Práctica 1

Diseño del Lenguaje

Valentín Pérez Carrasco

Juan José Segura González

Antonio M. Martín Jiménez

1. Introducción. Breve descripción del lenguaje asignado.

Para la realización de las prácticas se nos ha encargado el desarrollo de un lenguaje, éste debe de tener como lenguaje base **C**, es decir que tomaremos las reglas sintácticas usadas por dicho lenguaje como referencia para las instrucciones del lenguaje nuevo por lo que deberemos respetar los requerimientos impuestos por el mismo.

Las palabras claves de nuestro lenguaje deberán estar en **castellano**.

Como estructura elemental, a parte de las conocidas (enteros, reales, booleanos y caracteres) debemos de añadir **pilas**, junto con las operaciones de manejo de éstas. Tenemos que tener en cuenta que consideraremos las constantes de tipo pila.

Nuestro lenguaje deberá permitir la creación de **procedimientos** con el fin de poder estructurar nuestros programas.

Por último se nos dice que deberá poder definir estructuras de control tipo **switch/case**.

2. Descripción formal de la sintaxis del lenguaje usando BNF.

<Programa> ::= <Cabecera_programa> <bloque>

<bloque> ::= <Inicio_de_bloque>

<Variables_locales>

<Declar_de_subprogs>

<Sentencias>

<Fin_de_bloque>

<Declar_de_subprogs> ::= <Declar_de_subprogs> <Declar_subprog>

|<Declar_subprog>

<Declar_subprog> ::= <Cabecera_subprograma> <bloque>

<Variables_locales> ::= <Variables_locales> <Cuerpo_declar_variables>

|<Cuerpo_declar_variables>

<Cuerpo_declar_variables> ::= <proc> <tipo> <declar_variables> <fin_linea>

<declar_variables> ::= <id> |<id> <igualdad> |<id> <coma> <declar_variables>

<Cabecera_subprog> ::= <id> <parizq> <identificadores> <parder>

<Sentencias> ::= <Sentencias> <Sentencia> |<Sentencia>

<Sentencia> ::= <bloque>

|<sentencia_asignacion>

|<sentencia_if>

|<sentencia_while>

|<sentencia_entrada>

|<sentencia_salida>

|<llamada_proced>

|<sentencia_switch>

<sentencia_asignacion> ::= <id> <igualdad> <fin_linea>

<igualdad> ::= <igualdad> <igual> <id> |<igual> <expresion>

<sentencia_if> ::= <cond> <parizq> <expresion> <parder> <Sentencia>

|<cond> <parizq> <expresion> <parder> <Sentencia>
<sentencia_else>

|<sentencia_if> <cond> <parizq> <expresion> <parder>
<Sentencia> <sentencia_else>

<sentencia_else> ::= <else> <Sentencia>

<sentencia_while> ::= <mient> <parizq> <expresion> <parder> <Sentencia>

<sentencia_entrada> ::= <int> <identificadores> <fin_linea>

<sentencia_salida> ::= <imp> <lista_expresiones_o_letra> <fin_linea>

<lista_expresiones_o_letra> ::= <identificadores> | <cadena> | <letra>

<identificadores> ::= <identificadores> <coma> <id> |<id>

<llamada_proced> ::= <id> <parizq> <param_llamada> <parder> <fin_linea>

<param_llamada> ::= <expresion>

|<param_llamada> <coma> <expresion>

<sentencia_switch> ::= <comp> <parizq> <id> <parder> <sentencia_case>
<fin_bloque>

<sentencia_case> ::= <caso> <literal> <dospuntos>

<Sentencias>

<fin_caso> <fin_linea>

|<caso> <literal> <dospuntos>

<Sentencias>

<fin_caso> <fin_linea>

<sentencia_default>

|<sentencia_case>

<caso> <literal> <dospuntos>

<Sentencias>

<fin_caso> <fin_linea>

<sentencia_default> ::= <por_defecto> <dospuntos> <Sentencias>

<fin_caso> <fin_linea>

<expresion> ::= <parizq> <expresion> <parder>

|<op_unario> <expresion>

|<sum_res> <expresion>

|<expresion> <op_binario> <expresion>

|<expresion> <sum_res> <expresion>

|<id>

|<literal>

|<agregados_pila>

<agregados_pila> ::= <Inicio_de_bloque> <fin_de_bloque>

|<Inicio_de_bloque> <valores_pila> <fin_de_bloque>

<valores_pila> ::= <identificador> | <valores_pila> <coma> <identificador>

|<caracter> | <valores_pila> <coma> <caracter>

<literal> ::= <numero> | <caracter> | <v_f>

<caracter> ::= <comilla> <letra> <comilla> | <comilla> <digito> <comilla>

<id> ::= <letra> <identificador> | <letra>

<identificador> ::= <letra>

|<identificador> <letra>

|<identificador> <digito>

<numero> ::= <natural> | <natural> <punto> <natural> | <sum_res> <natural>

<cadena> ::= <comilla> <cad> <comilla>

<cad> ::= <numero> | <letra> | <cad> <numero> | <cad> <letra>

<v_f> ::= verdadero | falso

<cond> ::= si

<else> ::= en otro caso

<mient> ::= mientras

<int> ::= introducir

<imp> ::= imprimir

<comp> ::= comprobar

<caso> ::= caso

<fin_caso> ::= fin_caso

<dospuntos> ::= :

<Cabecera_programa> ::= inicio

<tipo> ::= entero | real | booleano | caracter

|pila entero | pila real | pila booleano | pila caracter

<natural> ::= <digito> | <digito> <natural>

<proc> ::= procedimiento

<por_defecto> ::= defecto

<punto> ::= .

<parizq> ::= (

<parder> ::=)

<Inicio_de_bloque> ::= {

<Fin_de_bloque> ::= }

<coma> ::= ,

<igual> ::= =

<comilla> ::= "

<fin_linea> ::= ;

<op_unario> ::= ++|--|!|&|#|@

<sum_res> ::= + | -

<op_binario> ::= *| / | > | < | >= | <= | && | || | ^ | == | %%

<letra> ::= A|B|..|Z|a|b|..|z

<digito> ::= 0|1|..|9

3. Definición de la semántica en lenguaje natural.

Con respecto a la semántica cabe destacar las operaciones que podemos realizar con las pilas ya que es un tipo compuesto que no suele estar definido como tipo base en los lenguajes con los que estamos acostumbrados a trabajar.

Operaciones con Pilas

Las operaciones con pilas (%%,&,#,!,*,+,-,/) se utilizan con las pilas como si de tipos elementales se tratasen. Las operaciones que nuestro lenguaje permite hacer sobre las pilas son las siguientes:

- Insertar: Inserta un elemento x en el tope de la pila p (P=P%%x)
- Sacar: Borra el tope de una pila p (P=&P)
- Tope: Devuelve el valor del tope de la pila p (x=#P)
- Vacía: Devuelve true si la pila está vacía (!P)

- Producto: Esta operación puede ser realizada de varias formas:
    - P=P1 * P2: Multiplica elemento a elemento las dos pilas. Para ello P1 y P2 deben tener el mismo tamaño.
    - P=P * x: Multiplica todos los elementos de P, por x.
    - a=x * P: Es el producto de x con todos los elementos de la pila P.

- Suma: Puede ser realizada de varias formas:
    - P=P1 + P2: Suma elemento a elemento las dos pilas. Para ello P1 y P2 deben tener el mismo tamaño.
    - P=P + x: Suma todos los elementos de P, con x.
    - a=x + P: Es la suma de x con todos los elementos de la pila P.

- Resta: Puede ser realizada de varias formas:
    - P=P1 - P2: Resta elemento a elemento las dos pilas. Para ello P1 y P2 deben tener el mismo tamaño.
    - P=P - x: Resta todos los elementos de P, con x.

- División: Esta operación puede ser realizada de varias formas:
    - P=P1 / P2: Divide elemento a elemento las dos pilas. Para ello P1 y P2 deben tener el mismo tamaño.
    - P=P / x: Divide todos los elementos de P, por x.

Cuando se aplique una operación aritmética a dos pilas, ambas deben de ser del mismo tipo base y deben tener el mismo número de elementos. Cuando se opera con un dato y una pila, ambos deben compartir el mismo tipo básico. Por último destacar que cuando realicemos una asignación debemos realizarla de la forma:

P1=P2;

donde P1 y P2 son pilas. Esa sentencia es válida si y sólo si ambas tienen el mismo tipo base. El resultado de dicha asignación hace que la pila P1 tenga los mismos elementos que P2

Para cualquier acceso a los elementos de la pila, debemos tener en cuenta que sólo podremos ver el tope de la misma en cada momento.

4. Identificación de los tokens con el máximo nivel de abstracción.

Token	Identificador	Patrón/Expresión Regular
IDENTIFICADOR	257	“cualquier carácter alfabético”{“cualquier carácter alfanumérico”}
LITERAL	258	([0-9])+|(([0-9])+. ([0-9])+)|[- +]([0-9])+| “cualquier carácter”|”true”|”false”
CADENA	259	“cualquier secuencia de caracteres”
OPBIN	260	{“*”|“/”|“>”|“<”|“>=”|“<=”|”&&”|“||”| “^”| “==”|“%%”}
OPUNA	261	{“++”|“—”|“!”|”@”|”#”|”&”}
SUMRES	262	{“+”|”-“}
FINLINEA	263	“;”
COMILLA	264	“ ” ”
IGUAL	265	“=”
COMA	266	“,”
INIBLOQUE	267	“{“
FINBLOQUE	268	“}”
PARIZQ	269	“(“
PARDER	270	“)”
PUNTO	271	“.”
TIPO	272	{“entero”|“real”|“booleano”|“carácter”|“pila entero”|“pila real”|“pila booleano”|”pila carácter”}
CABPROG	273	“inicio”
CASO	274	“caso”
DEFECTO	275	“defecto”
FINCASO	276	“fin_caso”

DOSPUNT	277	“:”
COMP	278	“comprobar”
IMP	279	“imprimir”
INT	280	“introducir”
MIENT	281	“mientras”
ELSE	282	“en otro caso”
COND	283	“si”
PROC	284	“proc”