PROCESADORES DE LENGUAJES

Práctica 2: Diseño del lenguaje con código BBAAA

Universidad de Granada

EQUIPO 1
INTEGRANTES:
Salvador Jesús Megías Andreu
Cristina Díaz García
Javier Rodríguez Rodríguez
Enrique Casale Linde

Granada, España.
Curso 2020/2021
Ingeniería Informática.

1. Descripción del lenguaje a desarrollar

La descripción del lenguaje se puede separar en dos partes bien definidas: la primera sería la parte común a todos los lenguajes asignados a todos los equipos de prácticas (común), y la otra sería las especificaciones o reglas propias asignadas a través del código de nuestro lenguaje (parte propia o específica).
Este código sería **"BBAAA"** en nuestro caso.

-Características

- **Parte común:**
  - Debe ser un subconjunto de un lenguaje de programación estructurado.
  - Los identificadores deben ser declarados antes de ser usados.
  - Tipos de datos: entero, real, carácter y booleano
    - entero y real (operaciones): suma, resta, producto, división, operaciones de relación.
    - booleano (operaciones): and, or, not, xor.
  - Todas las expresiones poseerán una sentencia de asignación
  - Permitirá expresiones aritméticas lógicas.
  - Tendrá una sentencia de entrada (que permita leer sobre una lista de identificadores) y otra de salida (que permita escribir una lista de expresiones y/o constantes de tipo cadena).
  - .Dispone de las estructuras de control:
    - IF-THEN-ELSE.
    - WHILE.
  - La estructura sintáctica del programa será:
    - <Programa> ::= <Cabecera_programa> <bloque>
  - Se pueden definir bloques como en C.
  - Se permite anidamiento de bloques.
  - La comprobación de tipos será fuertemente tipado.
  - En los argumentos de un subprograma los parámetros se pasan por valor.
  - No se permiten declaraciones fuera de los bloques (deben ir entre una marca de inicio y otra de final.

- **Parte específica:**
  - Lenguaje C.
  - Palabras reservadas en inglés.
  - Estructura de datos considerada como tipo elemental:
    - Listas con las operaciones para manejo de listas.
  - Subprogramas:
    - Funciones.
  - Estructuras de control adicional:
    - case/switch.

2. Descripción formal del lenguaje usando BNF “BBAAA”

<Programa>	::=	<Cabecera_programa> <Bloque>
<Bloque>	::=	<Inicio_de_bloque> <Declar_de_variables_locales> <Declar_de_subprogs> <Sentencias> <Fin_de_bloque>
<Declar_de_subprogs>	::=	<Declar_de_subprogs> <Declar_subprog>
<Declar_subprog>	::=	<Cabecera_subprog> <Bloque>
<Declar_de_variables_locales>	::=	<Marca_ini_declar_variables> <Variables_locales> <Marca_fin_declar_variables>
<Marca_ini_declar_variables>	::=	var
<Marca_fin_declar_variables>	::=	endvar
<Cabecera_programa>	::=	<Tipo> main()
<Inicio_de_bloque>	::=	{
<Fin_de_bloque>	::=	}
<Variables_locales>	::= |	<Variables_locales> <Cuerpo_declar_variables> <Cuerpo_declar_variables>
<Cuerpo_declar_variables>	::=	<Tipo> <Lista_identificador>;
<Lista_identificador>	::= |	<Lista_identificador>, <Identificador> <Identificador>
<Cabecera_subprog>	::=	<Tipo> <Identificador> (<Lista_parametros>)
<Lista_parametros>	::= |	<Lista_parametros> , <Parametros> <Parametros>
<Parametros>	::=	<Tipo> <Identificador>
<Sentencias>	::= |	<Sentencias>, <Sentencia> <Sentencia>

<Sentencia>	::=	<Bloque> | <Sentencia_asignacion> | <Sentencia_if> | <Sentencia_while> | <Sentencia_entrada> | <Sentencia_salida> | <Sentencia_return> | <Sentencia_switch>
<Sentencia_asignacion>	::=	<Identificador> = <Expresion>;
<Sentencia_if>	::=	if (<expresion>) <Sentencia>; | if (<expresion>) <Sentencia> else <Sentencia>;
<Sentencia_while>	::=	while (<expresion>) <Sentencia>;
<Sentencia_entrada>	::=	read <Lista_variables>;
<Sentencia_salida>	::=	write <Lista_expresiones_o_cadena>;
<Sentencia_return>	::=	return <expresion>;
<Sentencia_switch>	::=	switch <Bloque_switch>
<Bloque_switch>	::=	<Inicio_de_bloque> | <Sentencias_case> | <Fin_de_bloque>
<Sentencias_case>	::=	case <Constante> : | <Bloque> | break | <Sentencias_case> | default: | <Bloque> | break | <Sentencias_case> |
<Lista_variables>	::=	<Lista_variables> , <variable> <Variable>
<Variable>	::=	<Identificador> | <Identificador> [<Numero>] | <Identificador> [<Numero>][<Numero>]
<Lista_expresiones_o_cadena>	::=	<Lista_expresiones_o_cadena>, <Expresion> | <Lista_expresiones_o_cadena> , <Cadena> | <Expresion> | <Cadena>
<Expresion>	::=	(<Expresion>) | <Op_unario> <Expresion>

	|	<Expresion> <Op_binario> <Expresion>
	|	<Expresion> <Op_ternario1> <Expresion> <Op_ternario2> <Expresion>
	|	<Identificador>
	|	<Funcion>
	|	<Constantes>
	|	<Constante_lista>
<Cadena>	::=	“[a-zA-Z0-9_]*”
<Op_signo>	::=	+
	|	-
<Op_unario>	::=	!
	|	~
	|	&
	|	#
	|	?
	|	+
	|	-
<Op_binario>	::=	*
	|	/
	|	%
	|	>
	|	<
	|	==
	|	!=
	|	>=
	|	<=
	|	&&
	|	||
	|	@
	|	--
	|	**
<Op_ternario1>	::=	++
<Op_ternario2>	::=	@
<Op_real>	::=	.
<Identificador>	::=	<Identificador> <Letra>
	|	<Identificador> <Numero>
	|	<Letra>
<Constantes>	:=	<Constante_entera>
	|	<Constante_booleana>
	|	<Constante_char>
	|	<Constante_float>

<Constante_entera>	::=	<Numero>
<Constante_booleana>	::=	<Tipo_bool>
<Constante_char>	::=	\'[^\']\'
<Constante_float>	::= |	<Decimal> <Numero><Decimal>
<Constante_lista>	::=	‘[‘ “[“<Lista_enteros>”]” | “[“<Lista_booleanos>”]” | “[“<Lista_reales>”]” | “[“<Lista_char>”]” ’]’;
<Lista_enteros>	::= |	<Lista_enteros> , <Constante_entera> <Constante_entera>;
<Lista_booleanos>	::= |	<Lista_booleanos> , <Constante_booleana> <Constante_booleana>;
<Lista_float>	::= |	<Lista_reales> , <Constante_float> <Constante_float>;
<Lista_char>	::= |	<Lista_char> , <Constante_char> <Constante_char>;
<Decimal>	::=	<Op_real><Numero>
<Letra>	::=	[a-zA-Z]
<Numero>	::= |	<Numero>[0-9] [0-9]
<Tipo_bool>	::= |	true false
<Funcion>	::= |	<Identificador> (<Lista_expresiones>) <Identificador> ()
<Lista_expresiones>	::=	<Lista_expresiones>, <expresion>
<Tipo>	::= |	<Tipo_elem> <Tipo_lista>
<Tipo_elem>	::= | | |	int float char bool
<Tipo_lista>	::=	list of <Tipo_elem>

3. Definición de la semántica

por rellenar

4. Identificación de los tokens

Tokens	Código	Expresión	Atributo
MAIN	500	main	
BEGIN_BLOCK	501	{	
END_BLOCK	502	}	
SEMICOLON	503	;	
COMMA	504	,	
LEFT_PARENTHE	505	(	
RIGHT_PARENTHE	506	)	
ASSIGNMENT	507	=	
IF	508	if	
ELSE	509	else	
WHILE	510	while	
SWITCH	511	switch	
CASE	512	case	
RETURN	513	return	
ELEM_TYPE	514	int bool char float	0:int 1:bool 2:char 3:float
MODIFIER	515	const	
INPUT	516	read	
OUTPUT	517	write	
SIGN	518	+ -	0: + 1: -
UNARY_OP	519	! ~ & # ?	0: ! 1: ~ 2: & 3: # 4: ?

BINARY_OP	520	* / % > < == != >= <= && || @ -- **	0: * 1: / 2: % 3: > 4: < 5: == 6: != 7: >= 8: <= 9: && 10: || 11: @ 12: -- 13: **
LOGIC_VAR	521	true false	0: true 1: false
INT_VAR	522	[0-9]+	
REAL_VAR	523	[0-9]+\. [0-9]*	
CHAR_VAR	524	\'[^\']\'	
LISTING	525	list of	
CHAIN	526	\"[^\"]+\"	
IDENTIFIER	527	[a-zA-Z]([a-zA-Z]|[0-9]|_)*	

5. Análisis léxico

- **Definición de “constantes.h”**

```
#ifndef __CONSTANTES_H
#define __CONSTANTES_H

#define MAIN 500
#define BEGIN_BLOCK 501
#define END_BLOCK 502
#define SEMICOLON 503
#define COMMA 504
#define LEFT_PARENTHE 505
#define RIGHT_PARENTHE 506
#define ASSIGNMENT 507
#define IF 508
#define ELSE 509
#define WHILE 510
#define SWITCH 511
#define CASE 5012
#define RETURN 513
#define ELEM_TYPE 514
#define MODIFIER 515
#define INPUT 516
#define OUTPUT 517
#define SIGN 518
#define UNARY_OP 519
#define BINARY_OP 520
#define LOGIC_VAR 521
#define INT_VAR 522
#define REAL_VAR 523
#define CHAR_VAR 524
#define LISTING 525
#define CHAIN 526
#define IDENTIFIER 527

#endif
```

- **Definición de archivo LEX “practica2.l” (incompleto)**

```
%{

    // Archivos a usar

    #include <stdlib.h>
    #include <string.h>
    #include "constantes.h"


%}

%option yylineno
%option noyywrap

letra [a-zA-Z]
digito [0-9]
entero {digito}+
real {entero}.{entero}

%%

"main" {
    ECHO;
    return(MAIN);
}

"{" {
    ECHO;
    return(BEGIN_BLOCK);
}

"}" {
    ECHO;
    return(END_BLOCK);
}

";" {
    ECHO;
    return(SEMICOLON);
}
```

```
"," {
    ECHO;
    return(COMMA);
}

"(" {
    ECHO;
    return(LEFT_PARENTHE);
}

")" {
    ECHO;
    return(RIGHT_PARENTHE);
}

"=" {
    ECHO;
    return(ASSIGNMENT);
}

"if" {
    ECHO;
    return(IF);
}

"else" {
    ECHO;
    return(ELSE);
}

"while" {
    ECHO;
    return(WHILE);
}

"switch" {
    ECHO;
    return(SWITCH);
}

"case" {
    ECHO;
    return(CASE);
```

```
}

"return" {
    ECHO;
    return(RETURN);
}

"int" {
    ECHO;
    return(ELEM_TYPE);
}

"bool" {
    ECHO;
    return(ELEM_TYPE);
}

"char" {
    ECHO;
    return(ELEM_TYPE);
}

"float" {
    ECHO;
    return(ELEM_TYPE);
}

"const" {
    ECHO;
    return(MODIFIER);
}

"read" {
    ECHO;
    return(INPUT);
}

"write" {
    ECHO;
    return(OUTPUT);
}

"+" {
```

```
    ECHO;
    return(SIGN);
}

"-" {
    ECHO;
    return(SIGN);
}

"!" {
    ECHO;
    return(UNARY_OP);
}

"~" {
    ECHO;
    return(UNARY_OP);
}

"&" {
    ECHO;
    return(UNARY_OP);
}

"#" {
    ECHO;
    return(UNARY_OP);
}

"?" {
    ECHO;
    return(UNARY_OP);
}

"*" {
    ECHO;
    return(BINARY_OP);
}

"/" {
    ECHO;
    return(BINARY_OP);
}
```

```
"%" {
    ECHO;
    return(BINARY_OP);
}

">" {
    ECHO;
    return(BINARY_OP);
}

"<" {
    ECHO;
    return(BINARY_OP);
}

"==" {
    ECHO;
    return(BINARY_OP);
}

"!=" {
    ECHO;
    return(BINARY_OP);
}

">=" {
    ECHO;
    return(BINARY_OP);
}

"<=" {
    ECHO;
    return(BINARY_OP);
}

"&&" {
    ECHO;
    return(BINARY_OP);
}

"||" {
    ECHO;
```

```
    return(BINARY_OP);
}

"@" {
    ECHO;
    return(BINARY_OP);
}

"--" {
    ECHO;
    return(BINARY_OP);
}

"**" {
    ECHO;
    return(BINARY_OP);
}

"true" {
    ECHO;
    return(LOGIC_VAR);
}

"false" {
    ECHO;
    return(LOGIC_VAR);
}

{entero} {
    ECHO;
    return(INT_VAR);
}

{real} {
    ECHO;
    return(REAL_VAR);
}

"\'[^\']\'" {
    ECHO;
    return(CHAR_VAR);
}
```

```
"list of" {
    ECHO;
    return(LISTING);
}

\"[^\"]+\" {
    ECHO;
    return(CHAIN);
}

[ \t] {
    ECHO;
}

[\n\r] {
    ECHO;
}

({letra}|_)({letra}|{digito}|_)* {
     ECHO;
     return(IDENTIFIER);
 }

. {
    printf(" \nError Léxico en la línea: %d, no se reconoce '%s'. ",
yylineno, yytext);
}

%%

int main (int argc, char** argv) {

    // Si no se envía un argumento (fichero) da error
    if (argc <= 1) {

        printf("\nError al ejecutar la aplicación...\n");
        printf("Uso: %s nombre_fichero_fuente\n", argv[0]);

        exit(-1);

    }

    // Abrimos el fichero
```

```
    yyin = fopen(argv[1], "r");

    // Si "yyin" es nulo no se ha podido abrir
    if (yyin == NULL) {

        printf ("\nError al abrir el fichero %s\n", argv[1]);

        exit (-2);

    }

    // Llamamos al analizador léxico para comenzar el análisis
    int an = yylex();

    // Ejecutamos hasta que terminemos de analizar todo el archivo
    while (an != 0) {

        printf("__%d__ ", an);
        an = yylex();

    }

    exit(1);

}
```

MAIN CREADO DE TESTADOR

```
main ()
{

    int a=0, b=1, c=2;
    float d=5.5;
    list of char mensaje;


    switch (a){
        case(a<0){
            write "mal"
        }
        case(a==0){
            write "bien"
        }
        case(a>0){
            write "mejor"
        }
    }

    if(true){
        read b;
    }
    else a=b;

    if(!b){
        if(a==c)
            c=2.3;
    }
}
```

RESULTADO EN FLEX DE ANALIZADOR LÉXICO