LENGUAJE
DEUS EX-MACHINA

SERGIO ALEXANDER FLOREZ GALEANO
CAMILO FERNANDEZ BERNAL

PRESENTADO A:
MAURICIO ALEJANDRO ESPINOSA ARIAS

UNIVERSIDAD TECNOLOGICA DE PEREIRA
INGENIERIA DE SISTEMAS Y COMPUTACION
COMPILADORES
PEREIRA, JUNIO DE 2012

Tabla de contenido

INTRODUCCION

El presente documento describe formalmente los componentes para la construcción de un compilador, sobre un lenguaje descrito ficticio descrito por nosotros, llamado “DEUS EX-MACHINA”.
Se describirá formalmente cada una de las etapas de la construcción del compilador como lo son:

- **El análisis Léxico.**

  Consiste en leer texto o código del programa fuente carácter a carácter e irlos agrupando generando los tokens, los cuales tienen un significado para el lenguaje. Estos constituyen la entrada para la el análisis sintáctico.
  Resumiendo, las funciones del análisis léxico son:

  - Agrupar caracteres según categorías establecidas por la especificación del lenguaje fuente
  - Rechazar texto con caracteres ilegales o agrupados según un criterio no especificado.

- **Análisis Sintáctico.**

  Consiste en comprobar que los tokens que le suministra el analizador léxico van ordenados según la especificación de la gramática de nuestro lenguaje a compilar “DEUS EX-MACHINE”. Y si no es así, dar los mensajes de error adecuados, pero continuar funcionando sin detenerse, hasta que llegue al final del fichero de entrada.

  Es esencial que el proceso de análisis no se detenga en el primer error encontrado, ya que así podremos dar información al usuario en un solo informe de todos los errores generados.

En el desarrollo del presente informe las siguientes etapas no se definirán aun.

- Tabla de tipos y símbolos.
- Análisis Semántico.
- Generación de código intermedio y final.

LENGUAJE
"DEUS EX-MACHINA"

DEFINICION

El compilador definido sobre el lenguaje de programación "DEUS EX-MACHINA" el cual traduce "Dios surgido de la maquina" es un lenguaje imperativo estructurado. Es un lenguaje pequeño, sencillo, con fines propiamente académicos y no está especializado en ningún tipo de aplicación.

DESCRIPCION

Las especificaciones del lenguaje se definen a continuación:

- Se usa una **notación prefija** para definir expresiones aritmeticologicas.
- Se permite la importación de librerías o código externo mediante el comando **"importar"** seguido de la ruta del archivo, este debe ir al comienzo del programa o fichero de entrada.
- Una importante función de inicio de programa, llamada función **"principal".** Este se ubicara en cualquier parte después de la definición de prototipos, constantes y variables globales**.**
- Las variables o identificadores de funciones, arreglos etc, las representa el token **NICK**.
- Se permite la definición de constantes, las cuales se definirán con la palabra **"constante"** seguido del tipo y su valor, estas deben definirse justo después de la importación de librerías y antes de la definición del bloque principal.
- La esencia de todo lenguaje de programación son las funciones, en nuestro lenguaje "DEUS EX-MACHINE", estas deben definirse mediante un **prototipo**, el cual contiene:
    - Tipo de dato a retornar.
    - El Token "macro".
    - NICK que identifica a la funcion.
    - Cero, uno o más definición de parametros.

  La definición de prototipos se ubicara justo después de la definición de librerias y antes del bloque principal. Toda función debe estar definida por su prototipo, esto facilita la creación de la tabla de símbolos. Lo adicional a la funcion como:
    - Conjunto de cero, una o más sentencias a ejecutar
    - Valor de retorno.

  Se podrá definir en cualquier lugar después del prototipado del fichero del programa. Se permitirá la sobre carga de funciones.

  Para el **llamado a funciones** se requerirá del NICK de la funcion seguido de su conjunto de parametros. Para los parametros se permitirá todo tipo de valores y

expresiones siempre y cuando el valor referente a estas corresponda con el tipo de dato del prototipo de la funcion.

- No se permitirán valores **nulos**.
- Se definen 5 tipos de datos básicos: entero, real, carácter, cadena y booleano.
- Se permite la definición de estructuras, que funcionan como nuevos tipos de dato, mediante la palabra **"estructura".**
- Como es de esperar el conjunto de los enteros y reales permite números **negativos y positivos**.
- Se permite la definición de arreglos unidimensionales usando el NICK para identificar al arreglo y los tokens IZQCORCHET("[") y DERCORCHET("]").

Estructura de los componentes del lenguaje:

A continuación se muestra de manera muy general los componentes del lenguaje.

Estructura de un programa

#NICK#

<<<Importación de librerías>>>

<<<Prototipos de funciones>>>

<<<Constantes>>>

<<<Variables Globales y Estructuras>>>

<<<Función Main>>>

<<<Definición de funciones>>>

Definiciones

- Definicion de Variables

  <tipo de dato> = <identificador> <dato> #

- Prototipo de funciones

  <tipo de dato> macro <identificador>(<parámetro 1>, <parámetro 2>,...)#

- Macro (funcion)

  <tipo de dato> macro <identificador>(<parámetro 1>, <parámetro 2>,...):/

  <<<Bloque de código>>>

  \:

- Arreglos

  <tipo de dato>[ ] = <identificador> {<dato 1>, <dato 2>,...}#

Estructuras de control:

- Si- sino:

  si (<condición>):/

  <<<Bloque de código>>>

  \ : sino :/

  <<<Bloque de código>>>

  \:

- Selector

  selector (<identificador>) :/

  caso <valor1> : <<<Bloque de código>>>

  caso <valor2> : <<<Bloque de código>>>

  caso <valor3> : <<<Bloque de código>>>

  defecto : <<<Bloque de código>>>

  \:

- Hacer- mientras

  hacer:/

  <<<Bloque de código>>>

  \: mientras (<condición>)#

- Mientras

  mientras (<condición>):/

  <<<Bloque de código>>>

  \:

- Para

  para (<declaración>:<condición>:<expresión>):/

  <<<Bloque de código>>>

  :\

- Operador ternario

  (<condición>)? <dato si>: <dato no> #

ANALISIS LEXICO

A continuación se describen todos los componentes necesarios para el proceso de análisis Lexico del compilador que interpretara el lenguaje de programación “KABOOM!!! BABY”, el cual se desarrollara en Python haciendo uso de las herramientas suministradas por lex, de la librería PLY, para análisis léxicos.

El conjunto de símbolos que reconocerá el lenguaje “KABOOM BABY” son los caracteres alfanuméricos reconocidos por el alfabeto español, incluyendo símbolos especiales.

Descripción del Alfabeto

Nombre del Conjunto	Símbolos que pertenecen al conjunto
Alfabéticos	{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,u,v,w,x,y,z,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,U,V,W,X,Y,Z}
Numéricos	$\{\forall x \mid x \in \mathbb{N} \wedge 0 \leq x \leq 9\}$
Aritméticos	{+,-,*,/}
Relacionales	{<,>,~}
Lógicos	{|,&}
Asignación	{=}
Delimitadores	{.,{,},(,),[,]}

Definición del Alfabeto

$\sum$ = {Alfabéticos U Numéricos U Aritméticos U Relacionales U lógicos U Delimitadores}

Otros Símbolos

En el proceso de reconocimiento de caracteres cualquier otro símbolo que no pertenezca al alfabeto, será considerado como carácter ilegal, y en el caso de los espacios, tabuladores y saltos de línea, serán ignorados por el lenguaje.

COMPONENTES LEXICOS

Tabla de Tokens y Lexemas

TOKEN	LEXEMA	EXPRESION REGULAR	TIPO
PRINCIPAL	principal	r'principal'	Palabras Reservadas Estructuras De Control
IMPORTAR	importar	r'importar'	
CONSTANTE	Constante	r'constante'	
HACER	hacer	r'hacer'	
MIENTRAS	mientras	r'mientras'	
PARA	para	r'Para'	
SI	si	r'si'	
SINO	sino	r'sino'	
SELECTOR	selector	r'selector'	
CASO	caso	r'caso'	
DEFECTO	defecto	r'defecto'	
MACRO	macro	r'macro'	
ROMPER	romper	r'romper'	Sentencias de Salto
CONTINUE	continue	r'continue'	
RETORNO	retorno	r'retorno'	
CARACTER	caracter	r'caracter'	Tipos de Dato
CADENA	cadena	r'cadena'	
ENTERO	entero	r'entero'	
REAL	real	r'real'	
VACIO	vacio	r'vacio'	
BOOLEANO	booleano	r'booleano'	
ESTRUCTURA	estructura	r'estructura'	
IMPRIMIR	imprimir	r'imprimir'	Escritura / Lectura
LEER	leer	r'leer'	
NICK		r'[a-zA-Z][a-zA-Z0-9]*'	Identificadores Y Constantes
ENTVALOR		r'\d+([uU]|[lL]|[uU][lL]|[lL][uU])?'	
REALVALOR		r'((\d+)(\.\d+)(e(\+|-)?(\d+))? | (\d+)e(\+|-)?(\d+))([lL]|[fF])?'	
CARVALOR		r'(L)?\'([^\\\n]|(\\.))*?\''	
CADVALOR		r'\"([^\\\n]|(\\.))*?\"'	
BOOLVALOR		r'"verdad"|"mentira"'	
O	|	r'|'	

Y	&	r'&'	Operadores Lógicos
NEGAR	~	r'~'	
SUMA	+	r'\+'	Operadores Aritméticos
MENOS	-	r'\-'	
MULT	*	r'*'	
DIV	/	r'/'	
MODULO	%	r'%'	
EXP	^	r'^'	
MENOR	<	r'<'	Operadores Relacionales
MAYOR	>	r'>'	
MENORIGUAL	<=	r'<='	
MAYORIGUAL	>=	r'>='	
IGUAL	==	r'=='	
DIFERENTE	~=	r'~='	
ASIGNAR	=	r'='	Operadores de Asignación
SUMAIGUAL	+=	r'\+='	
MENOSIGUAL	-=	r'-='	
MULTIGUAL	*=	r'*='	
DIVIGUAL	/=	r'/='	
MASMAS	++	r'\+\+'	Incremento Decremento
MENOSMENOS	--	r'--'	
TERNARIO	?	r'\?'	Operador Ternario
IZQPAREN	(	r'\('	Delimitadores
DERPAREN	)	r'\)'	
IZQCORCHET	[	r'\['	
DERCORCHET	]	r'\]'	
IZQLLAVE	:/	r':/'	
DERLLAVE	\:	r'\\:'	
COMA	,	r','	
PUNTO	.	r'\.'	
PUNTOCOMA	;’	r';'	
DOSPUNTOS	:	r':'	
CIERRELINEA	#	r’#’	

Descripción

Palabras reservadas: A continuación se listaran las palabras reservadas que tienen algún significado gramatical para el lenguaje de programación “KABOOM!!! BABY”

- Tokens importantes
  - ‘PRINCIPAL’: Es el nombre de la funcion que se ejecutara primero en el programa.
  - ‘CONSTANTE’: Permite definir un dato inmutable durante todo el programa.
  - ‘IMPORTAR’: Permite traer código o librerias de otros ficheros de entrada

- Estructuras de control:
  - 'HACER': Ejecuta un bloque de código mientras se cumpla determinada condición acompañada de algún operador de relación como: “!”, “<”, ”<=”, “>”, “>=”, “==” o “!=”.
  - 'MIENTRAS': Mientras una condición se mantenga verdadera se repite la ejecución de un bloque de código, dicha condición podrá estar acompañada de algún operador de relación como: “!”, “<”, ”<=”, “>”, “>=”, “==” o “!=”.
  - 'PARA': Ejecuta un bloque de código un número determinado de veces, según indique el iterador.
  - 'SI': Ejecuta un grupo de sentencias, solo si se cumple determinada condición.
  - 'SINO': Cuando la condición principal no es válida pasara a ejecutarse el código contenido en esta estructura.
  - 'SELECTOR': Estructura que permite la ejecución de un bloque de código dependiendo del valor de un indicador.
  - 'CASO’: Es un bloque de código que se ejecutara si el valor del indicador corresponde al asignado.
  - 'DEFECTO': Valores por defecto de un indicador.

- Sentencias de salto
  - 'ROMPER’: Permite detener la ejecución del código contenido en una estructura de control y salirse de esta.
  - 'CONTINUE': Sirve para detener la iteración actual y volver al principio del bucle para realizar otra iteración, si corresponde.
  - 'RETORNO': Devuelve o retorna el valor actual de una función.

- Tipos de dato
  - 'CARACTER': Es un digito individual el cual se puede representar como una letra (a-z).
  - 'CADENA': Almacena un conjunto de caracteres.
  - 'ENTERO': Almacena números que solo poseen parte entera.
  - 'REAL': Almacenan números muy grandes que poseen parte entera y pueden o no tener parte decimal.
  - 'VACIO': No almacena ningún dato, usado cuando una función requiere ejecutar algún bloque de código, pero necesita retornar ningún valor.

- Entrada y salida de datos.
  - 'LEER': Lectura de datos
  - 'IMPRIMIR': Impresión por pantalla de datos.

- Identificadores y constantes.
  - “NICK”: Identificador que permite definir variables y constantes.
  - “ENTVALOR”: Tipo de dato numérico que representa valores enteros.
  - “REALVALOR”: Tipo de dato numérico que representa valores reales.
  - “CARVALOR”: Tipo de dato que representa cadenas de caracteres.
  - “CADVALOR”: Tipo de dato que representa caracteres.

- Operadores aritméticos.
  - “SUMA”: Suma de datos numéricos.
  - “MENOS”: Resta de datos numéricos.
  - “MULT”: Multiplicación de datos numéricos.
  - “DIV”: División de datos numéricos.
  - “MODULO”: Permite obtener el residuo de la división de dos números.
  - “EXP”: Permite elevar un numero a una potencia determinada.

- Operadores lógicos.
  - “Y”: Conjunción.
  - “O”: Disyunción.
  - “NEGAR”: Negación de un dato.

- Operadores relacionales.
  - “MENOR”: Comparación menor que.
  - “MAYOR”: Comparación mayor que.
  - “MENORIGUAL”: Comparación menor o igual que.
  - “MAYORIGUAL”: Comparación mayor o igual que.
  - “IGUAL”: Comparación igual.
  - “DIFERENTE”: Comparación diferente.

- Operadores de asignación.
  - “ASIGNAR”: Dar un valor a un dato numérico.
  - “MENOSIGUAL”: Asignar la resta de un dato numérico por sí mismo y por otro dato numérico.
  - “MASIGUAL”: Asignar la suma de un dato numérico por sí mismo y por otro dato numérico.
  - “MULTIGUAL”: Asignar la multiplicación de un dato numérico por sí mismo y por otro dato numérico.
  - “DIVIGUAL”: Asignar la división de un dato numérico por sí mismo y por otro dato numérico.

- Incremento / Decremento.
  - “MENOSMENOS”: Disminuir en una unidad de un dato numérico.
  - “MASMAS”: Aumentar en una unidad de un dato numérico.

- Operador ternario.
  - “TERNARIO”: Permite escribir expresiones condicionales.

- Delimitadores.
  - “IZPARENT”: Permite abrir un bloque de expresiones.
  - “DERPAREN”: Permite cerrar un bloque de expresiones.
  - “IZQCORCHET”: Permite abrir un bloque de definición de los valores de un arreglo.
  - “DERCORCHET”: Permite cerrar un bloque de definición de los valores de un arreglo.
  - “IZQLLAVE”: Permite abrir un bloque de código.
  - “DERLLAVE”: Permite cerrar un bloque de código.
  - “COMA”: Permite separar valores.
  - “PUNTO”: Permite separar la parte real de la imaginaria.
  - “PUNTOCOMA”:
  - “DOSPUNTOS”: Separa las condiciones en el operador ternario y en la estructura de control para.
  - “FINLINEA”: Determina donde termina una línea de código.

- Cierre de línea:
  - CIERRELINEA En el lenguaje “Z” se establece el cierre de línea con el carácter ‘#’
- Comentarios: En el lenguaje “Z” los bloques de comentario son todo aquello comprendido entre los caracteres “<<<” y “>>>”.

ANALISIS SINTACTICO

Durante esta etapa se comprobara que los tokens que le suministra el analizador léxico van ordenados según la especificación de la gramática del lenguaje a compilar. Y si no es así los menajes de error adecuados. Se definirán las gramáticas BNF necesarias para la descripción total del lenguaje.

Ejemplos

Primero se presentan una serie de ejemplos que muestran como esta construida cada expresión del lenguaje, y más adelante se muestra la gramática BNF del mismo

Programa

```
#Mi_Primer_Programa#:
Importar "C:/milibreria.py"#
booleano macro exprimo(entero numero)#
constante real = pi  3.1416#
vacío principal()/:
        imprimir(exprimo(5))#
\:
booleano exprimo(entero numero)/:
        entero mitad = / numero 2#
        para(entero = i 2 : <= i mitad : ++ i)/:
                si(== (% numero i) 0)
                        retorno mentira#
        \:
        retorno verdad#
\:
```

Importar Librerías

```
Importar "C:/python/léxico.py"#
```

Dato

```
5
 10.3
'a'
"soy una cadena"
verdad
funcion(x,4)
numerosprimos[5]
/ (+ 33 multiplicar(5,6)) 30
```

Asignación

```
=x 5#
+= x 3#   -= x 3#    *= x 3#   /= x 3#    %= x 3#
= x <dato>#
=x [<dato>,<dato>,<dato>...]#
```

Declaración

```
entero x#
```

real = x 6,5#
<tipodato> = x <dato>#
caracter[] x#
caracter[] = x caracter[5]#
cadena[] =x [“cad1”, “cad2”,”cad3”…]#
<tipodato>[] =x [<dato>,<dato>,<dato>…]#
<tipodato>[][] =x [[<dato>,<dato>,<dato>…],[<dato>,<dato>,<dato>…]]#
constante = pi 3.1416#

ESTRUCTURAS DE CONTROL

Condicional si

```
si (== x 10)
      imprimir “hola”#
si (== mod(x,2) 0):/
      <sentencia1>#
      <sentencia2>#
      …
\:
sino:/
     Imprimir(“cad3”)#
\:
```

Estructura Selector

```
Selector (var):/
       caso 1 :
          <sentencia1>#
          <sentencia2>#
          …
          ROMPER#
        caso 2:
          ….
          ROMPER#
        caso n:
          …
          ROMPER#
       defecto  :
          imprimir(“soy un caso especial”)#
          ROMPER#
\:
```

Operador Ternario

(x<y)? x : y#

Ciclo hacer

hacer :/

<sentencias>#
\: mientras(verdad)

Ciclo mientras

mientras(verdad):/
<sentencias>#
\:

Ciclo para

para(entero = var 1 : < var 10 : ++ var):/
<sentencias>#
\:

GRAMATICAS BNF

Programa

<programa> ::= CIERRELINEA NICK CIERRELINEA <importarlibrerias> <prototipos> <constantes> <declaraciones> <principal> <funciones>
| CIERRELINEA NICK CIERRELINEA importarlibrerias principal
| CIERRELINEA NICK CIERRELINEA principal
| CIERRELINEA NICK CIERRELINEA constantes principal
| CIERRELINEA NICK CIERRELINEA declaraciones principal
| CIERRELINEA NICK CIERRELINEA constantes declaraciones principal
| CIERRELINEA NICK CIERRELINEA importarlibrerias constantes principal
| CIERRELINEA NICK CIERRELINEA importarlibrerias constantes declaraciones principal
| CIERRELINEA NICK CIERRELINEA prototipos principal funciones
| CIERRELINEA NICK CIERRELINEA importarlibrerias prototipos principal funciones
| CIERRELINEA NICK CIERRELINEA importarlibrerias prototipos constantes principal funciones
| CIERRELINEA NICK CIERRELINEA prototipos constantes principal funciones
| CIERRELINEA NICK CIERRELINEA prototipos constantes declaraciones principal funciones

Importar librerias

<importarlibrerias>::= IMPORTAR CADVALOR CIERRELINEA
| <importarlibrerias> IMPORTAR CADVALOR CIERRELINEA

Prototipos de funciones

<prototipos> ::= <prototipofuncion>
| <prototipos> <prototipofuncion>

<prototipofuncion>::= <tipodato> MACRO NICK IZQPAREN DERPAREN
| <tipodato> MACRO NICK IZQPAREN <declaracionparametros> DERPAREN

Parametros

<declaracionparametros> ::= <nuevoparametro>
| <declaracionparametros> COMA <nuevoparametro>

<nuevoparametro> ::= <tipodato> NICK
| <tipodato> IZQCORCHET DERCORCHET NICK

Tipos de dato

<tipodato> ::= ENTERO
| REAL
| CARACTER
| CADENA
| BOOLEANO

Constantes

<constantes>::= CONSTANTE <tipodato> ASIGNAR NICK <dato> CIERRELINEA
| constantes CONSTANTE <tipodato> ASIGNAR NICK <dato> CIERRELINEA

Datos

<dato> ::= ENTVALOR
| REALVALOR
| CARVALOR
| CADVALOR
| BOOLVALOR

<datofuncion> : := NICK IZQPAREN <parámetros> DERPAREN'

<datoarreglo> ::= NICK IZQCORCHET ENTVALOR DERCORCHET

Parametros

<parametros> ::= <dato>
| <parámetros> COMA <dato>

Operador ternario

<operadorternario> ::= IZQPAREN <expresión> DERPAREN TERNARIO <expresión> DOSPUNTOS <expresión>

Expresiones

<expresion> ::= <expresionsimple>
| expresioncompuesta

<expresionsimple> ::= NICK
| <datofuncion>
| <dato>
| <operadorternario
| <datoarreglo>

<expresioncompuesta> ::= <operador> <expresion>
| IZQPAREN <expresión> DERPAREN
| <operador> <dato> IZQPAREN <expresión> DERPAREN
| <operador> IZQPAREN <expresión> DERPAREN <dato>
| <operador> IZQPAREN <expresión> DERPAREN IZQPAREN <expresion> DERPAREN
| <operador> <dato> <dato>

Operadores

<operador> ::= SUMA
| MENOS
| MASMAS
| MENOSMENOS
| MULT
| DIV
| MODULO

| EXP
| Y
| O
| NEGAR
| MENOR
| MAYOR
| MENORIGUAL
| MAYORIGUAL
| IGUAL
| DIFERENTE

Declaraciones

<declaraciones> ::= <declaración>
| <declaracionestructura>
| <declaraciones> <declaración>
| <declaraciones> <declaracionestructura>

<declaración> ::= <tipodato> <restodeclaracion> CIERRELINEA

<restodeclaracion> ::= NICK
| <asignación>
| IZQCORCHET DERCORCHET NICK
| IZQCORCHET DERCORCHET ASIGNAR NICK tipodato IZQCORCHET ENTVALOR DERCORCHET
| IZQCORCHET DERCORCHET ASIGNAR NICK IZQCORCHET parametros DERCORCHET

<declaracionestructura> ::= ESTRUCTURA NICK IZQPAREN <declaraciones> DERPAREN

Funcion Principal

<principal> ::= VACIO MACRO PRINCIPAL IZQPAREN DERPAREN IZQLLAVE <bloque> DERLLAVE

Bloque

<bloque> ::= sentencia
| bloque sentencia

Sentencias

<sentencia> ::= <declaración>
| <asignación> CIERRELINEA
| <sentenciasalto>
| <sentencialeer>
| <sentenciaescribir>
| <condicionalsi>
| <ciclohacer>
| <ciclomientras>
| <ciclopara>
| <selector>

Estructuras De Control

<ciclohacer> ::= HACER IZQLLAVE <bloque> DERLLAVE MIENTRAS IZQPAREN <expresión>
DERPAREN

<ciclomientras> ::= MIENTRAS IZQPAREN <expresion> DERPAREN IZQLLAVE <bloque> DERLLAVE'

<ciclopara> ::= PARA IZQPAREN <declaración> DOSPUNTOS <expresion> DOSPUNTOS
<asignación> DERPAREN IZQLLAVE <bloque> DERLLAVE

<selector> ::= SELECTOR IZQPAREN <variablecontrol> DERPAREN IZQLLAVE <casos> DERLLAVE'

<variablecontrol> ::= NICK

<casos> ::= <casoselector>
| <casoselector> <casoespecial>

<casoselector> ::= CASO <valorcontrol> DOSPUNTOS <bloque>
| <casoselector> CASO <valorcontrol> DOSPUNTOS <bloque>

<valorcontrol> ::= ENTVALOR
| CARVALOR

<casoespecial> ::= DEFECTO DOSPUNTOS <bloque>

<condicionalsi> ::= SI IZQPAREN <expresion> DERPAREN IZQLLAVE <bloque> DERLLAVE SINO
IZQLLAVE <bloque> DERLLAVE
| SI IZQPAREN <expresion> DERPAREN IZQLLAVE <bloque> DERLLAVE

<sentenciasalto> ::= <tokensalto> CIERRELINEA

<tokensalto> ::= ROMPER
| CONTINUE
| RETORNO

<sentencialeer> ::= LEER IZQPAREN DERPAREN'

<sentenciaescribir> ::= IMPRIMIR IZQPAREN <info> DERPAREN CIERRELINEA'

<info> ::= <dato>
| <dato> COMA <dato>

Asignación

<asignacion> ::= ASIGNAR NICK <expresion>
| <operadorasignacion> NICK <expresion>

<operadorasignacion> ::= IGUAL
| SUMAIGUAL
| MENOSIGUAL
| MULTIGUAL

| DIVIGUAL'''

Funciones

<funciones> ::= <declaracionfuncion>
| <funciones> <declaracionfuncion>

<declaracionfuncion> ::= VACIO MACRO NICK IZQPAREN DERPAREN IZQLLAVE <bloque> DERLLAVE
| <tipodato> MACRO NICK IZQPAREN DERPAREN IZQLLAVE <bloque> DERLLAVE
| <tipodato> MACRO NICK IZQPAREN <declaracionparametros> DERPAREN IZQLLAVE <bloque> DERLLAVE'''

Error

```
def p_error(p):
    print "Usted tiene un error de sintaxis en alguna parte de su codigo."
    print "Este puede estar cerca de la linea %d. " % p.lineno
    print "Hay un error cerca de " + str(p.value)
    print "Buena suerte encontrandolo."
```

TABLA DE ERRORES SINTACTICOS

CODIGO	MENSAJE DE ERROR	DESCRIPCION
1	Se esperaba la aparición del identificador 'PRINCIPAL'	Se omitió el identificador 'PRINCIPAL' en el bloque principal
2	Se esperaba la aparición del identificador 'IMPORTAR'	Se omitió el identificador 'IMPORTAR' para cargar una librería
3	Se esperaba la aparición del identificador 'CONSTANTE'	Se omitió el identificador 'CONSTANTE' para definir valores estáticos
	Se esperaba la aparición del identificador 'ESTRUCTURA'	Se omitió el identificador 'ESTRUCTURA' para definir valores estáticos
4	Se esperaba un identificador	Se omitió un identificador tipo NICK
5	Se esperaba la aparición del identificador de estructura de control	Se omitió el identificador el identificador de la estructura de control Estructura de control: 'HACER' | 'PARA' | 'SI' | 'SINO' | 'SELECTOR' | 'CASO' | 'DEFECTO' | 'PRINCIPAL'
6	Se esperaba la aparición de un identificador de sentencia de salto	Se omitió el identificador de sentencia de salto. Sentencias de salto: ROMPER | CONTINUE | RETORNO
7	Se esperaba un identicador de tipo de dato	Se omitió el identificador de tipo de dato Tipo de dato: CARÁCTER | CADENA | ENTERO |REAL |VACIO | BOOLEANO.
8	Se esperaba un identificador de lectura/escritura	Se omitió el identificador de lectura/escritura Lectura/Escritura: imprimir | leer
9	Se esperaba un operador lógico { '~' , '|' , '&' }	Se omitió un operador lógico 'O' o 'Y' o 'NEGAR'
10	Se esperaba un operador aritmético {'+' , '-' , '*' , '/' , '%' , '^' , '++' , '--'}	Se omitió un operador aritmético: 'SUMAR' , 'RESTAR' , 'MULT' , 'DIV' , 'MODULO' , 'EXP' , 'MASMAS' , 'MENOSMENOS'
11	Se esperaba un operador relacional { '<' , '>' , '<=','>=','==','~='}	Se omitió el operador relacional: 'MENOR' , 'MAYOR' , 'MENORIGUAL','MAYORIGUAL','IGUAL','DIFERENTE'

12	Se esperaba un operador de asignación {'=','+=','-=','*=','/='}	Se omitió un operador de asignación: 'ASIGNAR' , 'SUMAIGUAL' , 'MENOSIGUAL' , 'MULTIGUAL' , 'DIVIGUAL'
13	Se esperaba '?'	Se omitió '?' como operador ternario
14	Se esperaba '('	Se omitió '(' para abrir un grupo de parámetros, expresiones o datos.
15	Se esperaba ')'	Se omitió '(' para cerrar un grupo de parámetros, expresiones o datos.
16	Se esperaba '['	Se omitió '[' en el momento abrir una especificacaciom valores o tamaños de un vector
17	Se esperaba ']'	Se omitió ']' en el momento de cerrar una especificación de valores o tamaños de un vector
18	Se esperaba ':/'	Se omitió ':/' para abrir un bloque de código
19	Se esperaba '\:'	Se omitió '\:' para cerrar un bloque de condigo
20	Se esperaba ','	Se omitió ',' para separar valores
21	Se esperaba '.'	Se omitió '.' para separar la parte real de la imaginaria
22	Se esperaba ';'	Se omitió ';'
23	Se esperaba ':'	Se omitió ':' para la separación de parámetros en
24	Se esperaba '#'	Se omitió '#' para indicar el cierre de línea
25	El archivo especificado no ha sido encontrado	Se hace referencia a un archivo que no existe o la especificación del nombre o dirección no sigue la estructura determinada o cuenta con caracteres no validos.
26	Error de sintaxis o estructura de la sentencia	Existe algún error en la construcción de las sentencias definidas, o se encontró un carácter inválido en el código fuente.
27	Se esperaba un ' " '	Se esperaba un cierre de cadena 'CADVALOR'
28	Estructura no concuerda	Estructura de la función o estructura de control no concuerda con la definición previa
29	Operador lógico adicional no	Operador lógico 'O' o 'Y' o 'NEGAR'

	esperado { '~' , '|' , '&' }	
30	Operador aritmético adicional no esperado {'+' , '-' , '*' , '/' , '%' , '^' , '++' , '--'}	Operador aritmético extra: 'SUMAR' , 'RESTAR' , 'MULT' , 'DIV' , 'MODULO' , 'EXP' , 'MASMAS' , 'MENOSMENOS'
31	Operador relacional adicional no esperado { '<' , '>' , '<=','>=','==','~='}	Operador relacional extra: 'MENOR' , 'MAYOR' , 'MENORIGUAL','MAYORIGUAL','IGUAL','DIFERENTE'
32	Operador relacional adicional no esperado {'=','+=','-=','*=','/='}	Operador de asignación extra: 'ASIGNAR' , 'SUMAIGUAL' , 'MENOSIGUAL' , 'MULTIGUAL' , 'DIVIGUAL'
33	No se esperaba '?'	Carácter extra '?' como operador ternario
34	No se esperaba '('	Carácter extra '(' para abrir un grupo de parámetros, expresiones o datos.
35	No se esperaba ')'	Carácter extra '(' para cerrar un grupo de parámetros, expresiones o datos.
36	No se esperaba '['	Carácter extra'[' en el momento abrir una especificacaciom valores o tamaños de un vector
37	No se esperaba ']'	Carácter extra ']' en el momento de cerrar una especificación de valores o tamaños de un vector
38	No se esperaba ':/'	Carácter extra ':/' para abrir un bloque de código
39	No se esperaba '\:'	Carácter extra '\:' para cerrar un bloque de condigo
40	No se esperaba ','	Carácter extra ',' para separar valores
41	No se esperaba '.'	Carácter extra '.' para separar la parte real de la imaginaria
42	No se esperaba ';'	Se omitió ';'
43	No se esperaba ':'	Carácter extra ':' para la separación de parámetros en
44	No se esperaba '#'	Carácter extra '#' para indicar el cierre de línea

ANALISIS SEMANTICO

Programa

<programa> ::= CIERRELINEA NICK CIERRELINEA <importarlibrerias> <prototipos> <constantes>
<declaraciones> <principal> <funciones>
| CIERRELINEA NICK CIERRELINEA importarlibrerias principal
| CIERRELINEA NICK CIERRELINEA principal
| CIERRELINEA NICK CIERRELINEA constantes principal
| CIERRELINEA NICK CIERRELINEA declaraciones principal
| CIERRELINEA NICK CIERRELINEA constantes declaraciones principal
| CIERRELINEA NICK CIERRELINEA importarlibrerias constantes principal
| CIERRELINEA NICK CIERRELINEA importarlibrerias constantes declaraciones principal
| CIERRELINEA NICK CIERRELINEA prototipos principal funciones
| CIERRELINEA NICK CIERRELINEA importarlibrerias prototipos principal funciones
| CIERRELINEA NICK CIERRELINEA importarlibrerias prototipos constantes principal funciones
| CIERRELINEA NICK CIERRELINEA prototipos constantes principal funciones
| CIERRELINEA NICK CIERRELINEA prototipos constantes declaraciones principal funciones

Importar librerias

<importarlibrerias>::= IMPORTAR CADVALOR CIERRELINEA
| <importarlibrerias> IMPORTAR CADVALOR CIERRELINEA

```
if(len(p)==4):
        librerias.append(p[2])
else:
        librerias.append(p[3])
p[0]=librerías
```

Prototipos de funciones

<prototipos> ::= <prototipofuncion>
| <prototipos> <prototipofuncion>

<prototipofuncion>::= <tipodato> MACRO NICK IZQPAREN DERPAREN
| <tipodato> MACRO NICK IZQPAREN <declaracionparametros> DERPAREN

Parametros

<declaracionparametros> ::= <nuevoparametro>
| <declaracionparametros> COMA <nuevoparametro>

<nuevoparametro> ::= <tipodato> NICK
| <tipodato> IZQCORCHET DERCORCHET NICK

Tipos de dato

<tipodato> ::= ENTERO

```
        | REAL
        | CARACTER
        | CADENA
        | BOOLEANO
```

Constantes

```
<constantes>::= CONSTANTE <tipodato>  ASIGNAR NICK <dato>  CIERRELINEA
        | constantes CONSTANTE <tipodato>  ASIGNAR NICK <dato>  CIERRELINEA
        if(len(p)==7):
                if(not(Tabla.existeTipo(p[4])) and p[2]==p[5][1]):
                        Tabla.addTipo(p[4],p[2],-1,1,-1,-1)
                         Tabla.addSimbolo(p[4],'constante',p[2],-1,[],p[5][0])
                        constantes.append(p[4])
                 else:
                        raise ErrorSemantico("El simbolo '"+p[4]+"' ya existe o los tipos no coinciden")
        else:
                if(not(Tabla.existeTipo(p[5])) and p[3]==p[6][1]):
                        Tabla.addTipo(p[5],p[3],-1,1,-1,-1)
                        Tabla.addSimbolo(p[5],'constante',p[3],-1,[],p[6][0])
                        constantes.append(p[5])
                else:
                        raise ErrorSemantico("El simbolo '"+p[5]+"' ya existe o los tipos no coinciden")
        p[0]=constantes
```

Datos

```
<dato_1> ::= ENTVALOR
        p[0]=[p[1],"entero"]

<dato_2> ::=  REALVALOR
        p[0]=[p[1],"real"]

<dato_3> ::=  CARVALOR
        p[0]=[p[1],"caracter"]

<dato_4> ::=  CADVALOR
        p[0]=[p[1],"cadena"]

<dato_5> ::=  BOOLVALOR
        p[0]=[p[1],"booleano"]


<datofuncion> : := NICK IZQPAREN <parámetros>  DERPAREN'
```

<datoarreglo> ::= NICK IZQCORCHET ENTVALOR DERCORCHET

Parametros

<parametros> ::= <dato>
| <parámetros> COMA <dato>
if(len(p)==2):
tempparametros.append(p[1])
else:
tempparametros.append(p[3])
p[0]=tempparametros

Operador ternario

<operadorternario> ::= IZQPAREN <expresión> DERPAREN TERNARIO <expresión> DOSPUNTOS <expresión>

Expresiones

<expresion> ::= <expresionsimple>
| expresioncompuesta
p[0]=p[1]

<expresionsimple_1> ::= NICK
if(Tabla.existeTipo(p[1])):
temp=Tabla.getTablaTipos()[p[1]]
temp1= Tabla.getTablaSimbolos()[p[1]]
p[0]=[temp1.getDireccion(),temp.getTipoBase()]
else:
raise ErrorSemantico("El simbolo '"+p[1]+"' no existe")

<expresionsimple_2> ::= <datofuncion>

<expresionsimple_3> ::= <dato>
p[0]=p[1]

<expresionsimple_4> ::= <operadorternario

<expresionsimple_5> ::= <datoarreglo>

<expresioncompuesta> ::= <operador> <expresion>
| IZQPAREN <expresión> DERPAREN
| <operador> <dato> IZQPAREN <expresión> DERPAREN
| <operador> IZQPAREN <expresión> DERPAREN <dato>
| <operador> IZQPAREN <expresión> DERPAREN IZQPAREN <expresion> DERPAREN
| <operador> <dato> <dato>

Operadores

<operador> ::= SUMA
| MENOS

```
        | MASMAS
        | MENOSMENOS
        | MULT
        | DIV
        | MODULO
        | EXP
        | Y
        | O
        | NEGAR
        | MENOR
        | MAYOR
        | MENORIGUAL
        | MAYORIGUAL
        | IGUAL
        | DIFERENTE
        p[0]=p[1]
```

Declaraciones

```
<declaraciones> ::= <declaración>
        | <declaracionestructura>
        | <declaraciones> <declaración>
        | <declaraciones> <declaracionestructura>

<declaración> ::= <tipodato> <restodeclaracion>  CIERRELINEA
        if(len(p[2])==3):
                pass
        elif(len(p[2])==2):
                if(not(p[2][1]==p[1])):
                        mensaje =  "Los tipos de dato no coinciden"
                        raise ErrorSemantico(mensaje)
        else:
                 Tabla.setTipoTablas(p[2],p[1])

<restodeclaracion_1> ::= NICK
        | IZQCORCHET DERCORCHET  NICK
        | IZQCORCHET DERCORCHET ASIGNAR NICK tipodato IZQCORCHET ENTVALOR
        DERCORCHET
        if(len(p)==4):
                if(not(Tabla.existeTipo(p[3]))):
                        Tabla.addTipo(p[3],-1,-1,1,-1,-1)
                        Tabla.addSimbolo(p[3],'arreglo',-1,-1,[],-1)
                        p[0]=p[3]
                else:
                        raise ErrorSemantico("El simbolo '"+p[3]+"' ya existe")
        elif(len(p)==9):
                if(not(Tabla.existeTipo(p[4]))):
                        aux =int(p[7])-1
                        Tabla.addTipo(p[4],p[5],-1,p[7],0,aux)
```

```
                    Tabla.addSimbolo(p[4],'arreglo',p[5],-1,[],[0]*int(p[7]))
                    p[0]=[p[4],p[5]]
                else:
                    raise ErrorSemantico("El simbolo '"+p[4]+"' ya existe")
            else:
                if(not(Tabla.existeTipo(p[1]))):
                    Tabla.addTipo(p[1],-1,-1,1,-1,-1)
                    Tabla.addSimbolo(p[1],'variable',-1,-1,[],-1)
                    p[0]=p[1]
                else:
                    raise ErrorSemantico("El simbolo '"+p[1]+"' ya existe")
```

<restodeclaracion_2> ::= <asignación>

```
        if(p[1][0]=='='):
            if(not(Tabla.existeTipo(p[1][1]))):
                Tabla.addTipo(p[1][1],p[1][3],-1,1,-1,-1)
                Tabla.addSimbolo(p[1][1],'variable',p[1][3],-1,[],p[1][2])
                p[0]=[p[1][1],p[1][3]]
            else:
                raise ErrorSemantico("El simbolo '"+p[1][1]+"' ya existe")
        else:
            raise ErrorSemantico("El operador '"+p[1][0]+"' no se puede usar en una
            declaracion debe usarse '='")
```

<restodeclaracion_3> ::= IZQCORCHET DERCORCHET ASIGNAR NICK IZQCORCHET parámetros DERCORCHET

```
        parametros=p[6]
        tempparametros=[]
        tipoaux= parametros[0][1]
        for i in parametros:
            if(tipoaux!=i[1]):
                raise ErrorSemantico("Los tipos del array '"+str(p[6])+"' no coinciden")
            tipoaux=i[1]
        if(not(Tabla.existeTipo(p[4]))):
            Tabla.addTipo(p[4],tipoaux,-1,len(p[6]),0,len(p[6])-1)
            Tabla.addSimbolo(p[4],'arreglo',tipoaux,-1,[],p[6])
             p[0]=[p[4],tipoaux]
        else:
            raise ErrorSemantico("El simbolo '"+p[4]+"' ya existe")
```

<declaracionestructura> ::= ESTRUCTURA NICK IZQPAREN <declaraciones> DERPAREN

Funcion Principal

<principal> ::= VACIO MACRO PRINCIPAL IZQPAREN DERPAREN IZQLLAVE <bloque> DERLLAVE

Bloque

<bloque> ::= sentencia

| bloque sentencia

Sentencias

<sentencia> ::= <declaración>
| <asignación> CIERRELINEA
| <sentenciasalto>
| <sentencialeer>
| <sentenciaescribir>
| <condicionalsi>
| <ciclohacer>
| <ciclomientras>
| <ciclopara>
| <selector>

Estructuras De Control

<ciclohacer> ::= HACER IZQLLAVE <bloque> DERLLAVE MIENTRAS IZQPAREN <expresión> DERPAREN

<ciclomientras> ::= MIENTRAS IZQPAREN <expresion> DERPAREN IZQLLAVE <bloque> DERLLAVE'

<ciclopara> ::= PARA IZQPAREN <declaración> DOSPUNTOS <expresion> DOSPUNTOS <asignación> DERPAREN IZQLLAVE <bloque> DERLLAVE

<selector> ::= SELECTOR IZQPAREN <variablecontrol> DERPAREN IZQLLAVE <casos> DERLLAVE'

<variablecontrol> ::= NICK

<casos> ::= <casoselector>
| <casoselector> <casoespecial>

<casoselector> ::= CASO <valorcontrol> DOSPUNTOS <bloque>
| <casoselector> CASO <valorcontrol> DOSPUNTOS <bloque>

<valorcontrol> ::= ENTVALOR
| CARVALOR

<casoespecial> ::= DEFECTO DOSPUNTOS <bloque>

<condicionalsi> ::= SI IZQPAREN <expresion> DERPAREN IZQLLAVE <bloque> DERLLAVE SINO IZQLLAVE <bloque> DERLLAVE
| SI IZQPAREN <expresion> DERPAREN IZQLLAVE <bloque> DERLLAVE

<sentenciasalto> ::= <tokensalto> CIERRELINEA
p[0]=p[1]

<tokensalto> ::= ROMPER
| CONTINUE
| RETORNO
p[0]=p[1]

<sentencialeer> ::= LEER IZQPAREN DERPAREN'

<sentenciaescribir> ::= IMPRIMIR IZQPAREN <info> DERPAREN CIERRELINEA'
        print p[4]

<info> ::= <dato>
        | <dato> COMA <dato>

Asignación
<asignación_1> ::= ASIGNAR NICK <expresion>
        p[0]=[p[1],p[2],p[3][0],p[3][1]]

<asignación_2> ::= <operadorasignacion> NICK <expresion>
        p[0]=[p[1],p[2],p[3][0],p[3][1]]

<operadorasignacion> ::= IGUAL
        | SUMAIGUAL
        | MENOSIGUAL
        | MULTIGUAL
        | DIVIGUAL'''
        p[0]=p[1]

Funciones
<funciones> ::= <declaracionfuncion>
         | <funciones> <declaracionfuncion>

<declaracionfuncion> ::= VACIO MACRO NICK IZQPAREN DERPAREN IZQLLAVE <bloque> DERLLAVE
        | <tipodato> MACRO NICK IZQPAREN DERPAREN IZQLLAVE <bloque> DERLLAVE
        | <tipodato> MACRO NICK IZQPAREN <declaracionparametros> DERPAREN IZQLLAVE
        <bloque> DERLLAVE'''

Error
```
def p_error(p):
        print "Usted tiene un error de sintaxis en alguna parte de su codigo."
        print "Este puede estar cerca de la linea %d. " % p.lineno
        print "Hay un error cerca de " + str(p.value)
        print "Buena suerte encontrandolo."
```

Clase para Errores Semanticos
```
class ErrorSemantico(Exception):
    def __init__(self, valor):
        self.valor = valor
    def __str__(self):
        return repr((self.valor))
```

Clase para Errores Sintacticos

```
class ErrorSintactico(Exception):
    def __init__(self, valor):
        self.valor = valor
    def __str__(self):
        return repr((self.valor))
```