GRAMATICA

P= {

<S> -> # publicus genus id {Principalem () {<instrucciones>} <funciones>}
<clases>
<clases> -><clase><clases>/ &
<clase> -> publicus genus id{<atributos><funciones>} / privatus genus id{<atributos><funciones>}
<atributos> -> <declarar>;<atributos>/&
<funciones> -> <función><funciones>/ &
<función> -> munus id <A>
<A> -> <tipo_dato> (<declarar>) {<instrucciones> reditus id;} /vacuum (<declarar>) {<instrucciones>}
<instrucciones> -> <instruccion><instrucciones>/ &
<instrucción> -> <si_sino >/ <mientras_hacer> /<según_hacer> / <declarar>;/<funcion>/id <asignar>;
<declarar> -> *<M> id <asignar><mas>
<M> -> <tipo_dato> / id
<mas> ->, id <asignar><mas>/ &
<asignar> -> = <valor> / &
<valor> -> <E> / <funcion>
< mientras_hacer > -> dum (id operador_logico id <condicion>) facite {<instrucciones>}
< si_sino > -> si (id operador_logico id <condicion>) {<instrucciones>} <C>
<C> -> autem {<instrucciones>} / &
<segun_hacer>-> per (id) facio {< lista_CASE> default: {<instrucciones>} break;}
<lista_CASE> -> <CASE><LISTA_CASE>/ &
<CASE> -> apud <D>: <instrucciones> break;
<D> -> Letra/Digito
<condicion> -> <condicion_AND>/<condicion_OR > / &
<condicion_AND> -> quod id operador_logico id <condicion>
<condicion_OR >-> uel id operador_logico id <condicion >
<E> -> <Z><V> / (<E><H><V>
<Z> -> id / N.Entero / N.Real
<V> -> <operador_aritmetico> <E> / &
<operador_aritmetico> -> + / - / ^ / * / /
<operador_logico> <=/ >= /! =/ ==
<H> ->)
<tipo_dato> -> TOT/BOOL/VOLI /REALI/CAR/BIN/OCT/HEX/TORQ

}

SIMBOLOS	PRIMERO	SIGUIENTE
S	#	$
clases	publicus,privatus,&	$
clase	publicus,privatus	$
funciones	munus,&	} , break , reditus
función	munus	} , break , reditus , ;
A	TOT,BOOL,VOLI, REALI, CAR, BIN , OCT, HEX ,TORQ ,vacuum	} , break , reditus , ;
instrucciones	si,dum,per,munus,*,&,id	} , break , reditus
instrucción	si,dum,per,munus,*,id	} , break , reditus
declarar	*	; ,)
mas	, , &	; ,)
asignar	= , &	, , ; ,)
valor	id,munus,C, N.Entero , N. Real	;
mientras_hacer	dum	} , break , reditus
si_sino	si	} , break , reditus
C	autem,&	} , break , reditus
segun_hacer	per	} , break , reditus
lista_CASE	apud,&	default
CASE	apud	default
D	letra,digito	:
condición	quod,uel,&	)
condicion_AND	quod	)
condicion_OR	uel	)
E	id, (, N.Entero , N.Real	) , ; , ,
V	+, - , ^ , *,/,&	) , } , break, reditus, ;
H	)	) , } , break, reditus, + , - , ^,*,/
tipo_dato	TOT,BOOL,VOLI, REALI, CAR, BIN , OCT, HEX ,TORQ	id, (
Z	id,N.Entero,N.Real	operador_matematico ,) , ; , + , - , ^
operador_aritmetico	+, -, ^,*,/	id, (, N.Entero, N.Real
operador_logico	<= , >= ,!=, ==	id
M	TOT,BOOL,VOLI, REALI, CAR, BIN , OCT, HEX ,TORQ	id
atributos	*,&	Munus, }

OBJETIVOS DEL ANALIZADOR SEMÁNTICO

- Trabajar simultáneamente con el analizador sintáctico.
- Realizar comprobaciones de unicidad.
- Realizar comprobaciones de declaración de identificadores y reglas de ámbitos.
- Realizar comprobaciones de inferencia de tipo.
- Mostrar los TAS correspondientes a cada ámbito.
- Mostrar las clases y funciones de todo el programa.

Clases adicionales usadas (atributos y funciones):

AMBITO
-String nombreClase -String nombreFuncion +TASPRIMITIVO tasprim +TASCLASES tasclas +AMBITO subzona
+AMBITO () +mostrarAMBITO() +mostrarSUBZONA() //getters y setters +String getNombreClase() +void setNombreClase(String nombreClase) +String getNombreFuncion() +void setNombreFuncion(String nombreFuncion)

Definición de la clase:

Un ámbito es una zona donde un conjunto de variables tienen acción. El analizador maneja 2 ámbitos: uno para la clases y otro para las funciones(subzone).

Cada ámbito contiene un TAS primitivo y un TAS de objetos.

Por defecto, si la clase no tiene funciones, su subámbito es igual a **null.**

Cada vez que se crea una nueva clase, el ámbito más grande (de las clases) se actualiza y lo mismo ocurre con el ámbito más pequeño (de las funciones).

Descripción de las funciones:

AMBITO():

Constructor de la clase.

mostrarAMBITO():

Muestra el TAS primitivo y el TAS de Objetos de la última clase registrada.

mostrarSUBZONA():

Muestra el TAS primitivo y el TAS de Objetos de una función.

SEM
-String estado -String tempNombre -String tempTipo -float tempValor
+SEM() //getters y setters +String getEstado() +void setEstado(String estado) +String getTempNombre() +void setTempNombre(String tempNombre) +String getTempTipo() +void setTempTipo(String tempTipo) +float getTempValor() +void setTempValor(float tempValor)

Definición de la clase:

La clase SEM sirve como un asistente para realizar el análisis semántico. Su atributo principal es el **estado** ya que permite saber qué acción realizar cuando se elimina una Vt en el análisis sintático. También contiene variables temporales que apoyan en la asignación de valores a los identificadores primitivos, entre otras acciones.

Descripción de las funciones:

SEM():

Constructor de la clase.

REGTASOBJETOS
-String nombre -String tipo
+REGTASOBJETOS () +REGTASOBJETOS(String nombre, String tipo) //getters y setters +String getNombre() +void setNombre(String nombre) +String getTipo() +void setTipo(String tipo)

Definición de la clase:

Se puede entender como un objeto que pertenece al TAS de objetos. Tiene información de nombre y tipo. Donde el tipo es una clase.

Descripción de las funciones:

REGTASOBJETOS ():

Constructor de la clase.

REGTASOBJETOS (String nombre, String tipo):

Constructor con parámetros de la clase.

REGTASPRIM
-String nombre -String tipo -double valor
+REGTASPRIM() +REGTASPRIM(String nombre, String tipo, double valor) //getters y setters: +String getNombre() +void setNombre(String nombre) +String getTipo() +void setTipo(String tipo) +double getValor() +void setValor(double valor)

Definición de la clase:

Se puede entender como un identificador que pertenece al TAS primitivo. Tiene información de nombre, tipo y valor. Donde el tipo es un tipo de dato primitivo.

Por defecto, las variables numéricas se inicializan en cero.

Descripción de las funciones:

REGTASPRIM():

Constructor de la clase.

REGTASPRIM(String nombre, String tipo, double valor):

Constructor con parámetros de la clase.

MATRIZ_DE_CLASES_EXISTENTES
+String [][] matClases; +int n;
+ MATRIZ_DE_CLASES_EXISTENTES () +void nueva Clase (String nombre) +boolean existeClase(String nom) +void mostrar()

Definición de la clase:

Matriz que contiene todas las clases con sus respectivas funciones.

Descripción de las funciones:

MATRIZ _DE_CLASES_EXISTENTES():

Constructor de la clase.

void nuevaClase(String nombre):

Registra el nombre de la clase *pasada por parámetro* en la primera columna de la matriz. Esto servirá para, con ayuda de otras funciones, declarar objetos de dicha clase y evitar que se cree una clase con el mismo nombre.

boolean existeClase(String nom):

Busca el nombre de la clase *pasada por parámetro* en la matriz. Si lo encuentra, retorna **true,** de lo contrario, retorna **false.**

void mostrar():

Muestra la matriz.

TASOBJETOS
+ REGTASOBJETOS[] tasob +int n
+TASOBJETOS () +void nuevoObjeto(String nombre, String tipo,MATRIZ_DE_CLASES_EXISTENTES mat) +boolean existeEnTAS(String nom) +void mostrar()

Definición de la clase:

TAS que contiene los objetos declarados.

Descripción de las funciones:

TASCLASES():

Constructor de la clase.

void nuevoObjeto (String nombre, String tipo, MATRIZ _DE_CLASES_EXISTENTES vec):

Registra un objeto (nombre y tipo) en el TAS si cumple dos requisitos:

1. El tipo del objeto es una clase que ha sido creada anteriormente
2. El objeto no ha sido declarado anteriormente.

boolean existeEnTAS(String nom):

Busca el nombre del objeto *pasado por parámetro* en **tasc** (TAS de objetos). Si lo encuentra, retorna **true,** de lo contrario, retorna **false.**

void mostrar():

Muestra el TAS de forma ordenada en una tabla.

TASPRIMITIVO
+REGTASPRIM [] tasp; +int n;
+TASPRIMITIVO() +void nuevoID(String nombre, String tipo, double valor) +void modificarValor(String nombre,double valor) +int buscarID(String nom) +boolean existeEnTAS(String nom) + void mostrar()

Definición de la clase:

TAS que contiene los identificadores primitivos.

Descripción de las funciones:

TASPRIMITIVO():

Constructor de la clase.

nuevoID(String nombre, String tipo, double valor):

Registra una variable primitiva (nombre, tipo y valor) en el TAS si no ha sido declarado anteriormente.

boolean existeEnTAS(String nom):

Busca el nombre de la variable *pasada por parámetro* en **tasp** (TAS primitivo). Si lo encuentra, retorna **true,** de lo contrario, retorna **false.**

void mostrar ():

Muestra el TAS de forma ordenada en una tabla.

COMPROBACION DE TIPOS

COMPROBACION DE DECLARACION DE IDENTIFICADORES Y REGLAS DE AMBITOS

EJEMLPOS

CORRECTO

1. #publicus genus Gonzales {Principalem () {*TOT a=27;}}
2. #publicus genus Gonzales {Principalem () {*REALI a=27.56;}}
3. #publicus genus Pariona {Principalem () {*REALI a=27.56;}} publicus genus Huapaya {*Pariona aa;}

INCORECTO

1. #publicus genus Gonzales {Principalem () {a=27;}}
2. #publicus genus Gonzales {Principalem () {a=27.56;}}
3. #publicus genus Pariona {Principalem () {*REALI a=27.56;}} publicus genus Huapaya { aa;}

COMPROBACIONES DE UNICIDAD

EJEMPLOS

CORRECTO

1. #publicus genus Pipo {Principalem () {*TOT id, a; dum (id ==a) facite {si (id > id) {} autem {}}}}
2. #publicus genus carro {Principalem () {*TOT id;per (id) facio { default: {*carro a = 1;} break;} }}

INCORRECTO

1. #publicus genus Pipo {Principalem () {*TOT id, a; *VOLI a; dum (id ==a) facite {si (id > id) {} autem {}}}}
2. #publicus genus carro {Principalem () {*TOT id;per (id) facio { default: {*carro a = 1;} break;} }}

UNICIDAD DE FUNCIONES

CORRECTO

1. #publicus genus guerrero {Principalem(){} munus atacar vacuum(*TOT atk){} munus defender vacuum(*TOT def){}}

INCORRECTO

1. #publicus genus guerrero {Principalem(){} munus atacar vacuum(*TOT atk){} munus defender vacuum(*TOT def){} munus atacar vacuum(*REALI atk2){}}

La declaración de un identificador en un ámbito ha de ser única en multitud de lenguajes de programación.

COMPROBACIONES DE INFERENCIA DE TIPO

EJEMPLOS

CORRECTO

1. #publicus genus joseff {Principalem () {*TOT a; a=20;}}
2. #publicus genus joseff {Principalem () {*VOLI a; a=2.12+3.42*5.1;}}

INCORRECTO

1. #publicus genus joseff {Principalem () {*TOT a; a=20.1;}}
2. #publicus genus joseff {Principalem () {*TOT a; a=2.12+3.42*5.1;}}
3. #publicus genus aaa{Principalem(){*VOLI a=3.2;*TOT x=3;*TOT b=(((a)))+x;}}

Inferir el tipo de cada construcción del lenguaje. Para poder implementar la comprobación anterior, es necesario conocer el tipo de toda construcción sintácticamente válida del lenguaje. Así, el analizador semántico deberá aplicar las distintas reglas de inferencia de tipos descritas en la especificación del lenguaje de programación, para conocer el tipo de cada construcción del lenguaje.

Otras cadenas correctas usando estructuras de control:

dum - facite

#publicus genus aaa{Principalem(){*TOT a,b;a=12;b=1; dum (a > b) facite {a=a+1;}}}

Si - autem

#publicus genus joseff {Principalem (){*TOT aca,alla,hola,chau; si (aca == alla quod hola < chau){hola=alla;}autem {aca=chau; }}}

Per-facio

#publicus genus joseff {Principalem (){*TOT a,b,xx; per (xx) facio {default: {a=b*0.1;} break;}}}