Procesadores de Lenguajes

Práctica 3: Diseño del Lenguaje

UNIVERSIDAD
DE GRANADA

Grupo BAABB
Aparicio Martos, Francisco José: pacoam@correo.ugr.es
Bedmar López, Pedro: pedrobedmar@correo.ugr.es
Carmona Segovia, Mario: mcs2000carmona@correo.ugr.es
Díaz Bustos, Victor: victordiazb@correo.ugr.es

Índice

Práctica 2

Introducción

El objetivo de esta práctica consiste en diseñar un lenguaje que cumpla con las características indicadas en el guión. Especialmente queremos destacar aquellas que son específicas de nuestro grupo (BAABB):

- Sintaxis similar a la del lenguaje C.
- Palabras reservadas en español.
- Listas como estructura de datos.
- Uso de procedimientos.
- Implementación de bucles for como estructura de control adicional.

Tal y como se detalla a continuación en el BNF, nuestra implementación cuenta con enteros, reales, booleanos, listas y caracteres como tipos y estructuras de datos; ofreciendo además un conjunto de operaciones unarias, binarias y ternarias que trabajan sobre ellos.

También incluye estructuras if-else, bucles while y for, operadores de entrada y salida y procedimientos.

BNF del lenguaje

<programa>	::=	<cabecera_programa> <bloque>
<cabecera_programa>	::=	principal ()
<bloque>	::=	<inicio_de_bloque> <declar_de_variables_locales> <declar_proced> <sentencias> <fin_de_bloque>
<inicio_de_bloque>	::=	{
<fin_de_bloque>	::=	}
<lista_parametros>	::= | |	<lista_parametros> , <lista_para_por_defecto> <lista_parametros> , <parametro> <parametro>
<lista_para_por_defecto>	::= | | |	<lista_para_por_defecto> , <parametro> = <constante> <parametro> = <constante> <lista_para_por_defecto> , <parametro> = <agregado_lista> <parametro> = <agregado_lista>
<parametro>	::=	<tipos> ID
<declar_de_variables_locales>	::= |	<marca_ini_declar_variables> <variables_locales> <marca_fin_declar_variables>
<marca_ini_declar_variables>	::=	inicio_var
<marca_fin_declar_variables>	::=	fin_var
<variables_locales>	::= |	<variables_locales> <cuerpo_declar_variables> <cuerpo_declar_variables>
<cuerpo_declar_variables>	::=	<tipos> <declar_variables> ;
<declar_variables>	::= | | |	ID ID = <expresion> ID, <declar_variables> ID = <expresion>, <declar_variables>
<declar_proced>	::= |	<cabecera_proced> <bloque>
<cabecera_proced>	::= |	procedimiento ID (<lista_parametros>) procedimiento ID ()
<sentencias>	::=	<sentencias> <sentencia>

	|	<sentencia>
<sentencia>	::=	<bloque>
	|	<sentencia_asignacion>
	|	<sentencia_if>
	|	<sentencia_while>
	|	<sentencia_for>
	|	<sentencia_entrada>
	|	<sentencia_salida>
	|	<llamada_proced>
	|	<sentencia_avanzar_lista>
	|	<sentencia_retro_lista>
	|	<sentencia_al_comi_lista>
<sentencia_asgnacion>	::=	<identificador> = <expresion> ;
<sentencia_if>	::=	si (<expresion>) <sentencia>
	|	si (<expresion>) <sentencia> en otro caso <sentencia>
<sentencia_while>	::=	mientras (<expresion>) <sentencia>
<sentencia_for>	::=	para <sentencia_asignacion> hasta <expresion> iterando <expresion> hacer <sentencia>
<sentencia_entrada>	::=	leer <lista_identificadores> ;
<lista_identificadores>	::=	<lista_identificadores> , <identificador>
	|	<identificador>
<sentencia_salida>	::=	imprimir <mensajes> ;
<mensajes>	::=	<mensajes> , <mensaje>
	|	<mensaje>
<mensaje>	::=	<expresion>
	|	<cadena>
<cadena>	::=	\"[^\"]*\"
<llamada_proced>	::=	<identificador> (<lista_expresiones>);
	|	<identificador> ();
<lista_expresiones>	::=	<lista_expresiones>, <expresion>
	|	<expresion>
<sentencia_avanzar_lista>	::=	<expresion> >> ;
<sentencia_retro_lista>	::=	<expresion> << ;
<sentencia_al_comi_lista>	::=	$ <expresion> ;
<expresion>	::=	(<expresion>)
	|	<op_unario_izq> <expresion>
	|	<expresion> <op_unario_der>
	|	<expresion> <op_binario> <expresion>

	|	<expresion> <op_ternario_1> <expresion> <op_ternario_2> <expresion>
	|	<identificador>
	|	<constante>
	|	<agregado_lista>
<op_unario_izq>>	::=	+
	|	-
	|	++
	|	--
	|	no
	|	#
	|	?
<op_unario_der>	::=	++
		--
<op_binario>	::=	+
	|	-
	|	*
	|	/
	|	==
	|	!=
	|	<
	|	<=
	|	>
	|	>=
	|	y
	|	o
	|	o_exclusiva
	|	@
	|	--
	|	%
	|	**
<op_ternario_1>	::=	++
<op_ternario_2>	::=	@
<identificador>	::=	([a-z]|[A-Z])+([a-z]|[A-Z]|[0-9]|_)*
<constante>	::=	([1-9]([0-9])* | 0)
	|	([1-9]([0-9])* | 0).([0-9])+
	|	(“verdadero” | “falso”)
	|	(‘[^\”’]‘)
<agregado_lista>	::=	[<lista_expresiones>]
<tipos>	::=	<tipo_basico>
	|	lista de <tipo_basico>
<tipo_basico>	::=	entero
	|	real
	|	booleano
	|	caracter

Descripción en lenguaje natural

Nuestro lenguaje funciona utilizando diferentes estructuras, cada una con una jerarquía dentro de este.

Programas y bloques

El programa es la estructura más alta dentro de la jerarquía. Representa el punto de inicio de la ejecución. La palabra clave que lo representa es `principal`, almacenando su información en un bloque `{}`:

```
principal () {
	...
}
```

Dentro del bloque se declaran primero variables locales y a continuación procedimientos y sentencias.

Declaración de variables locales

Para declarar una variable local es necesario indicar en primer lugar el tipo de dato, después el identificador de la variable y por último se puede añadir opcionalmente una asignación de valor.

```
entero n1, n2, n3 = 3 * 2;
booleano b;
caracter c = 'r';
real r1 = 1.2, r2 = 2.3;
lista de entero l1, l2;
```

Declaración de procedimientos

La declaración de un procedimiento comienza con la palabra reservada “procedimiento”, seguida del identificador del procedimiento y de una lista de parámetros que se encuentra entre paréntesis, cada parámetro debe indicar a qué tipo básico pertenece; y finalmente se coloca un bloque.

```
procedimiento p (entero x, entero y, real z) {
	...
}
```

Sentencias

Las sentencias son estructuras que nos permiten implementar algoritmos sobre unas variables. En esta sección repasamos la sintaxis que siguen en nuestro lenguaje.

Sentencia de asignación

Almacenar el valor de una expresión en una variable.

```
n2 = 2.3;
l2 = [2, 4, 17];
```

Sentencia condicional if

Estructura condicional. Puede ir acompañada de un else. La condición puede ser cualquier expresión que se pueda formar con el lenguaje.

```
si (n1 == 2) {
	...
}

si (r1 != n2)
	...
en otro caso
	...

si (a%2 == 0)
	a /= 2;
else
	a /= 3;
```

Sentencia condicional while

Bucle condicional. La condición puede ser cualquier expresión que se pueda formar con el lenguaje.

```
mientras (b == verdadero) {
	...
}
```

Sentencia condicional for

Bucle condicional que recorre una serie de valores, empezando por un valor que se asigna a la variable local que recorre la serie de valores y terminando cuando se deja de cumplir la condición. La condición puede ser cualquier expresión que se pueda formar con el lenguaje.

```
para entero i = 0 hasta i < 10 iterando i++ hacer {
	...
}
```

Sentencia de entrada

Lee valores por teclado y los almacena en las variables indicadas.

```
leer n1, n2, n3;
```

Sentencia de salida

Muestra por pantalla cadenas de texto y expresiones.

```
imprimir “Hola, el resultado es: “, c;
```

Llamada a procedimiento

Para llamar a un procedimiento es necesario usar el identificador de dicho procedimiento y pasarle el valor de las variables con las que vaya a funcionar

```
p (n1, n2, r2);
```

Sentencia de avance en la lista

Para poder avanzar posiciones en una lista se deberá de usar la siguiente forma:

```
l2 >>;
```

Sentencia de retroceso en la lista

Para poder retroceder posiciones en una lista se deberá de usar la siguiente forma:

```
l2 <<;
```

Sentencia de regreso al comienzo de la lista

Para situarse al comienzo de una lista se usará el dólar junto al nombre de la lista

```
$ l2;
```

Agregado

El agregado será un conjunto de datos del mismo tipo que se declarará de la siguiente forma:
[1, 3, 6, 2]

Tabla de tokens

Palabras	Tokens
principal	{principal}
{	{ { }
}	{ } }
,	{,}
lista de	{lista de}
procedimiento	{procedimiento}
(	{ (}
)	{) }
[	{ [}
]	{] }
=	{=}
inicio_var	{inicio_var}
fin_var	{fin_var}
si	{si}
en otro caso	{en otro caso}
mientras	{mientras}
para	{para}
hasta	{hasta}
iterando	{iterando}
hacer	{hacer}
leer	{leer}
imprimir	{imprimir}
cadena	{cadena}
identificador	{identificador}
<<	{<<, >>}
>>	{$}
$	{;}
;	{++}
++	{--}
--	{no, #, ?}
no	{+, -}
#	{*,/,**,%,==,!=,<,<=,>,>=,y, o, o_exclusiva}
?	{@}
+	{entero, real, booleano, caracter}
-	{constante_entero, constante_real, constante_booleano, constante_caracter, agregado}
*	
/	
==	
!=	
<	
<=	

>
>=
y
o
o_exclusiva
@
%
**
entero
real
booleano
caracter
constante_entero
constante_real
constante_booleano
constante_caracter
agregado

Tokens	Atributos	Patrón	COD
PRINCIPAL		“principal”	257
LLAVEIZQ		“{“	258
LLAVEDER		“}“	259
COMA		“,”	260
LISTADE		“lista de”	261
PROCEDIMIENTO		“procedimiento”	262
PARDER		“)“	263
PARIZQ		“(”	264
CORCHIZQ		“[“	265
CORCHDER		“]”	266
IGUAL		“=”	267
INICIOVAR		“inicio_var”	268
FINVAR		“fin_var”	269

SI		“si”	270
MIENTRAS		“mientras”	271
OTROCASO		“en otro caso”	272
PARA		“para”	273
HASTA		“hasta”	274
ITERANDO		“iterando”	275
HACER		“hacer”	276
LEER		“leer”	277
IMPRIMIR		“imprimir”	278
CADENA		\”[^\”]*\”	279
ID		([a-z]|[A-Z])+([a-z]|[A-Z]|[0-9]|_)*	280
RETROCEDER	0: << 1: >>	(“<<” | “>>”)	281
DOLLAR		“$”	282
PYC		“;”	283
OP1		“++”	284
OP2		“--”	285
OP3	0: no 1: # 2: ?	(“no” | ”#” | ”?”)	286
OP4	0: + 1: -	(“+” | “-”)	287
OP5	0: * 1: / 2: ** 3: % 4: == 5: != 6: < 7: <= 8: > 9: >= 10: y 11: o 12: o_exclusiva	(”*” | ”/” | ”**” | ”%” | “==” | “!=” | “<” | “<=” | “>” | “>=” | “y” | “o” | “o_exclusiva”)	288
OP6		“@”	289

TIPOS	0: entero 1: real 2: booleano 3: caracter	(“entero” | “real” | “booleano” | “caracter”)	290
CONSTANTE	0: constante_entero 1: constante_real 2: constante_booleano 3: constante_caracter	(([1-9]([0-9])* | 0) | ([1-9]([0-9])* | 0).([0-9])+ | (“verdadero” | “falso”) | (‘[^\”’]‘))	291

Práctica 3

Introducción

En esta práctica adaptamos las reglas de nuestra gramática de forma que no generen errores en el proceso de análisis sintáctico. Se entrega la tabla de tokens y la gramática abstracta con estas modificaciones.

Además, implementamos estas reglas en el analizador sintáctico YACC.

Gramática abstracta del lenguaje

<programa>	::=	<cabecera_programa> <bloque>
<cabecera_programa>	::=	PRINCIPAL PARIZQ PARDER
<bloque>	::=	LLAVEIZQ <declar_de_variables_locales> <declar_proced> <sentencias> LLAVEDER
<lista_parametros>	::= | |	<lista_parametros> COMA <lista_para_por_defecto> <lista_parametros> COMA <parametro> <parametro>
<lista_para_por_defecto>	::= | | |	<lista_para_por_defecto> COMA <parametro> IGUAL CONSTANTE <parametro> IGUAL CONSTANTE <lista_para_por_defecto> COMA <parametro> IGUAL <agregado_lista> <parametro> IGUAL <agregado_lista>
<parametro>	::=	<tipos> ID
<declar_de_variables_locales>	::= |	INICIOVAR <variables_locales> FINVAR
<variables_locales>	::= |	<variables_locales> <cuerpo_declar_variables> <cuerpo_declar_variables>
<cuerpo_declar_variables>	::=	<tipos> <declar_variables> PYC
<declar_variables>	::= | | |	ID ID IGUAL <expresion> <declar_variables> COMA ID <declar_variables> COMA ID IGUAL <expresion>
<declar_proced>	::= |	<cabecera_proced> <bloque>
<cabecera_proced>	::= |	PROCEDIMIENTO ID PARIZQ <lista_parametros> PARDER PROCEDIMIENTO ID PARIZQ PARDER
<sentencias>	::= |	<sentencias> <sentencia> <sentencia>
<sentencia>	::= | | | | |	<bloque> <sentencia_asignacion> <sentencia_if> <sentencia_while> <sentencia_for> <sentencia_entrada>

	|	<sentencia_salida> <llamada_proced> <expresion> RETROCEDER PYC <sentencia_al_comi_lista>
<sentencia_asignacion>	::=	ID IGUAL <expresion> PYC
<sentencia_if>	::= |	SI PARIZQ <expresion> PARDER <sentencia> SI PARIZQ <expresion> PARDER <sentencia> OTROCASO <sentencia>
<sentencia_while>	::=	MIENTRAS PARIZQ <expresion> PARDER <sentencia>
<sentencia_for>	::=	PARA <sentencia_asignacion> HASTA <expresion> ITERANDO <expresion> HACER <sentencia>
<sentencia_entrada>	::=	LEER <lista_identificadores> PYC
<lista_identificadores>	::= |	<lista_identificadores> COMA ID ID
<sentencia_salida>	::=	IMPRIMIR <mensajes> PYC
<mensajes>	::= |	<mensajes> COMA <mensaje> <mensaje>
<mensaje>	::= |	<expresion> CADENA
<llamada_proced>	::= |	ID PARIZQ <lista_expresiones> PARDER PYC ID PARIZQ PARDER PYC
<lista_expresiones>	::= |	<lista_expresiones> COMA <expresion> <expresion>
<sentencia_al_comi_lista>	::=	DOLLAR <expresion> PYC
<expresion>	::= | | | | | | |	PARIZQ <expresion> PARDER <op_unario_izq> <expresion> <expresion> <op_unario_der> <expresion> <op_binario> <expresion> <expresion> OP1 <expresion> OP6 <expresion> ID CONSTANTE <agregado_lista>
<op_unario_izq>	::= | | |	OP4 OP1 OP2 OP3
<op_unario_der>	::=	OP1 OP2

<op_binario>	::= | | |	OP4 OP5 OP6 OP2
<agregado_lista>	::=	CORCHIZQ <lista_expresiones> CORCHDER
<tipos>	::= |	TIPOS LISTADE TIPOS

Tabla de tokens con las modificaciones sintácticas

Tokens	Atributos	Patrón	COD
PRINCIPAL		“principal”	257
LLAVEIZQ		“{“	258
LLAVEDER		“}“	259
COMA		“,”	260
LISTADE		“lista de”	261
PROCEDIMIENTO		“procedimiento”	262
PARDER		“)“	263
PARIZQ		“(”	264
CORCHIZQ		“[“	265
CORCHDER		“]”	266
IGUAL		“=”	267
INICIOVAR		“inicio_var”	268
FINVAR		“fin_var”	269
SI		“si”	270
MIENTRAS		“mientras”	271
OTROCASO		“en otro caso”	272
PARA		“para”	273
HASTA		“hasta”	274
ITERANDO		“iterando”	275
HACER		“hacer”	276
LEER		“leer”	277
IMPRIMIR		“imprimir”	278
CADENA		\”[^\”]*\”	279
ID		([a-z]|[A-Z])+([a-z]|[A-Z]|[0-9]|_)*	280
MOV_LISTA	0: << 1: >>	(“<<” | “>>”)	281

DOLLAR		“$”	282
PYC		“;”	283
TIPOS	0: entero 1: real 2: booleano 3: caracter	(“entero” | “real” | “booleano” | “caracter”)	284
CONSTANTE	0: constante_entero 1: constante_real 2: constante_booleano 3: constante_caracter	(([1-9]([0-9])* | 0) | ([1-9]([0-9])* | 0).([0-9])+ | (“verdadero” | “falso”) | (‘[^\”’]‘))	285
LLAVE_OP_TER		“^”	286
OR		“o”	287
AND		“y”	288
XOR		"o_exclusiva"	289
IGUALDAD	0: == 1: !=	(“==” | “!=”)	290
RELACION	0: < 1: <= 2: > 3: >=	(“<” | “<=” | “>” | “>=”)	291
ADITIVOS	0: + 1: -	(“+” | “-”)	292
MULTIPLICATIVOS	0: * 1: / 2: %	(“*” | “/” | “%”)	293
POTENCIAS		"**"	294
NOT		“no”	295
DECRE_PRE		“--”	296
INCRE_PRE		“++”	297
UNARIO_PRE_LISTA		"#"	298
ELEM_POSI		"@"	299

Gramática abstracta del lenguaje con las modificaciones sintácticas

<programa>	::=	<cabecera_programa> <bloque>
<cabecera_programa>	::=	PRINCIPAL PARIZQ PARDER
<bloque>	::=	LLAVEIZQ <declar_de_variables_locales> <declar_proced> <sentencias> LLAVEDER
	|	LLAVEIZQ <declar_de_variables_locales> <sentencias> LLAVEDER
<lista_parametros>	::=	<lista_parametros> COMA <parametro>
	|	<parametro>
<lista_para_por_defecto>	::=	<lista_para_por_defecto> COMA <parametro> IGUAL CONSTANTE
	|	<parametro> IGUAL CONSTANTE
	|	<lista_para_por_defecto> COMA <parametro> IGUAL <agregado_lista>
	|	<parametro> IGUAL <agregado_lista>
<parametro>	::=	<tipos> ID
<declar_de_variables_locales>	::=	INICIOVAR <variables_locales> FINVAR
	|	
<variables_locales>	::=	<variables_locales> <cuerpo_declar_variables>
	|	<cuerpo_declar_variables>
<cuerpo_declar_variables>	::=	<tipos> <declar_variables> PYC
	|	error
<declar_variables>	::=	ID
	|	ID IGUAL <expresion>
	|	<declar_variables> COMA ID
	|	<declar_variables> COMA ID IGUAL <expresion>
<declar_proced>	::=	<cabecera_proced> <bloque>
<cabecera_proced>	::=	PROCEDIMIENTO ID PARIZQ <lista_parametros> COMA <lista_para_por_defecto> PARDER
	|	PROCEDIMIENTO ID PARIZQ <lista_parametros> PARDER
	|	PROCEDIMIENTO ID PARIZQ PARDER
	|	error
<sentencias>	::=	<sentencias> <sentencia>
	|	<sentencia>

<sentencia>	::=	<bloque>
	|	<sentencia_asignacion>
	|	<sentencia_if>
	|	MIENTRAS PARIZQ <expresion> PARDER <sentencia>
	|	PARA <sentencia_asignacion> HASTA <expresion> ITERANDO <expresion> HACER <sentencia>
	|	LEER <lista_identificadores> PYC
	|	IMPRIMIR <mensajes> PYC
	|	<llamada_proced>
	|	<expresion> RETROCEDER PYC
	|	DOLLAR <expresion> PYC
<sentencia_asignacion>	::=	ID IGUAL <expresion> PYC
<sentencia_if>	::=	SI PARIZQ <expresion> PARDER <sentencia>
	|	SI PARIZQ <expresion> PARDER <sentencia> OTROCASO <sentencia>
<lista_identificadores>	::=	<lista_identificadores> COMA ID
	|	ID
<mensajes>	::=	<mensajes> COMA <mensaje>
	|	<mensaje>
<mensaje>	::=	<expresion>
	|	CADENA
<llamada_proced>	::=	ID PARIZQ <lista_expresiones> PARDER PYC
	|	ID PARIZQ PARDER PYC
<lista_expresiones>	::=	<lista_expresiones> COMA <expresion>
	|	<expresion>
<expresion>	::=	DECRE_PRE <expresion>
	|	INCRE_PRE <expresion>
	|	NOT <expresion>
	|	UNARIO_PRE_LISTA <expresion>
	|	ADITIVOS <expresion> UNARIOS
	|	<expresion> ADITIVOS <expresion>
	|	<expresion> DECRE_PRE <expresion>
	|	<expresion> ELEM_POSI <expresion>
	|	<expresion> MULTIPLICATIVOS <expresion>
	|	<expresion> POTENCIAS <expresion>
	|	<expresion> IGUALDAD <expresion>
	|	<expresion> RELACION <expresion>
	|	<expresion> OR <expresion>
	|	<expresion> AND <expresion>
	|	<expresion> XOR <expresion>
	|	LLAVE_OP_TER <expresion> INCRE_PRE <expresion> ELEM_POSI <expresion> LLAVE_OP_TER
	|	ID
	|	agregado_lista
	|	CONSTANTE
	|	error
<agregado_lista>	::=	CORCHIZQ <lista_expresiones> CORCHDER

<tipos>	::= |	TIPOS LISTADE TIPOS