

Memoria Final

Analizador sintáctico BSL

Realizado por:

Marcelino Tena Blanco; NIA: 100383266

Año 2019-2020

Índice

Contenido

[1. Introducción 3](#_Toc41298107)

[2. Gramática 3](#_Toc41298108)

[Programa y bloques de sentencias 3](#_Toc41298109)

[Bloque de sentencias 3](#_Toc41298110)

[Sentencias 3](#_Toc41298111)

[Tipos de variables 7](#_Toc41298112)

[Reglas aritmeticológicas 7](#_Toc41298113)

[3. Descripción de la solución 9](#_Toc41298114)

[5. Cambios en el léxico 9](#_Toc41298115)

[6. Conclusiones 9](#_Toc41298116)

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 que la gramática de 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.

1. 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. 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. 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. 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. 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. 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. 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

;

1. 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

;

* 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

;

* + 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

;

* 1. Una calculadora sin variable

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

1. 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

;

* 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

;

* 1. 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

;

1. 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 structs, 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. Agregado también un sistema anti-duplicados 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 structs.
* 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 structs, 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 structs 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. En sí, en esta característica solo es capaz de declarar bloques de sentencias con variables que no sean de entrada.
* 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 (structs) 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, números o caracteres, además de dar los valores de structs y de símbolos. No es capaz de dar valores de retorno de funciones.
* Impresión completa y detallada 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 structs.

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
* Declaración de símbolos o de varios símbolos según el tipo de variable
* Declaración de símbolos con inicialización
* Cambio de valor de un símbolo
* Sentencias de flujo
* Calculadora
* Declaración de structs
* Declaración de funciones
* Uso de funciones
* Sistema de error
* Operaciones aritmeticológicas o de obtención de valor
* Impresión final de todas las tablas

1. 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.

1. 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.