**Universidad de Granada**

*DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMÁTICOS*

Práctica 1 – Procesadores de Lenguajes

**Diseño del Lenguaje**

**Autores**: Alexander Moreno Borrego

Carlos Jesus Fernandez Basso

Francisco Santolalla Quiñonero

**Correos:** [alexmobo@correo.ugr.es](mailto:alexmobo@correo.ugr.es)

[karloos@correo.ugr.es](mailto:karloos@correo.ugr.es)

[fransan@correo.ugr.es](mailto:fransan@correo.ugr.es)

**DNI:** 39906263-K

75927137-C

76439251-Q

**Profesor**: Salvador Villena Morales

Contenido

[Introducción 3](#_Toc353571633)

[Descripción formal 5](#_Toc353571634)

[Semántica en lenguaje natural 9](#_Toc353571635)

[Programa 9](#_Toc353571636)

[Declaración de variables 9](#_Toc353571637)

[Declaración de subprogramas (funciones) 9](#_Toc353571638)

[Sentencias del programa 9](#_Toc353571639)

[Identificación de los tokens 11](#_Toc353571640)

Introducción

El lenguaje a implementar será asignado por el profesor de prácticas y tendrá las siguientes características mínimas:

* Ser un subconjunto de un lenguaje de programación estructurado.
* Los identificadores debe ser declarados antes de ser usados.
* Los tipos de datos mínimos son: entero, real, carácter y booleano. Se definirán las operaciones típicas para cada uno de ellos, según se puede ver en la siguiente tabla:

|  |  |
| --- | --- |
| Tipo de dato | Operaciones |
| entero, real | suma, resta, producto, división, operaciones de relación |
| booleano | and, or, not, xor |

* Poseerá la sentencia de asignación para todos los tipos de expresiones.
* Permitirá expresiones aritméticas lógicas.
* Tendrá una sentencia de entrada y otra de salida (se utilizará como dispositivo de entrada el teclado y de salida la pantalla). Además, la sentencia de entrada deberá permitir leer sobre una lista de identificadores y la sentencia de salida deberá permitir escribir una lista de expresiones y/o constantes de tipo cadena. A diferencia de los lenguajes conocidos y usados como referencia, estas sentencias no representan llamada a subprograma.
* Dispone de las estructuras de control siguientes:
  1. IF-THEN-ELSE
  2. WHILE
* Con independencia del tipo de lenguaje asignado:
  1. La estructura sintáctica del programa es:

<Programa> ::= <Cabecera\_programa> <bloque>

* 1. En cualquier parte se podrán definir bloques como en C, es decir, tendremos una estructura sintáctica como la que se muestra a continuación:

<bloque> ::= <Inicio\_de\_bloque>

<Declar\_de\_variables\_locales>

<Declar\_de\_subprogs>

<Sentencias>

<Fin\_de\_bloque>

* 1. Una sentencia puede ser, un bloque, por lo que se permite el anidamiento de bloques y subprogramas.
  2. La comprobación de tipos será como la del Pascal, es decir, fuertemente tipado.
  3. Para los argumentos de un subprograma, el mecanismo de paso de parámetros es por valor.
  4. No se permiten declaraciones fuera de los bloques. Las declaraciones deben ir entre una marca de inicio y otra de final de las declaraciones1.
  5. La estructura sintáctica de un subprograma será el siguiente:

<Declar\_subprog> ::= <Cabecera\_subprograma> <bloque>

* 1. El lenguaje debe admitir tanto las letras mayúsculas como las minúsculas, exceptuando aquellos que tengan lenguaje C que ofrece sensibilidad en este sentido, mientras que para el caso de Pascal no sucede.
* El lenguaje debe incluir **cinco elementos nuevos**, además de los enumerados anteriormente:
  1. Sintaxis inspirada en un lenguaje de programación.
     + **Pascal.** fue diseñado y publicado para que los programadores pudieran aprenderlo fácilmente utilizando programación estructurada y estructuras de datos.

Se caracteriza por ser un lenguaje de programación **estructurado fuertemente tipado**. Lo que implica dos aspectos:

* El código está dividido en porciones fácilmente legibles (funciones o procedimientos).
* El tipo de dato de todas las variables debe ser declarado previamente para que su uso quede habilitado.
  1. Palabras reservadas.
     + **Inglés**
  2. Estructura de datos considerada como tipo elemental.
     + **Conjuntos** con las operaciones para manejo de conjuntos. Debe definirse la constante de tipo conjunto y un término para indicar el conjunto vacío.
  3. Subprogramas.
     + **Funciones.** A diferencia d los procedimientos, las funciones, devuelven un valor.
  4. Estructura de control adicional.
     + **FOR**

Descripción formal

La siguiente especificación corresponde con la descripción formal del lenguaje en BNF.

<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\_subprograma>

<bloque>

| ""

<Declar\_de\_variables\_locales> ::= <Marca\_ini\_declar\_variables>

<Variables\_locales>

<Marca\_fin\_declar\_variables>

|""

<Cabecera\_programa> ::= "program" <identificador> ";"

| ""

<Marca\_ini\_declar\_variables> ::= "var"

<Marca\_fin\_declar\_variables> ::= ""

<Inicio\_de\_bloque> ::= "begin"

<Fin\_de\_bloque> ::= "end"

<Variables\_locales> ::= <Variables\_locales> <Cuerpo\_declar\_variables>

| <Cuerpo\_declar\_variables>

<Cuerpo\_declar\_variables> ::= <lista\_variables> ": set of integer " ";"

| <lista\_variables> ": set of real " ";"

| <lista\_variables> ": set of character " ";"

| <lista\_variables> ": set of boolean " ";"

| <lista\_variables> ": integer " ";"

| <lista\_variables> ": real " ";"

| <lista\_variables> ": character " ";"

| <lista\_variables> ": boolean " ";"

<variable> ::= "," <identificador><variable>

|""

<Cabecera\_subprog> ::= "function" <identificador> "(" <lista\_argumentos> ")" ": set of integer " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": set of real " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": set of character " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": set of boolean " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": integer " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": real " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": character " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": boolean " ";"

<lista\_argumentos> ::= <lista\_argumentos> ";" <Cuerpo\_declar\_argumentos>

| <Cuerpo\_declar\_argumentos>

<Cuerpo\_declar\_argumentos> ::= <lista\_variables> ": set of integer"

| <lista\_variables> ": set of real"

| <lista\_variables> ": set of character"

| <lista\_variables> ": set of boolean"

| <lista\_variables> ": integer"

| <lista\_variables> ": real"

| <lista\_variables> ": character"

| <lista\_variables> ": boolean"

<Sentencias> ::= <Sentencias> ";" <Sentencia>

| <Sentencia>

<Sentencia> ::= <bloque>

| <sentencia\_asignacion>

| <sentencia\_if>

| <sentencia\_while>

| <sentencia\_for>

| <sentencia\_entrada>

| <sentencia\_salida

| <sentencia\_return>

<sentencia\_asignacion> ::= <identificador>":="<expresion>

<sentencia\_if> ::= "if" <expresion> "then" <Sentencia> "else" <Sentencia>

| "if" <expresion> "then" <Sentencia>

| "if" <expresion> "then" <Sentencia> <sentencia\_elseif> "else" <Sentencia>

<sentencia\_elseif> ::= "else if" <Sentencia> <sentencia\_elseif>

<sentencia\_while> ::= "while" <expresion> "do" <Sentencia>

<sentencia\_for> ::= "for" <identificador>":="<expresion> "to" <expresion> "do" <Sentencias>

<sentencia\_entrada> ::= <nomb\_entrada> <lista\_variables>

<lista\_variables> ::= <identificador>

| <identificador> "," <lista\_variables>

<nomb\_entrada> ::= "readln"

| "read"

<sentencia\_salida> ::= <nomb\_salida> <lista\_expresiones\_o\_cadena>

<lista\_expresiones\_o\_cadena> ::= <expresion>

| <lista\_expresiones\_o\_cadena> "," <expresion>

| """<frase>"""

| <lista\_expresiones\_o\_cadena> "," """<frase>"""

<frase> ::= <letra> <frase>

| <letra>

| <digito> <frase>

| <digito>

<nomb\_salida> ::= "writeln"

|"write"

<funcion> ::= <identificador> "(" <lista\_expresiones> ")"

<lista\_expresiones> ::= <expresion>

| <lista\_expresiones> "," <expresion>

<sentencia\_return> ::= <sentencia\_asignacion>

<expresion> ::= "("<expresion>")"

| "+"<expresion>

| "-"<expresion>

| <expresion>"+"<expresion>

| <expresion>"-"<expresion>

| <expresion>"\*"<expresion>

| <expresion>"/"<expresion>

| <expresion> ">" <expresion>

| <expresion> ">=" <expresión

| <expresion> "<" <expresion>

| <expresion> "<=" <expresion>

| <expresion> "=" <expresion>

| <expresion> "¬=" <expresion>

| <expresion> "<>" <expresion>

| "not"<expresion>

| <expresion>"and"<expresion>

| <expresion>"or"<expresion>

| <expresion>"xor"<expresion>

| <expresion>"in"<expresion>

| <identificador>

| <constante>

| <conjunto>

| <funcion>

<identificador> ::= <letra> <restoIdentificador>

| "\_"<restoIdentificador>

<restoIdentificador> ::= <restoIdentificador> <digito>

| <restoIdentificador> <letra>

| <digito>

| <letra>

| ""

<conjunto> ::= "["<restoConjunto>"]"

<restoConjunto> ::= <expresion> "," <restoConjunto>

| <constante>".."<constante>

|<expresion>

| ""

<constante> ::= <constanteReal>

| <constateEntera>

| <caracter>

| <booleano>

<constanteReal> ::= <constanteEntera>"."<constanteEntera>

<constanteEntera> ::= <constanteEntera> <digito>

| <digito>

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

<digito> ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"

<letra> ::= "a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|"l"|"m"

|"n"|"o"|"p"|"q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|"y"|"z"

|"A"|"B"|"C"|"D"|"E"|"F"|"G"|"H"|"I"|"J"|"K"|"L"|"M"

|"N"|"O"|"P"|"Q"|"R"|"S"|"T"|"U"|"V"|"W"|"X"|"Y"|"Z"

<booleano> ::= "true"

| "false"

Semántica en lenguaje natural

En este apartado es explicará cómo funciona todas y cada una de las instrucciones del lenguaje que estamos creando. Se irán describiendo de más generales a más concretas.

Programa

Primeramente, todo programa debe definirse. Por tanto debe constar de una cabecera (donde se le da nombre al programa) y un bloque de instrucciones (que hace el programa).

El bloque de instrucciones tiene delimitadores, para saber dónde empieza y dónde acaba, y tres partes principales:

* La declaración de variables.
* La declaración de subprogramas (funciones en nuestro lenguaje).
* Las sentencias del programa.

Declaración de variables

En la declaración de variables tenemos dos partes:

* Delimitadores.
* Variables (nombre y tipo). Para declarar las variables se declaran primero los tipos y luego se añaden los identificadores de las variables para cada tipo.

Declaración de subprogramas (funciones)

En esta parte se declaran los subprogramas o funciones que queremos utilizar en nuestro programa. Tiene dos partes:

* Cabecera. Se describe el nombre de la función y su valor de retorno.
* Bloque. En este bloque es donde colocaremos la declaración de variables (se declaran como se ha visto en el apartado anterior) locales y nuestras instrucciones a ejecutar, ya sean estructuras de control o operaciones (que explicaremos en el apartado siguiente)..

Sentencias del programa

Este apartado es el más extenso de todos por la variedad de opciones que tenemos. Se pueden poner todas las sentencias que queramos, separadas todas ellas por punto y coma (;). Las sentencias son de varios tipos y se clasifican en:

* Sentencias de control del flujo del programa.
* Sentencias de asignación.
* Sentencias de I/O.
* Sentencias de retorno

En las estructuras de control del flujo siempre se evalúa una expresión para decidir qué hacer. Estas expresiones están detalladas en la introducción de esta documentación y dependiendo de los diferentes tipos de variables se pueden hacer unas u otras.

Los tipos se han definido como:

* Carácter. Letras (mayúsculas y minúsculas) e números.
* Booleano. True o false.
* Entero. Número entero.
* Real- Número real, con parte entera y parte real separadas por un punto.
* Conjunto sobre cualquiera de los tipos anteriores. Variables de tipos anteriores separadas por comas y englobadas por corchetes.

La asignación se realiza sobre identificadores (variables).Los identificadores son formaciones de caracteres (letras o \_) y se asigna una expresión.

Las sentencias de I/O son para leer y escribir por los periféricos predeterminados. Leer de teclado y escribir en la pantalla. Aquí aparece una nueva producción la “frase”. La frase es una cadena de caracteres (un string).

Por último, la sentencia de retorno se una para indicar que expresión va a devolver nuestra función.

Identificación de los tokens

Según nuestra gramática hemos podido identificar los siguientes tokens:

* TIPO. Tipos de variables/conjuntos
* OPBI. Operadores binarios
* OPUN. Operadores unarios
* CONS. Constantes carácter, booleano y real
* CONSEN. Constante entera
* IDEN. Identificador
* NOT. Operador negación
* CO. La coma
* PAA. Paréntesis abierto
* PAC. Paréntesis cerrado
* SALIDA. Instrucción de salida
* FRASE. Cadena de caracteres
* COSIM. Comilla simple
* CODOB. Comilla doble
* ENTRADA. Instrucción de entrada
* FOR. Estructura de control para
* IGUAL. Operador de asignación
* TO. Inicio del bloque para
* DO. Final del bloque para
* WHILE. Estructura de control mientras
* IF. Estructura de control si
* THEN. Inicio del bloque si
* ELSE. Inicio del bloque sino
* ELIF. Final del bloque si
* PUNCO. Punto y coma
* FUNCION. Denotación de declaración de función
* BEGIN. Inicio de bloque
* END. Final de bloque
* VAR. Variables
* PROGRAM. Denotación de declaración de programa
* PUN. Punto
* PUN2. Dos puntos
* CORA. Corchete abierto
* CORC. Corchete cerrado