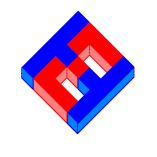
**UNIVERSIDAD NACIONAL DE INGENIERÍA**

**FACULTAD DE ELECTROTECNIA Y COMPUTACIÓN**

**PROYECTO COMPILADOR**

**HAMAT**

****

**AUTOR**

**HARVIN MANUEL TOLEDO POLANCO**

**/\* Segunda versión de programa triangulo**

**Realizado por Harvin Toledo Polanco**

**Fecha: domingo, 15 de abril de 2012**

**e-mail: harvintoledo@gmail.com**

**Ilustra algunos elementos del lenguaje Nohemi**

**Para proyecto Compilador HAMAT**

**\*/**

**programa triangulo;**

**principal()**

**{**

**real lado\_a, lado\_b, lado\_c, lado\_mayor, existe, falso, verdadero, tipo;**

**real suma;**

**/\* tipo=3 equilátero, tipo=2 isósceles; tipo =1 escaleno \*/**

**/\* booleana existe=falso; suponemos que triangulo no existe \*/**

**escribir "Ingrese los lados del triangulo\n";**

**escribir "ingrese lado a ";**

**leer lado\_a;**

**escribir "Ingrese lado b ";**

**leer lado\_b;**

**escribir "Ingrese lado c ";**

**leer lado\_c;**

**/\* si el triangulo es equilatero \*/**

**si (lado\_a == lado\_b && lado\_b == lado\_c**

**&& lado\_b== lado\_c)**

**{**

**existe=verdadero;**

**}**

**sino**

**/\* el triangulo es isósceles \*/**

**si (lado\_a == lado\_b || lado\_a == lado\_c ||**

**lado\_b == lado\_c)**

**{**

**tipo=2;**

**existe=verdadero;**

**}**

**/\* el triangulo es escaleno \*/**

**sino**

**{**

**si (lado\_a > lado\_b)**

**{**

**lado\_mayor = lado\_a;**

**}**

**sino**

**{**

**lado\_mayor = lado\_b;**

**}**

**si (lado\_c > lado\_mayor)**

**{**

**lado\_mayor = lado\_c;**

**}**

**/\* si uno de los lados es el mayor hacer la respectiva suma \*/**

**si (lado\_a == lado\_mayor)**

**{**

**suma = lado\_b + lado\_c;**

**}**

**sino**

**si (lado\_b == lado\_mayor)**

**{**

**suma = lado\_a + lado\_c;**

**}**

**sino**

**{**

**suma = lado\_a + lado\_b;**

**}**

**si (suma >= lado\_mayor)**

**{**

**tipo = 1;**

**existe = verdadero;**

**}**

**sino**

**{**

**tipo = 0;**

**existe = falso;**

**}**

**}**

**si( existe == verdadero )**

**{**

**escribir "El triangulo existe\n";**

**seleccionar( tipo )**

**{**

**caso 1:**

**escribir "El triangulo es escaleno\n";**

**interrumpir;**

**caso 2:**

**escribir "El triangulo es isóseles\n";**

**interrumpir;**

**caso 3:**

**escribir "El triángulo es equilátero\n";**

**interrumpir;**

**alcontrario: /\* este nunca debería de ocurrir \*/**

**escribir "No se puede clasificar\n";**

**}**

**}**

**sino**

**{**

**escribir "El triangulo no existe\n";**

**escribir "El triangulo no se puede clasificar\n";**

**}**

**}**

**/\* Ejemplo de herencia en el lenguaje Nohemi para el compilador HAMAT \*/**

**clase** felino {

**entero** **edad**;

**flotante** **peso**;

**flotante** **tamaño**;

**entero** **obtenerEdad**(){

**retornar** **este**.**edad**;

}

**flotante** **obtenerPeso**(){

**retornar** **este**.**Peso**;

}

**presentaEdad**() {

imprimir "Edad: " + **este**.**edad**;

}

}

**/\* Esta función es equivalente a la anterior \*/**

**felino** :: **presentaEdad**() {

**imprimir** "Edad: " + **este**.**edad**;

}

**clase** **leon** : **felino** {

**cadena** **habitad**;

}

**NOTACION BNF PARA LENGUAJE NOHEMI**

**ANÁLISIS SINTÁCTICO DESCENDENTE RECURSIVO PARA LENGUAJE NOHEMI**

**GRÁMATICA MODIFICADA DEL LENGUAJE NOHEMI**

1. Programa-> **programa** ID;

lista-declaración

2.lista-declaracion-> lista-declaración declaración | declaración

3. declaración-> variable-declaración | función-declaración | clase-declaración

4. variable-declaración -> tipo-especificación ID; | tipo-especificación ID[NUM];

4. variable-declaración ->

**Var** variable-declaración tipo-especificación variables, | variables

5. tipo-especificación -> **entero | real | cadena | carácter**

6. función-declaración -> tipo-especificación ID (parámetros)

sentencia-compuesta

6.1 clase-declaracion -> modificador-global **clase** cuerpo-clase

6.01 clase-declaración -> clase-declaración cuerpo-clase | cuerpo-clase

6.1 cuerpo-clase-> función-declaración | procedimiento-declaración | variable-declaración | clase-declaración

modificador-global -> **acceso protegido | acceso privado | acceso publico**

cuerpo-clase -> { lista-clase cuerpo-clase | lista-clase }

lista-clase -> lista-declaración | función-declaración

7. parámetros -> lista-parámetros | **void**

8. lista-parámetros -> lista-parámetros, parámetro | parámetro

9. parámetro -> tipo-especificación ID | tipo-especificación ID[]

10. sentencia-compuesta -> **{**declaración-local secuencia-de-sentencia}

11. declaración-local -> declaración-local variable-declaración | vacío

12. secuencia-de-sentencia -> secuencia-de-sentencia sentencia | vacio

13. secuencia-de-sentencia ->

Expresión-sentencia | sentencia-compuesta| retornar-sentencia |

| si-sentencia | sentencia-seleccionar | caso-sentencia | repetir-sentencia | asignación-sentencia | leer-sentencia | escribir-sentencia | para-sentencia | mientras-sentencia | hacer-mientras-sentencia | identificador

| carácter-sentencia | entero-sentencia | real-sentencia | cadena-sentencia

14. Expresión-sentencia -> expresión; | ;

15. retornar-sentencia -> **retornar** ; | **retornar** expresión;

16. expresión -> variable = expresión | expresión-simple

17. variable -> ID | ID [NUMERO]

18. si\_sentencia ->

**si (expresion)**

secuencia\_de\_sentencia;

**[ sino** secuencia\_de\_sentencia;]

19. sentencia\_seleccionar ->

**seleccionar (expresion)**

{[ caso\_sentencia ]+

[**alcontrario**: secuencia\_de\_sentencia;]}

20. caso\_sentencia ->

**caso expresion:** secuencia\_de\_sentencia;

**interrumpir;**

21. repetir\_sentencia ->

**repetir**

secuencia\_de\_sentencia;

**hasta ( expresion);**

22. leer\_sentencia ->

**leer identificador {, identificador};**

23. escribir\_sentencia ->

**escribir expresion;**

24. para\_sentencia ->

**para (asignacion\_sentencia; expresion; operación)**

**secuencia\_de\_sentencia**

25. mientras\_sentencia ->

**mientras (expresion)**

**secuencia\_de\_sentencia**

26. hacer\_mientras\_sentencia ->

**hacer**

**secuencia\_de\_sentencia**

**mientras (expresion);**

27. expresión\_logica -> expresion\_comparacion {operacion\_logica expresion\_comparacion }

28. operación\_logica –> && | ||

29. expresión-simple -> expresión-aditiva operación\_comparación expresión-aditiva | expresión-aditiva

30. operación\_comparación -> **==** | **<>** | **<** | **>** | **>=** | **<=**

31. expresión-aditiva -> expresión-aditiva operación\_suma término | término

32. operación\_suma -> **+** | **-**

33. término -> término operación\_multiplicación factor

34. operación\_multiplicación -> **\*** | **/**

35. factor -> ( expresion ) | variable | llamada | NUM

36. llamada -> ID ( argumentos )

37. argumentos -> lista-argumentos | vacio

38. lista-argumentos -> lista-argumentos, expresión | expresion

**Nuevas gramáticas**

expresión-simple->

expresión-simple expresión-comparación logop expresión-comparación

| expresión-comparación

Logop -> **&&** | **||**

expresión-comparación ->

expresión-comparación expresión-aditiva compop expresión-aditiva |

expresión-aditiva

compop -> **<=** | **<** | **>** | **>=** | **==** | **¡=**

expresión-aditiva ->

expresion-aditiva termino adiop termino | termino

adiop -> **+** | **-**

función-declaración ->

tipo-especificación ID ( parámetros ) sentencias-compuesta

tipo-especificación ->

**entero | real | bit | texto**

**Documentación para uso del lenguaje Nohemí**

**Identificadores**

**Los identificadores como ya es conocido son palabras que nos sirven para usar variables dentro de un programa.**

**En este lenguaje se pueden usar tanto mayúsculas como minúsculas en**

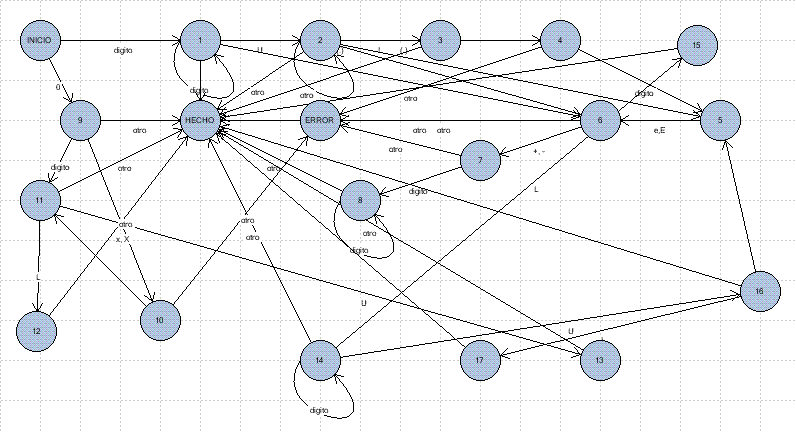
**Palabras reservadas**

**Como las palabras reservadas son un subconjunto de los identificadores, estos tienen un propósito específico dentro del lenguaje Nohemí.**

**Las palabras reservadas son las siguientes: mientras, repetir, hasta, hacer, si, para, entero, real, carácter, cadena,**

**Constantes numericas**

**Para generar las posibles producciones numericas dentro del lenguaje nohemi se uso el siguiente automata:**

****

**Números en punto flotante:**

**Documentación del programa**

Traducción en lenguaje ensamblador

El ambiente de ejecución del lenguaje Nohemí será en

El compilador **HAMAT** traducirá el lenguaje de alto nivel a ensamblador utilizando la arquitectura del microprocesador 8086.

Ej:

**int** x, y, z;

x=3;

y=5;

z=7;

x=x\*y+z;

MOV [1000], 3

MOV [1001], 5

MOV [1002], 7

MOV AX, [1000]

MUL AX, [1001]

ADD AX, [1002]

MOV [1000], AX

**int** x, i;

x=0;

i=0;

**para**(i=0; i < 10; i++)

x=x+i\*i;

MOV[1000], 0

MOV[1001], 0

LOOP: MOV AX, [1000]

MOV BX, [1001]

MUL BX, BX

ADD AX, BX

MOV [1000], AX

INC [1000]

CMP [1000], 10

JLE LOOP

**BIBLIOGRAFIA**

**Kenneth C. Louden (2004); Construcción de compiladores Principios y Práctica; ( Editorial Thomson )**

**Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman; Compiladores principos, técnicas y herramientas (Editorial Pearson Addison Wesley, Segunda Edición)**

**Byron S. Gottfried; Programacion en C; Editorial MC Graw Hill;**