UNIVERSIDAD DE GRANADA

ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Práctica 3: Analizador Sintáctico

Procesadores de Lenguajes

Grupo 4

Autores:
Campoy Maldonado, Sergio · 75943758N
Castro Rivero, Exequiel Alberto · 78308513E
Galera Gazquez, Antonio · 20887221R
Lugli, Valentino Glauco · YB0819879

Índice

1 Introducción **2**

2 Descripción formal del BNF **2**

3 Definición de la semántica **5**
3.1 Elementos básicos del lenguaje 5
3.2 Variables 6
3.3 Funciones 6
3.4 Cuerpo del Programa 7
3.4.1 Condicionales 7
3.4.2 Bucles 7
3.4.3 Entrada y Salida por Pantalla 8
3.4.4 Listas 8

4 Identificación de los tokens **8**

1. Introducción

En esta práctica se presenta un nuevo lenguaje de programación compilado y fuertemente tipado denominado C+O-.

Este lenguaje posee una estructura similar al lenguaje de programación C, aunque –entre otras diferencias– palabras reservadas en español, posibilidad de declarar únicamente funciones y la posesión de listas de otros tipos de datos como tipo de dato nativo.

La gramática del lenguaje se desarrolla principalmente en la notación de Backus-Naur (BNF) mostrando todas las peculiaridades del lenguaje.

Luego se realiza una definición más a alto nivel de la semántica, es decir, se explica en esencia cual es la estructura del lenguaje y como se escribe un programa en C+O-.

Por último se presenta la tabla de tokens identificados en la gramática del lenguaje.

2. Descripción formal del BNF

```
<programa>                        ::= <cabeceraPrograma> <bloque>

<cabeceraPrograma>                ::= "inicio"

<bloque>                          ::= <inicioBloque>
                                      <declararVariablesLocalesMulti>
                                      <declararFuncionMulti>
                                      <sentencias>
                                      <finBloque>

<declararFuncionMulti>            ::= <declararFuncionMulti> | <declararFuncion>
                                  |   E

<declararFuncion>                 ::= <cabeceraFuncion> <bloque>

<declararVariablesLocalesMulti> ::= <marcaInicioVariable>
                                      <variablesLocalesMulti>
                                      <marcaFinVar> | E

<cabeceraFuncion>                 ::= <tipoDato>
                                      <identificador>
                                      <inicioParametros>
                                      <parametros>
                                      <finParametros>

<inicioParametros>                ::= "("

<finParametros>                   ::= ")"

<marcaInicioVariable>             ::= "inivar"

<marcaFinVar>                     ::= "finvar" <finSentencia>
```

```
<inicioBloque>                  ::= "{"

<finBloque>                     ::= "}"

<variablesLocalesMulti>         ::= <variablesLocalesMulti> <variableLocal>
                                |   E

<variableLocal>                 ::= <tipoDato> <variableSolitaria>
                                    <identificador> <finSentencia>

<variableSolitaria>             ::= <variableSolitaria> <identificador> ","
                                | E

<tipoDato>                      ::= <tipoElemental> | "lista de"
                                    <tipoElemental>

<tipoElemental>                 ::= "entero" | "real" | "caracter" |
                                    "booleano"

<sentencias>                    ::= <sentencias> <sentencia>
                                | <sentencia>

<sentencia>                     ::= <bloque>
                                |   <sentenciaAsignacion>
                                |   <sentenciaIf>
                                |   <sentenciaWhile>
                                |   <sentenciaEntrada>
                                |   <sentenciaSalida>
                                |   <sentenciaReturn>
                                |   <sentenciaFor>
                                |   <sentenciaLista>

<sentenciaAsignacion>           ::= <identificador> "=" <expresion>
                                    <finSentencia>

<sentenciaIf>                   ::= "si" <inicioParametros> <expresion>
                                    <finParametros> <sentencia>
                                |   "si" <inicioParametros> <expresion>
                                    <finParametros> <sentencia>
                                    "sino"
                                    <sentencia>

<sentenciaWhile>                ::= "mientras" <inicioParametros> <expresion>
                                    <finParametros> <sentencia>

/* Sentencia de entrada */
<sentenciaEntrada>              ::= <nombreEntrada> <listaVariables>
                                    <finSentencia>

<nombreEntrada>                 ::= "escanear"

<listaVariables>                ::= <inicioParametros> <parametros>
                                    <finParametros>
```

```
<sentenciaReturn>               ::= "repatriar" <expresion> <finSentencia>

<sentenciaFor>                  ::= "para" <sentenciaAsignacion> "hasta"
                                    <expresion> <sentido> <sentencia>
                                |   "para" <expresion> "hasta" <expresion>
                                    <sentido> <sentencia>

<sentido>                           ::= "aumentando" | "decrementando"

<sentenciaSalida>               ::= <nombreSalida> <inicioParametros>
                                    <listaExpresionesCadena> <finParametros>
                                    <finSentencia>

<listaExpresionesCadena>        ::= <expresion> | <cadena>
                                |   <listaExpresionesCadena> "," <expresion>
                                |   <listaExpresionesCadena> "," <expresion>

<nombreSalida>                  ::= "anunciar"

/* Parametros */
<parametros>                    ::= <tipoDato> <identificador>
                                |   <parametros> "," <tipoDato> <identificador> | E
/* Expresion */
<expresion>                     ::= "(" <expresion> ")"
                                |   <opUnario> <expresion>
                                |   <expresion> <opBinario> <expresion>
                                |   <identificador>
                                |   <literal>
                                |   <funcion>
                                |   "#" <expresion>
                                |   "?" <expresion>
                                |   <expresion> "@" <expresion>
                                |   <expresion> "++" <expresion> "@" <expresion>
                                |   <expresion> "--" <expresion>
                                |   <expresion> "**" <expresion>

<funcion>                       ::= <identificador> "(" <argumentos> ")"

<argumentos>                    ::= <expresion> | <argumentos> "," <expresion>
<identificador>                 ::= <letra> | <identificador> <letra>
                                |   <identificador> <entero>

<opBinario>                     ::= "+" | "-" | "/" | "^" | "*" | "<" | "<="
                                |   ">" | ">=" | "y" | "o" | "oex" | "==" | "!=" |
                                     "%"

<opUnario>                      ::= "+" | "-" | "no"

<finSentencia>                  ::= ";"

/* Listas */
<lista>                         ::= "[" <contenidoLista> "]"

<contenidoLista>                ::= <expresion>
```

```
                                |   <contenidoLista> "," <expresion>

<sentenciaLista>                ::= <identificador> ">>" <finSentencia>
                                |   <identificador> "<<" <finSentencia>
                                |   "$" <identificador> <finSentencia>

<expresionLista>                ::=

/* Literales */
<literal>                       ::= <entero> | <real> | <booleano>
                                | <caracter> | <lista>

<entero>                        ::= <digito> | <entero> <digito>

<real>                          ::= <entero> "." <entero>

<booleano>                      ::= "Verdadero" | "Falso"

<caracter>                      ::= "'" <letra> "'" | "'" <digito> "'"

                                | "'" <caracterEspecial> "'"

<cadena>                        ::= "\"" <cadenaCruda> "\""

<cadenaCruda>                   ::= <letra>
                                | <cadenaCruda> <letra>
                                | <cadenaCruda> <digito>
                                | <cadenaCruda> <caracterEspecial>

/* Atomos */
<letra>                         ::= [a-z] | [A-Z]

<caracterEspecial>              ::= "_" | " " | "-"

<digito>                        ::= [0-9]
```

3. Definición de la semántica

3.1. Elementos básicos del lenguaje

El lenguaje de programación propuesto C+O- posee una estructura similar a C. Para empezarlo a escribir se debe declarar la cabecera del programa:

```
inicio
{
    <Bloque>
}
```

Dentro de este bloque estará el programa entero, el código fuente del programa.

Se entiende como bloque todo el código que se encuentra entre llaves permitiendo el lenguaje tener bloques de código anidado.

El lenguaje también posee expresiones, una expresión puede ser:

- Una variable única.
- Una operación entre una(o dos) variables.
- Una constante numérica o un carácter.
- Una operación entre constantes y variables.
- Una lista u operaciones de listas.

3.2. Variables

Dentro de cada bloque del programa, **solamente** está permitido declarar variables al inicio del mismo.

El lenguaje soporta los siguientes tipos de dato `entero`, `real`, `caracter`, `booleano` y `lista de` los tipos de datos anteriores aparte también se soportan operaciones unarias y binarias comunes a otros lenguajes. Una declaración de variables seguirá el siguiente esquema:

```
inivar
    <Tipo de dato> <variable> [, <variable> , ...];
    <...>
finvar;
```

No está permitido declarar y asignar directamente las variables, esto debe de realizarse posteriormente en el bloque, y se realiza de la misma manera que se realiza en `C`.

```
inivar
    entero goku, maya, bobobo;
    flotante kiryu, solaire;
    caracter pikachu;
    lista de booleano heidi;
finvar;

goku = 10;
kiryu = 20.30;
```

3.3. Funciones

Se deben declarar e implementar las funciones a utilizar dentro de ese bloque, si fuera necesario. La declaración e implementación será así:

```
<Tipo de dato> <Identificador> (<Tipo de dato> <variable> [, <Tipo de
   dato> <variable> , ...])
{
    <...>
    repatriar <Expresión>;
}
```

El lenguaje no soporta procedimientos, esto significa que los subprogramas **forzosamente** deben de retornar un valor. La palabra reservada para el retorno de valores es `repatriar`.

```
entero suma (entero a, entero b)
{
    repatriar a + b;
}
```

3.4. Cuerpo del Programa

Una vez se han declarado las variables principales y las funciones, se puede escribir el programa principal.

Las sentencias descritas a continuación son válidas tanto para el programa principal como para cualquier función, y se entiende que una sentencia puede ser además de que se describe continuación puede ser también un bloque de código.

3.4.1. Condicionales

Los condicionales se definen con la palabra reservada `si` y opcionalmente se puede añadir el `sino` para tratar el caso que no se cumpla la expresión.

```
si (<Expresión>) <Sentencia> [sino <Sentencia>]
```

```
si (a > b)
{
    a = 1500;
}
sino
{
    b = a - b;
}
```

3.4.2. Bucles

Los bucles de tipo "**`while`**" se escriben con la palabra reservada `mientras`

```
mientras (<Expresión>) <Sentencia>
```

```
mientras (b < 100)
{
    a = a + b + 5;
    b = b + 1;
}
```

Para los bucles tipo "**`for`**", se tienen las palabras reservadas `para` que indica el comienzo del bucle, `hasta` indica la condición de finalizacón y se puede controlar que los valores aumenten o disminuyan indicando `aumentando` o `decrementando`.

```
para <Variable> = <Expresión> hasta <Expresión> aumentando|
   decrementando <Sentencia>
```

```
para i = 1 hasta 10 aumentando
{
    a = a + i;
}
```

3.4.3. Entrada y Salida por Pantalla

Para la entrada de valores por pantalla se utiliza la palabra reservada `escanear`

```
escanear (<Tipo de Dato> <Identificador> [, <Tipo de dato> <
   Identificador>, ...]);
```

```
escanear (entero edad);
```

Para la salida de valores por pantalla se utiliza la palabra reservada `anunciar` y permite no solamente mostrar por pantalla expresiones, sino también cadenas de caracteres.

```
anunciar(<Expresión>|<Cadena> [, <Expresión>|<Cadena>, ...]);
```

```
anunciar("El valor de ", a, " sumado con ", b ," es ", c);
```

3.4.4. Listas

El lenguaje soporta listas como un tipo de dato elemental, y si bien se ha visto como se declaran anteriormente, también tienen su propio formato para realizar operaciones sobre ellas.

Para iterar por la lista se pueden usar los operadores $\gg$ y $\ll$ para avanzar y retroceder la posición del cursor y $ para moverlo al comienzo de la lista.

Además, tiene más métodos para acceder a información de la lista.

- **#**`lista`. Devuelve el número de elementos de la lista.
- `?lista`. Devuelve el elemento actual de la lista.
- `lista @ x`. Devuelve el elemento de la posición x.
- `lista ++ x @ z`. Devuelve una copia de la lista con x añadido en la posición z.
- `lista -- x`. Devuelve una copia de l con el elemento de la posición x borrado.
- `lista % x`. Devuelve una copia de l sin los elementos a partir de la posición x.
- `lista1 ** lista2`. Añade los elementos de lista2 en lista1 y devuelve una copia.
- `lista + x`. Suma de x con cada elemento.
- `x + lista`. Suma de cada elemento con x.
- `lista - x`. Resta de x con cada elemento de la lista.
- `lista * x`. Producto de x con cada elemento de la lista.
- `x * lista`. Producto de cada elemento de la isla con x.
- `lista / x`. División de x con cada elemento.

4. Identificación de los tokens

TOKEN	ATRIBUTOS	PATRÓN	COD
INIPROG		(“inicio”)	256
ABRPAR		(“(”)	257
CERPAR		(“)”)	258
INIVAR		(“inivar”)	259
FINVAR		(“finvar”)	260
ABRLLA		(“{”)	261
CERLLA		(“}”)	262
DEFLISTA		“lista de”	263
TIPODATO	0: entero, 1: real, 2: caracter, 3: booleano	(“entero” | “real” | “caracter” | “booleano”)	264
ASIG		(“=”)	265
IF		(“si”)	266
ELSE		(“sino”)	267
WHILE		(“mientras”)	268
SCAN		(“escanear”)	269
RETURN		(“repatriar”)	270
FOR		(“para”)	271
TO		(“hasta”)	272
SENTIDO	0: aumentando, 1:decrementando	(“aumentando” | “decrementando”)	273
COMA		(“,”)	274
PRINT		(“anunciar”)	275
PYC		(“;”)	276
MASMENOS	0: +, 1: −	(“+” | “(”−))	277
OPBIN	0: /, 1: ∧, 2: <, 3: <=, 4: >, 5: >=, 6: y, 7: o, 8: oex 9: ==, 10: ! =	, 11: % (”/” | ” ∧ ” | “<” | “<=” | “>” | “>=” | “y” | “o” | “oex” | “==” | “! =”, “%”)	278
OPUNARIO	0: no, 1: #, 2: ?	(“no” | “#” | “?”)	279
CONCAT		(“**”)	280
ARROBA		(“@”)	282
POR		(“*”)	283
INITER		(“$”)	284
ITER	0: ≫, 1: ≪	(“≪”, “≫”)	285
ABRCOR		(“[”)	286
CERCOR		(“]”)	287
LITERAL		([0-9]+\.[0-9]+|[0-9]+|’.’| “Verdadero” | “Falso” | \’[^\’]\’)	288
CADENA		\”[^\”]*\”	289