores de tipo primitivo y en caso de ser un tipo primitivo estructura será nulo.

Y TablaRegistro, donde almacena el nombre del struct o de la función y los datos que almancena la misma como los parámetros o los datos de los struct
Por último existe una función llamada FuncionesAyuda que otorga funciones capaces de refactorizar código del análisis semántico.

- Declaración de símbolos o de varios símbolos según el tipo de variable

Para realizar esto, lo que hago es almacenar las variables en una lista auxiliar. El tipo de variable lo que realizo es ponerle "|declarar" para después poderle el tipo que realmente es, darle el valor predeterminado y por último lo agrego en la tabla de símbolos.

- Declaración de símbolos con inicialización

Creo un objeto DatosVar donde voy a almacenar los nuevos datos de la nueva variable. Según el tipo de dato, le daré un valor u otro, ya que hay que transfórmalo según el tipo. Después se inserta en la tabla de símbolos la nueva variable. También verifico que la variable no se va a duplicar o que el valor que le has pasado es correcto con el tipo o si es posible transformarlo. En casos contrarios, entonces saltaría el debido error informando el problema.

- Cambio de valor de un símbolo

Lo primero que hago es buscarlo en la tabla de símbolos. En el caso de que no exista, paro el compilador y cierro. Una vez obtenido donde está almacenado le cambio el valor.

- Sentencias de flujo

Lo único que se realiza es verificar que el condicional es un booleano. Para ello lo que hace es una verificación te tipo, en la que si el tipo detectado no es booleano entonces se para el analizador y muestra el error.

- Calculadora

Obtiene el valor de la expresión y devuelve por pantalla lo que obtiene. No se agrega a ningún sitio ni nada. Solo muestra el valor por pantalla.

- Declaración de struct

Para declarar un struct lo primero que hago es revisar si el id no existe ya. En el caso de que no exista, entonces añado a la tabla de registros el nuevo id con todos sus id, que estos irán en objeto DatosVar. Los objetos DatosVar tienen dos formas de ser: pueden ser valores únicos, o conjuntos de DatosVar. Por lo que pueden es posible realizar struct.

- Declaración de funciones

Lo primero que realizo es verificar que el id no está duplicado. Después agrego la nueva función en la tabla de registros con el id, la entrada y la salida. Las

funciones se corresponden con solo dos DatosVar, donde el primero es el input y el segundo es el output. En ambos las variables se agregan en la lista de DatosVar de cada uno.

- Uso de funciones

Es una parte de obtener el valor de las expresiones. Lo único que realiza es conocer si los parámetros dados son correctos según la función. Para ello lo primero que hace es buscar la función en la tabla de registros. En el caso de que no exista da error. Después obtiene los valores de entrada mediante un string y lo parte en arrays, donde cada celda es un valor. Entonces cada una de las entradas se verifica con el tipo que puede ser. En caso de que el tipo no sea convertirle o no se corresponda con lo que dice la tabla de registros, se dará error.

- Impresión final de todas las tablas

Para imprimir todos los registros lo que hago es que en el programa se añaden a un String mediante [fors]. Los guiones significan que son variables que están dentro de otras. Se almacena también en un archivo llamado salidaVariables.txt.

4. Pruebas realizadas

Existe 2 tipos de pruebas: Las pruebas de error y las pruebas de acierto. Las de acierto van a ser una modificación de cada uno de los ejemplos dados más grupo_t_pruebaOK.txt que prueba todas las anteriores, y las de fallo consistirá cada una en:

- el valor que se está asignando a una variable es incorrecto
- la variable no existe en la tabla de símbolos
- la función o el struct no existe en la tabla de registros
- el tipo al que se quiere declarar no es compatible con inicializaciones (struct) o que no existe ese tipo
- la variable ya existe en la tabla de símbolos
- el struct o la función ya existe en la tabla de registros
- le pases un valor que no es una expresión condicional a las funciones de flujo (condicional y bucle). Hay que realizar un buen uso de los paréntesis en condiciones como {(a >= b) AND (a >= c)} porque si no este error saltará ya que realizará la acción de b AND a y dará error en el caso de que las variables sean números
- le pases por parámetro a una función valores que no se pueden convertir o que no son los que necesita la función

La tabla siguiente muestra los ficheros de pruebas erróneas. La tabla de nombre del fichero especifica como está denominado el archivo, la acción es que se va a realizar, que se espera es lo que se espera que se obtenga y por último verificación expresa si lo que se esperaba es verdad (correcto) o mentira (incorrecto)

NOMBRE DEL FICHERO	ACCIÓN	QUE SE ESPERA	VERIFICACIÓN
ElStructNoExisteAlcambiarValor.txt	Cuando se intenta cambiar el valor de un struct, el struct no existe en la parte izquierda.	Que dé un error de que el struct no existe.	CORRECTO.
ElStructNoExisteAlDarValor.txt	Cuando se intenta modificar el valor de una variable, en la parte derecha hay un struct que no existe.	Que dé un error de que el struct no existe.	CORRECTO
ElStructYaExiste.txt	Se registran dos struct con el mismo identificador.	Dar un error de que el struct ya está registrado	CORRECTO
ErrorDeBucle.txt	El valor dado para el condicional no es un valor buleano.	Da un error de que el valor no es un valor condicional.	CORRECTO
ErrorDeCondicional.txt	El valor dado para el condicional no es un valor buleano.	Da un error de que el valor obtenido no es un valor para hacer un condicional.	CORRECTO
FuncionParametrosNoIguales.txt	El valor que le pasas por parámetro a una función no es correcto.	Da un error diciendo que los valores pasados por parámetro son erróneos.	CORRECTO
LaFuncionNoExiste.txt	En el caso de que se intente dar el valor que se obtiene a través de una función y no existe.	Da un error de que la función no existe.	CORRECTO
LaFuncionYaExiste.txt	En el caso de que se declare una función. 2 veces.	Da el error de que la función ya existe en el registro.	CORRECTO

laVariableYaExiste.txt	En el caso de que intentes declarar una variable 2 veces.	Da el error de que la variable ya existe	CORRECTO
ValorTipoIncorrectoBoolean.txt	En el caso de que intentes declarar un booleano.	Da el error de que la variable no es un booleano	CORRECTO
ValorTipoIncorrectoChar.txt	En el caso de que intentes declarar un carácter que no se puede declarar.	Da el error de que no se puede declarar	CORRECTO
ValorTipoIncorrectoEntero.txt	Declarar una variable entera con un valor buleano.	Da el error de que no se puede declarar.	CORRECTO
ValorTipoIncorrectoReal.txt	Declarar una variable real con un valor buleano.	Da el error de que no se puede declarar.	CORRECTO
VariableNoExisteAlCambiarValor.txt	Al no estar declarada la variable al cambiar el valor.	Da un error de que no existe la variable.	CORRECTO
VariableNoExisteAlObtenerValor.txt	Al intentar obtener el valor de la variable.	Da el error de que no existe la variable.	CORRECTO

5. Conclusiones

El analizador semántico cumple con todo lo propuesto excepto los casos opcionales de recuperación de errores y de función. Esto es debido a que en las funciones existe el problema de que no soy capaz de añadir en la tabla de símbolos de forma temporal los datos de entrados por parámetro en la función. Por lo demás de las funciones, son correctas. El caso de los errores es debido a que no entiendo como añadir los cuidados (warnings) porque típicamente son del estilo de que un struct no está inicializado o una variable no está inicializada, pero en nuestro ejercicio todas las variables se inicializan con un valor por defecto por lo que no es posible dar un aviso.

En parte global del ejercicio, el procesador funciona correctamente, teniendo en cuenta los tres inputs diversos de aciertos y todos las pruebas de fallos que se obtienen al realizar mal las acciones.

Para concluir, quiero decir que el ejercicio no ha llegado a ser complicado una vez que lo entiendes, pero para llegar a ello primero debes encontrar información. Creo que los manuales dados en aula global son algo escasos para poder encaminar la parte de semántica debido a que te cuenta como debería ser, pero no cómo se debería hacer en el sentido de cómo se debería guardar los símbolos en las tablas o cómo se debería guardar los registros.